Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Ακαδημαϊκό Έτος: 2020-2021

Αναγνώριση Προτύπων

1η Εργαστηριακή Άσκηση

Θέμα: Οπτική Αναγνώριση Ψηφίων

Τζε Χριστίνα-Ουρανία | 03116079
Ψαρουδάκης Ανδρέας | 03116001

9 Νοεμβρίου 2020

Περιεχόμενα

Εισαγωγή

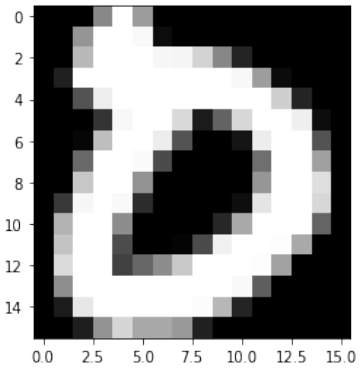
Σκοπός της παρούσας εργαστηριακής άσκησης είναι η υλοποίηση ενός συστήματος οπτικής αναγνώρισης ψηφίων. Τα δεδομένα προέρχονται από την US Postal Service (γραμμένα στο χέρι σε ταχυδρομικούς φακέλους και σκαναρισμένα) και περιέχουν τα ψηφία από το 0 έως το 9 και διακρίνονται σε train και test. Τα δεδομένα κάθε αρχείου αναπαριστούν τα περιεχόμενα ενός πίνακα (οι τιμες των στοιχείων του πίνακα διαχωρίζονται με κενό). Κάθε γραμμή αφορά ένα ψηφίο (δείγμα). Οι στήλες αντιστοιχούν στα χαρακτηριστικά (features) που περιγράφουν τα ψηφία. Στόχος είναι η δημιουργία και αποτίμηση (evaluation) ταξινομητών οι οποίοι θα ταξινομούν κάθε ένα απο τα ψηφία που περιλαμβάνονται στα test δεδομένα σε μία από τις δέκα κατηγορίες (από το 0 έως το 9).

Βήμα 1: Διάβασμα δεδομένων εκπαίδευσης και ελέγχου

Διαβάζουμε τα δεδομένα εκπαίδευσης και δοκιμής από τα αρχεία **train.txt** και **test.txt** αντίστοιχα. Τα δεδομένα θέλουμε να είναι σε μορφή συμβατή με το **scikit-learn**. Φροντίζουμε λοιπόν με κατάλληλο slicing ο πίνακας **X_train** να περιέχει τα δείγματα εκπαίδευσης (χωρίς τα labels) και να είναι διάστασης **n_samples_train** $\times$ **n_features** ενώ ο **y_train** να είναι ένας μονοδιάστατος πίνακας μήκους **n_samples** και να περιέχει τα αντίστοιχα **labels** για τον **X_train**. Ακριβώς αντίστοιχα δημιουργούμε και τους πίνακες **X_test** και **y_test** για τα δεδομένα δοκιμής.

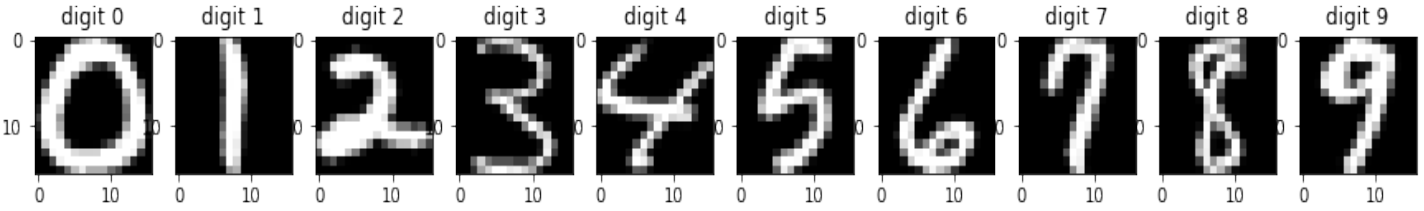
Βήμα 2: Απεικόνιση του υπ' αριθμόν 131 ψηφίου

Ορίζουμε τη συνάρτηση show_sample η οποία δέχεται ως όρισμα ένα σύνολο δεδομένων X και έναν δείκτη index ενώ επιστρέφει μια εικόνα 16 $\times$ 16 του ψηφίου που αντιστοιχεί στη θέση index του συνόλου X. Αξιοποιούμε τη συνάρτηση **reshape** της numpy για να οργανώσουμε τα 256 χαρακτηριστικά σε ένα πίνακα 16 $\times$ 16 ενώ με τη χρήση της **imshow** της matplotlib απεικονίζουμε την τελική εικόνα. Για την απεικόνιση του υπ΄ αριθμόν 131 ψηφίου των δεδομένων εκπαίδευσης καλούμε την συνάρτηση **show_sample** με ορίσματα τον πίνακα X_train και τον δείκτη 130 (επειδή η αρίθμηση ξεκινάει από το μηδέν).



Βήμα 3: Απεικόνιση ενός τυχαίου δείγματος από κάθε κλάση

Ορίζουμε τώρα τη συνάρτηση **plot_digits_samples** η οποία δέχεται ως όρισμα ένα σύνολο δεδομένων X και ένα μονοδιάστατο πίνακα y που περιέχει τα labels του συνόλου . Για κάθε ένα από τα 10 ψηφία (0,1,2,...,9) η συνάρτηση εντοπίζει τα indexes (θέσεις) που αυτό εμφανίζεται στον πίνακα y και τα αποθηκεύει σε έναν πίνακα. Αυτό πραγματοποιείται μέσω της συνάρτησης **argwhere** της βιβλιοθήκης numpy. Στη συνέχεια, από τον πίνακα αυτό επιλέγεται τυχαία μια τιμή (ένα index) καθώς μας ενδιαφέρει απλά ένα τυχαίο δείγμα από κάθε label. Αυτό πραγματοποιείται με τη βοήθεια της συνάρτησης **random.choice**. Μέσω της τιμής αυτής επιλέγεται η αντίστοιχη γραμμή του πίνακα X στην οποία είναι αποθηκευμένο το εν λόγω ψηφίο. Αξιοποιούμε τη συνάρτηση **reshape** της numpy για να οργανώσουμε τα 256 χαρακτηριστικά του δείγματος σε ένα πίνακα 16×16 και έπειτα απεικονίζουμε την τελική εικόνα σε ένα subplot. Τελικά η συνάρτηση επιστρέφει ένα subplot που περιέχει τυχαίες απεικονίσεις για κάθε ένα από τα 10 ψηφία.



Βήμα 4: Μέση τιμή χαρακτηριστικών του pixel (10,10) για το ψηφίο 0

Θέλουμε τώρα να υπολογίσουμε τη μέση τιμή των χαρακτηριστικών του pixel (10,10) για το ψηφίο '0' με βάση τα train δεδομένα.
Για το σκοπό αυτό ορίζουμε τη συνάρτηση **digit_mean_at_pixel** η οποία δέχεται τέσσερα ορίσματα:

- ένα σύνολο δεδομένων X
- ένα μονοδιάστατο πίνακα y με τα αντίστοιχα labels του
- ένα δεδομένο ψηφίο digit
- ένα tuple με τις συντεταγμένες του pixel, οι οποίες έχουν default τιμή $(10, 10)$

Η συνάρτηση αρχικά υπολογίζει το χαρακτηριστικό (0-255) που αντιστοιχεί στη θέση του pixel. Αυτό πραγματοποιείται πολλαπλασιάζοντας με 16 (σύνολο στηλών του πίνακα) το σύνολο όλων των γραμμών που υπάρχουν πριν από τη γραμμή που βρίσκεται το pixel και αθροίζοντας στο αποτέλεσμα το σύνολο των στηλών που βρίσκονται πριν από τη στήλη του pixel. Ο αριθμός αυτός μας υποδεικνύει τη στήλη του πίνακα X που πρόκειται να εξετάσουμε. Μέσω της συνάρτησης **argwhere** της βιβλιοθήκης numpy εντοπίζουμε τις θέσεις του πίνακα y που περιέχουν το ψηφίο digit, και κατά συνέπεια τις γραμμές του πίνακα X που αφορούν το εν λόγω

ψηφίο. Έτσι, αξιοποιώντας τώρα τη συνάρτηση **mean** της numpy λαμβάνουμε τη μέση τιμή όλων των τιμών που βρίσκονται στις γραμμές αυτές και στην προαναφερθείσα στήλη.

Για να υπολογίσουμε λοιπόν τη μέση τιμή των χαρακτηριστικών του pixel $(10, 10)$ για το ψηφίο '0' με βάση τα train δεδομένα καλούμε τη συνάρτηση **digit_mean_at_pixel** με ορίσματα: **X_train**, **y_train**, **0**, **(10, 10)**. Τελικά λαμβάνουμε μέση τιμή ίση με: **−0.5042**

Βήμα 5: Διασπορά χαρακτηριστικών του pixel (10,10) για το ψηφίο 0

Θέλουμε τώρα να υπολογίσουμε τη διασπορά των χαρακτηριστικών του pixel (10,10) για το ψηφίο '0' με βάση τα train δεδομένα. Για το σκοπό αυτό ορίζουμε,με ακριβώς αντίστοιχο τρόπο με το Βήμα 4, τη συνάρτηση **digit_variance_at_pixel**. Η μοναδική διαφορά της σε σχέση με την **digit_mean_at_pixel** (του προηγούμενο βήματος) είναι ότι, σαν τελευταίο βήμα, αξιοποιεί τη συνάρτηση var της βιβλιοθήκης numpy (αντί της numpy.mean) για τον υπολογισμό της διασποράς μεταξύ των ζητούμενων χαρακτηριστικών.

Για να υπολογίσουμε λοιπόν τη διασπορά των χαρακτηριστικών του pixel $(10, 10)$ για το ψηφίο '0' με βάση τα train δεδομένα καλούμε τη συνάρτηση **digit_variance_at_pixel** με ορίσματα: **X_train**, **y_train**, **0**, **(10, 10)**. Τελικά λαμβάνουμε διασπορά ίση με: **0.5245**

Βήμα 6: Μέση τιμή και διασπορά χαρακτηριστικών κάθε pixel για το 0

Σε αυτό το βήμα θέλουμε να υπολογίσουμε τη μέση τιμή και διασπορά των χαρακτηριστικών κάθε pixel για το ψηφίο '0' με βάση τα train δεδομένα. Ακολουθούμε ακριβώς την διαδικασία των βημάτων 4 και 5, ωστόσο τώρα δεν προσδιορίζουμε κάποιο συγκεκριμένο pixel.

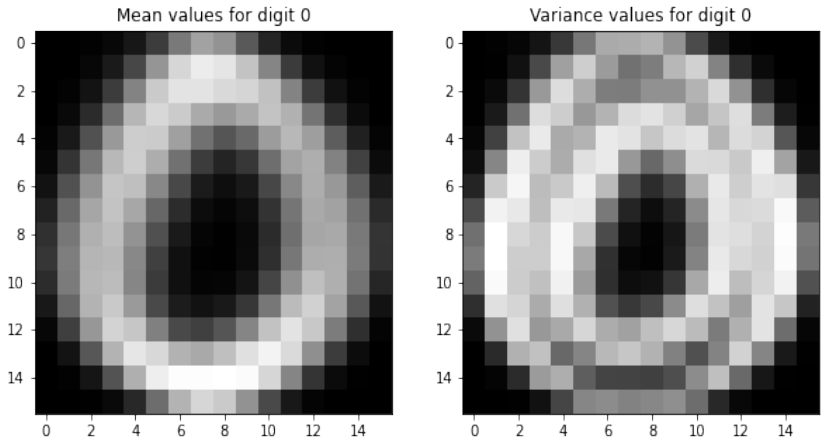
Έτσι, ορίζουμε τις συναρτήσεις **digit_mean** και **digit_variance**, σε αντιστοιχία με τις digit_mean_at_pixel και digit_var_at_pixel. Απλώς τώρα οι νέες συναρτήσεις δέχονται τρία μόνο ορίσματα:

- ένα σύνολο δεδομένων X
- ένα μονοδιάστατο πίνακα y με τα αντίστοιχα labels του X
- ένα δεδομένο ψηφίο digit

Μέσω της συνάρτησης **argwhere** της βιβλιοθήκης numpy εντοπίζουμε τις θέσεις του πίνακα y που περιέχουν το ψηφίο digit, και κατά συνέπεια τις γραμμές του πίνακα X που αφορούν το εν λόγω ψηφίο. Στη συνέχεια, αξιοποιώντας τις συναρτήσεις **mean** και **var** της numpy λαμβάνουμε τη μέση τιμή και διασπορά μεταξύ των γραμμών αυτών. Εδώ δηλαδή δεν εξετάζουμε κάποια συγκεκριμένη στήλη του πίνακα X αλλά υπολογίζουμε μέση τιμή και διασπορά όλων των χαρακτηριστικών, οπότε το τελικό αποτέλεσμα είναι ένας μονοδιάστατος πίνακας 256 στοιχείων.

Βήματα 7,8: Απεικόνιση ψηφίου 0 με βάση τη μέση τιμή και διασπορά του

Σχεδιάζουμε το ψηφίο '0' χρησιμοποιώντας τις τιμές της μέσης τιμής και της διασποράς που υπολογίσαμε στο Βήμα 6. Για την απεικόνιση χρησιμοποιούμε την συνάρτηση **imshow** αφού μετασχηματίσουμε πρώτα τα διανύσματα mean_values και variance_values σε πίνακες διαστάσεων 16×16.



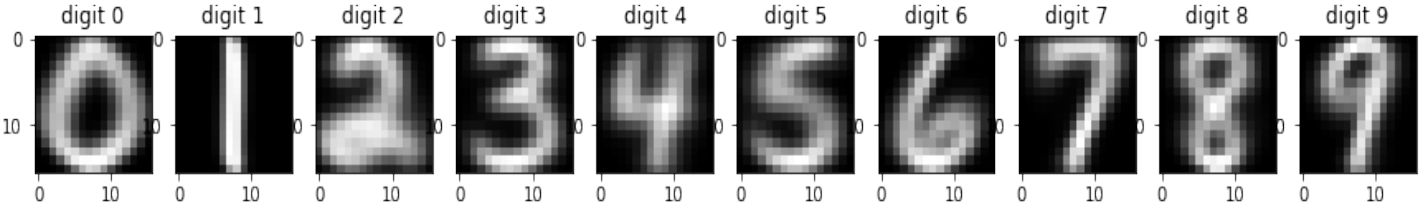
Σχολιασμός

Συγκρίνοντας τις δύο παραπάνω εικόνες, συμπεραίνουμε πως το σχήμα του ψηφίου '0' είναι ορατό και στις δύο περιπτώσεις. Παρατηρούμε ωστόσο, πως η απεικόνισή του με βάση τη μέση τιμή προσεγγίζει το ψηφίο με μεγαλύτερη λεπτομέρεια και ακρίβεια. Αυτό οφείλεται στον ορισμό του mean, το οποίο μας επιστρέφει τη μέση εικόνα μεταξύ όλων των δειγμάτων εκπαίδευσης. Επίσης, είναι φανερό πως η εκτίμηση του ψηφίου μέσω της διακύμανσής του, δίνει σαν αποτέλεσμα μια παχύτερη και ευρύτερη αναπαράσταση, σε σχέση με την πιο συμπαγή που προκύπτει χρησιμοποιώντας την μέση τιμή. Αν επομένως εστιάσουμε στο περίγραμμα κάθε ψηφίου, βλέπουμε πως η διακύμανση λαμβάνει μεγάλες τιμές και προσεγγίζει το λευκό χρώμα, σε αντίθεση με τη μέση τιμή που εμφανίζει χαμηλές τιμές και προσεγγίζει το μαύρο. Αυτό είναι αναμενόμενο αν σκεφτεί κανείς πως τα περιγράμματα των ψηφίων του συνόλου εκπαίδευσης μπορεί να είναι λιγότερο ή περισσότερο έντονα καθώς επίσης και να διαφέρουν ως προς την καμπυλότητα σχεδίασης, ανάλογα με τον γραφικό χαρακτήρα κάθε χρήστη.
Συγκεφαλαιώνοντας, μπορούμε να θεωρήσουμε την **απεικόνισης της μέσης τιμής** ως μια πιο **αξιόπιστη αναπαράσταση**.

Βήμα 9: Απεικόνιση κάθε ψηφίου με βάση τη μέση τιμή του

Υπολογίζουμε τη μέση τιμή και διασπορά των χαρακτηριστικών για όλα τα ψηφία (0-9) με βάση τα train δεδομένα. Συγκεκριμένα, αυτό γίνεται με χρήση των συναρτήσεων **digit_mean** και **digit_variance** ενώ τα αποτελέσματα αποθηκεύονται στους πίνακες means και variances αντίστοιχα, διαστάσεων 10×256 ο καθένας. Ταυτόχρονα, αναπαριστούμε κάθε ψηφίο

χρησιμοποιώντας τις 256 τιμές της μέσης τιμής με τρόπο ανάλογο του βήματος 7.



Βήμα 10: Ταξινόμηση υπ'αριθμόν 101 ψηφίου βάσει ευκλείδειας απόστασης

Στο βήμα αυτό ταξινομούμε το υπ'αριθμόν 101 ψηφίο των test δεδομένων (βρίσκεται στη γραμμή 100 του πίνακα X_test αφού η αρίθμηση ξεκινάει από το μηδέν) σε μία από τις 10 κατηγορίες βάσει της Ευκλείδειας απόστασης. Για τον σκοπό αυτό ορίζουμε τη συνάρτηση **euclidean_distance** η οποία δέχεται σαν ορίσματα ένα δείγμα 256 χαρακτηριστικών και ένα διάνυσμα που αντιπροσωπεύει τη μέση τιμή ενός ψηφίου ενώ υπολογίζει την ευκλείδεια απόστασή τους. Αυτό γίνεται με χρήση της συνάρτησης **linalg.norm** της numpy.

Για την ταξινόμηση ενός ψηφίου υπολογίζουμε, με χρήση της συνάρτησης **euclidean_distance**, την ευκλείδεια απόστασή του από τα διανύσματα των μέσων τιμών και των 10 ψηφίων/κατηγοριών. Οι αποστάσεις αυτές αποθηκεύονται σειριακά σε μία λίστα. Το δείγμα ταξινομείται τελικά στην κλάση εκείνη από της οποίας το μέσο όρο απείχε λιγότερο. Για τον σκοπό αυτό βρίσκουμε την θέση στην οποία η λίστα με τις ευκλείδειες αποστάσεις παίρνει την ελάχιστη τιμή της.

Τελικά, το 101 ψηφίο εκτιμάται ότι ανήκει στην κατηγορία '0'. Για να αποφανθούμε σχετικά με το αν η ταξινόμηση είναι επιτυχής ή όχι συγκρίνουμε την εκτίμηση του ταξινομητή με το αντίστοιχο label του ψηφίου (βρίσκεται στη θέση 100, ξεκινώντας την αρίθμηση από το μηδέν, του διανύσματος y_test). Παρατηρούμε πως η τιμή της εκτίμησης συμπίπτει με αυτήν του ground truth, από όπου και συμπεραίνουμε πως η ταξινόμηση είναι επιτυχής.

Βήμα 11: Ταξινόμηση των test δεδομένων και υπολογισμός accuracy

Ταξινομούμε τώρα όλα τα ψηφία των test δεδομένων σε μία από τις 10 κατηγορίες με βάση την Ευκλείδεια απόσταση. Ορίζουμε την συνάρτηση **euclidean_distance_classifier** η οποία δέχεται σαν ορίσματα έναν πίνακα X με τα δείγματα που θέλουμε να ταξινομήσουμε και τον πίνακα X_mean με τις μέσες τιμές και των 10 ψηφίων. Επιστρέφει με τρόπο ανάλογο του βήματος 10 ένα διάνυσμα διαστάσεων όσο ο αριθμός των δειγμάτων όπου το κάθε στοιχείο του περιέχει την κατηγορία στην οποία ταξινομείται τελικά το αντίστοιχο δείγμα.
Για τον υπολογισμό του ποσοστού επιτυχίας αρχικοποιούμε μία μεταβλητή την οποία αυξάνουμε κάθε φορά κατά ένα όταν η πρόβλεψη του Ευκλείδειου ταξινομητή συμπίπτει με το label ενός

δείγματος (ground truth). Αφού διατρέξουμε όλα τα δεδομένα του συνόλου δοκιμής, το ποσοστό επιτυχίας του ταξινομητή προκύπτει διαιρώντας την τιμή της μεταβλητής αυτής με τον συνολικό αριθμό των δειγμάτων.

Τελικά, το **ποσοστό επιτυχίας** είναι **81.42 %** το οποίο είναι αρκετά υψηλό λαμβάνοντας υπόψιν την απλότητα της μεθόδου.

Βήμα 12: Υλοποίηση Ευκλείδειου ταξινομητή σαν ένα scikit-learn estimator

Στο βήμα αυτό υλοποιούμε τον ταξινομητή ευκλείδειας απόστασης σαν ένα **scikit-learn estimator**. Συγκεκριμένα ορίζουμε την κλάση **EuclideanDistanceClassifier** η οποία κληρονομεί από τις **BaseEstimator** και **ClassifierMixin**. Ο κατασκευαστής της κλάσης (__**init**__) αρχικοποιεί τον πίνακα X_mean_ των μέσων τιμών όλων των ψηφίων (0,1,...,9) και ορίζει τον αριθμό των διαθέσιμων κατηγοριών. Στη συνέχεια υλοποιούμε τις ακόλουθες μεθόδους:

- **fit**: Δέχεται σαν ορίσματα το train set X μαζί με τα αντίστοιχα labels y και υπολογίζει τον πίνακα X_mean_ των μέσων τιμών όλων των κατηγοριών. Για την εύρεση του διανύσματος της μέσης τιμής μιας κατηγορίας βρίσκουμε αρχικά τις θέσεις όλων των δειγμάτων των οποίων το label ισούται με την εν λόγω κατηγορία. Στη συνέχεια διατρέχουμε τα feature vectors των δειγμάτων αυτών και με χρήση της συνάρτησης **mean** της numpy υπολογίζουμε το διάνυσμα της μέσης τιμής αυτής της κατηγορίας. Το διάνυσμα αυτό αποθηκεύεται στον πίνακα X_mean_. Η διαδικασία αυτή επαναλαμβάνεται για κάθε κατηγορία ώστε να συμπληρωθεί ο πίνακας X_mean_.
- **predict**: Δέχεται σαν όρισμα το test set X και για κάθε δείγμα σε αυτό εκτιμάει την κατηγορία στην οποία ανήκει, επιλέγοντας αυτήν από της οποίας το μέσο όρο απέχει λιγότερο. Η συνάρτηση επιστρέφει ένα διάνυσμα με τις εκτιμήσεις για όλα τα test δεδομένα.
- **score**: Δέχεται σαν όρισμα το test set X μαζί με τις αντίστοιχες επισημειώσεις y. Καλεί την μέθοδο predict και αναθέτει σε κάθε δείγμα ένα label $\hat{y}$. Για τον υπολογισμό του accuracy score διαιρεί τον αριθμό των σωστών προβλέψεων (εκείνες για τις οποίες το $\hat{y}$ συμπίπτει με το y) με τον συνολικό αριθμό των δειγμάτων του συνόλου δοκιμής.

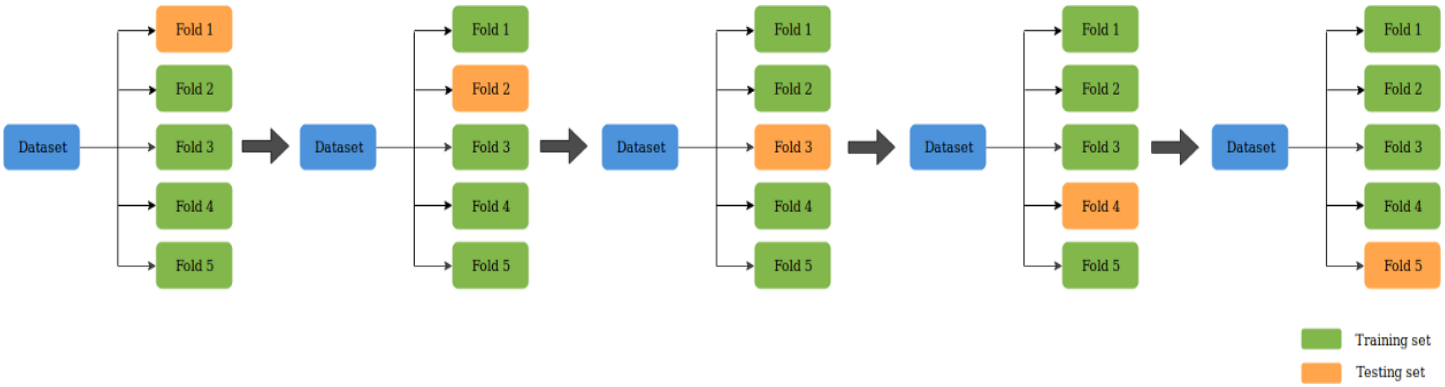
Βήμα 13: score Ευκλείδειου ταξινομητή, περιοχή αποφάσης, learning curve

(α′) Στο βήμα αυτό υπολογίζουμε το score του Ευκλείδειου Ταξινομητή με χρήση **5-fold-cross-validation**. Συγκεκριμένα, ορίζουμε τη συνάρτηση **evaluate_classifier** η οποία δέχεται τα ακόλουθα ορίσματα:

- ένα μοντέλο ταξινόμησης clf

- ένα σύνολο δεδομένων X : προκύπτει από την συνένωση με χρήση της συνάρτησης **concatenate** της numpy των αρχικών δεδομένων εκπαίδευσης με τα δεδομένα δοκιμής
- τα αντίστοιχα labels y : προκύπτουν από συνένωση των labels των δεδομένων εκπαίδευσης με τα labels των δεδομένων δοκιμής.
- τον αριθμό των folds : καθορίζει τον αριθμό των διαμερίσεων

Η συνάρτηση διαιρεί το σύνολο δεδομένων X (ανίστοιχα και το σύνολο y) σε 5 μέρη. Από αυτά το ένα fold επιλέγεται κάθε φορά σαν σύνολο δοκιμής και τα υπόλοιπα χρησιμοποιούνται για την εκπαίδευση του μοντέλου. Η διαδικασία αυτή επαναλαμβάνεται κυκλικά (το test set είναι κάθε φορά διαφορετικό) μέχρι να ολοκληρωθούν και οι πέντε επαναλήψεις.



Σε κάθε επανάληψη προσθέτουμε σε μία μεταβλητή, την οποία αρχικοποιούμε στο μηδέν, τα επιμέρους scores και στο τέλος διαιρούμε με τον συνολικό αριθμό των διαμερίσεων (folds=5). Με τον τρόπο αυτό λαμβάνουμε το **5-fold-cross-validation score**.

Στην περίπτωση του **Ευκλείδειου ταξινομητή** αυτό προκύπτει ίσο με **84.07 %**. Για λόγους σύγκρισης υπολογίζουμε το αντίστοιχο score με χρήση της έτοιμης συνάρτησης **cross_validate** του **scikit-learn**. Το ποσοστό που λαμβάνουμε είναι **84.19 %**.

Σχολιασμός

Παρατηρούμε αρχικά πως τόσο η δική μας υλοποίηση όσο και η έτοιμη του Scikit-learn δίνουν παρόμοια ποσοστά. Στην περίπτωση του cross validation μπορούμε να είμαστε πιο σίγουροι για την απόδοση του μοντέλου μας καθώς αυτό αξιολογείται σε πέντε διαφορετικά σύνολα δοκιμής. Αντιθέτως, όταν αξιολογούμε το μοντέλο σε ένα test set λαμβάνουμε μόνο μία μετρική απόδοσης η οποία δεν είναι πλήρως αντιπροσωπευτική. Το αποτέλεσμα αυτό μπορεί να είναι τυχαίο ή να επηρεάζεται από bias των test δεδομένων. Επομένως, με χρήση του cross validation επιτυγχάνουμε μία μεγαλύτερη συνοχή και ισορροπία στα δεδομένα μας και κατά συνέπεια εξασφαλίζουμε ένα μοντέλο που μπορεί να λύνει πιο αποτελεσματικά το γενικό πρόβλημα κατηγοριοποίησης.

(β′) Ο Ευκλείδειος ταξινομητής υπολογίζει αποστάσεις ανάμεσα σε διανύσματα 256 διαστάσεων το καθένα. Επομένως, είναι αδύνατον να τα οπτικοποιήσουμε σε ένα δισδιάστατο επίπεδο και πρέπει να λάβουμε υπόψιν μία μέθοδο η οποία μειώνει την διάσταση των διαθέσιμων features. Μερικές από αυτές είναι οι ακόλουθες:

- Dimensionality Reduction (πχ. PCA, t-SNE)
- Feature Selection

Δύο από τις πιο δημοφιλείς τεχνικές dimensionality reduction για οπτικοποίηση δεδομένων υψηλής διάστασης είναι οι PCA και t-SNE. Οι κυριότερες διαφορές τους συνοψίζονται στον ακόλουθο πίνακα:

PCA	**t-SNE**
Γραμμική τεχνική μείωσης διαστάσεων	Μη γραμμική τεχνική μείωσης διαστάσεων
Δεν περιλαμβάνει υπερπαραμέτρους	Περιλαμβάνει υπερπαραμέτρους (π.χ ρυθμός εκμάθηση, αριθμός βημάτων)
Επηρεάζεται έντονα από μη αναμενόμενα δεδομένα (outliers)	Μπορεί να διαχειριστεί μη αναμενόμενα δεδομένα (outliers)
Ντετερμινιστικός αλγόριθμος	Μη ντετερμινιστικός/τυχαιοποιημένος αλγόριθμος

Με βάση τα παραπάνω, καταλήγουμε στο ότι ο **t-SNE** υπερτερεί έναντι του **PCA** και για τον λόγο αυτό επιλέγουμε να χρησιμοποιήσουμε αυτόν. Ωστόσο, σύμφωνα με το documentation της **scikit-learn**, όταν ο αριθμός των διαστάσεων είναι αρκετά υψηλός, συνίσταται η πρότερη χρήση μιας άλλης τεχνικής για την μείωση των διαστάσεων σε έναν λογικό αριθμό, πχ. 50. Αυτό γίνεται με σκοπό την επιτάχυνση της υπολογιστικής διαδικασίας και την απομάκρυνση τμήματος θορύβου. Η επιπρόσθετη τεχνική μείωσης διαστατικότητας που χρησιμοποιούμε είναι η **TruncatedSVD**.

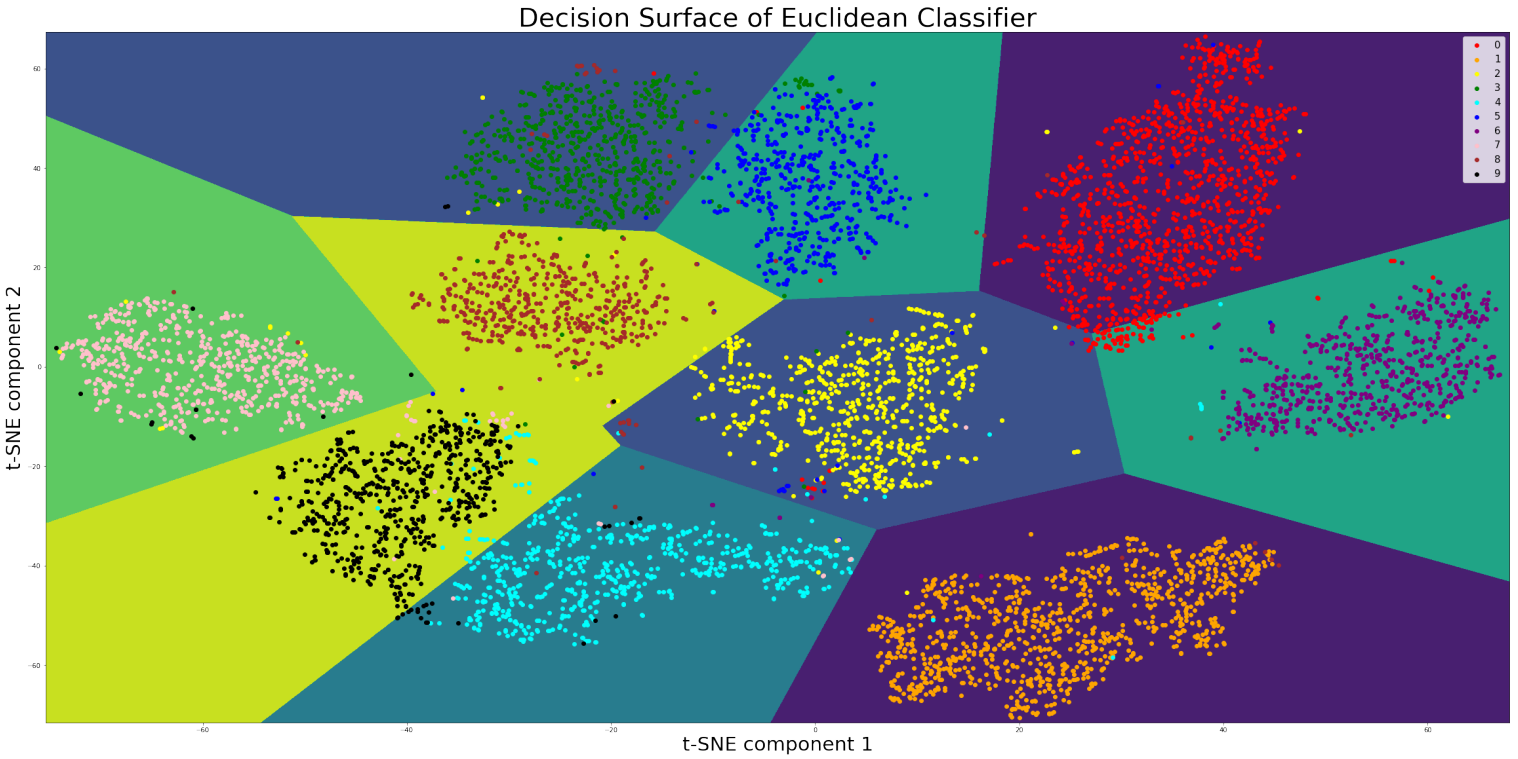
Συνοψίζοντας όλα τα παραπάνω, οι αλγόριθμοι που χρησιμοποιούμε είναι ο **TruncatedSVD** για την μείωση των διαστάσεων από 256 σε 50 και ο **t-SNE** για την τελική μείωση των συνιστωσών σε 2.

Περιοχές απόφασης Ευκλείδειου ταξινομητή

Σε γενικές γραμμές, ένας ταξινομητής χωρίζει το χώρο των χαρακτηριστικών σε τμήματα που ονομάζονται περιοχές αποφάσεων (decision boundaries). Όλα τα διανύσματα χαρακτηριστικών σε μια περιοχή αντιστοιχίζονται στην ίδια κατηγορία. Οι περιοχές αποφάσεων διαχωρίζονται από επιφάνειες που ονομάζονται όρια αποφάσεων. Αυτές οι

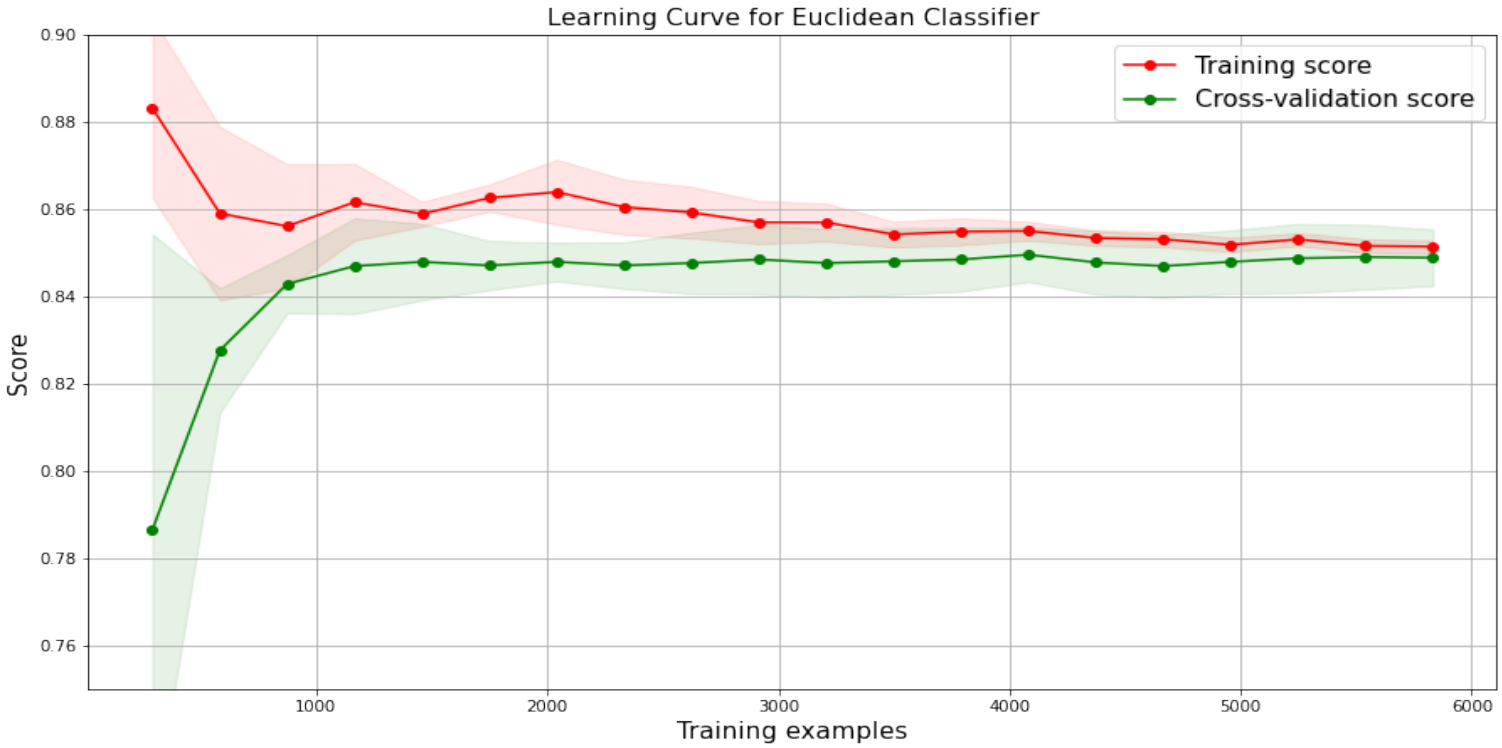
διαχωριστικές επιφάνειες αντιπροσωπεύουν σημεία όπου υπάρχουν 'ισοπαλίες' μεταξύ δύο ή περισσότερων κατηγοριών.

Για τον σχεδιασμό της περιοχής απόφασης του Ευκλείδειου ταξινομητή εκπαιδεύουμε το μοντέλο μας με τα νέα δισδιάστατα δεδομένα εκπαίδευσης. Επιπλέον, κατασκευάζουμε ένα πλέγμα του οποίου η έκταση καθορίζεται από τις μέγιστες και ελάχιστες τιμές των δεδομένων του train set στις δύο διαστάσεις. Κάθε σημείο αυτού του πλέγματος κατηγοριοποιείται από τον ταξινομητή σε μία από τις δέκα διαθέσιμες κατηγορίες. Με βάση τις εκτιμήσεις αυτές και με χρήση της συνάρτησης **contourf** απεικονίζουμε τις περιοχές απόφασης του Ευκλείδειου ταξινομητή, οι οποίες αντιστοιχούν σε διαφορετικές κατηγορίες του συνόλου δεδομένων. Ωστόσο, παρατηρούμε πως στην εικόνα διακρίνονται 9 ξεχωριστές χρωματικές περιοχές αντί για 10 (πλήθος διαφορεκών κλάσεων), γεγονός που οφείλεται σε σφάλμα του ταξινομητή.



Παρατηρούμε ότι η κατανομή των δεδομένων εκπαίδευσης είναι συμβατή με τις περιοχές απόφασης που έχουν σχεδιαστεί παρά την μεγάλη μείωση της διάστασής τους (από 256 σε 2). Ωστόσο, εντοπίζουμε και ορισμένα δείγματα τα οποία βρίσκονται εκτός της περιοχής απόφασής τους. Στα δείγματα αυτά οφείλονται απώλειες (loss) του ταξινομητή.

(γ′) Η καμπύλη εκμάθησης (learning curve) ενός ταξινομητή δείχνει τον τρόπο με τον οποίο επηρεάζεται η ακρίβειά του καθώς αυξάνονται τα δείγματα εκπαίδευσης. Για τον σχεδιασμό της χρησιμοποιούμε την έτοιμη συνάρτηση **plot_learning_curve** που παρουσιάστηκε στο εργαστήριο.



Παρατηρούμε πως για μικρό αριθμό δειγμάτων εκπαίδευσης το training score είναι πολύ μεγαλύτερο από το cross-validation score, γεγονός αναμενόμενο αφού στην περίπτωση αυτή είναι έντονο το φαινόμενο της υπερκπαίδευσης (overfitting). Όσο αυξάνεται ο αριθμός των δειγμάτων εκπαίδευσης (training examples) τόσο μεγαλώνει το cross-validation score, καθώς πλέον δημιουργούνται πιο robust και αντιπροσωπευτικά μοντέλα. Παράλληλα ελαττώνεται λίγο και το training score καθώς μειώνεται το bias. Μετά από έναν δεδομένο αριθμό δειγμάτων (κοντά στα 6.000), οι δύο αυτές μετρικές συγκλίνουν στην ίδια τιμή ακρίβειας (0.85). Επειδή για περισσότερα από 1000 training examples, οι δύο καμπύλες δεν εμφανίζουν μεγάλη απόκλιση αλλά ακολουθούν μια ισορροπημένη συμπεριφορά, μπορούμε να συμπεράνουμε ότι το μοντέλο μας είναι αρκετά ικανοποιητικό και αποδίδει καλά. Συνεπώς, δεν απαιτείται κάποιο περισσότερο πολύπλοκο για το συγκεκριμένο πρόβλημα κατηγοριοποίησης.

Βήμα 14: Υπολογισμός a-priori πιθανοτήτων για κάθε κατηγορία

Στο βήμα αυτό υπολογίζουμε τις **a-priori** πιθανότητες για κάθε κατηγορία (**class priors**). Για το σκοπό αυτό, αρχικά βρίσκουμε το πλήθος των εμφανίσεων κάθε ψηφίου (δηλαδή κάθε κλάσης) στο σύνολο των train δεδομένων. Αυτό επιτυγχάνεται αξιοποιώντας τη συνάρτηση **count_nonzero** της numpy. Στη συνέχεια, διαιρούμε το πλήθος εμφανίσεων κάθε κατηγορίας με το συνολικό αριθμό δειγμάτων εκπαίδευσης, οπότε και λαμβάνουμε τις ζητούμενες πιθανότητες. Η διαδικασία αυτή υλοποιείται μέσω της συνάρτησης **calculate_priors** που ορίζουμε, η οποία δέχεται σαν ορίσματα ένα σύνολο δεδομένων X καθώς και τα αντίστοιχα labels y ενώ επιστρέφει μια λίστα που περιέχει τις a-priori πιθανότητες όλων των κλάσεων.

Καλούμε λοιπόν τη συνάρτηση με ορίσματα X_train και y_train, οπότε λαμβάνουμε τις **a-priori** πιθανότητες των 10 διαφορετικών κλάσεων, που αντιστοιχούν στα 10 διαφορετικά ψηφία (0-9):

κλάση	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
a-priori	0.164	0.138	0.1	0.09	0.089	0.076	0.091	0.088	0.074	0.088

Παρατηρούμε ότι τη μεγαλύτερη a-priori πιθανότητα εμφανίζει το ψηφίο '0', ενώ τη χαμηλότερη το ψηφίο 8. Οι περισσότερες a-priori έχουν παρόμοιες τιμές, κοντά στο 0.1, γεγονός που υποδηλώνει ότι το train dataset είναι αρκετά ισορροπημένο.

Βήματα 15,16: Υλοποίηση Naive Bayesian ταξινομητή

Στο βήμα αυτό υλοποιούμε έναν **Naive Bayes classifier** για την ταξινόμηση όλων των ψηφίων των test δεδομένων ως προς τις 10 κατηγορίες (0-9).

Ο Naive Bayes ταξινομητής είναι ένα πιθανοτικό μοντέλο που βασίζεται κατά βάση στον κανόνα του Bayes (**Bayes Theorem**). Σύμφωνα με αυτόν, αν θεωρήσουμε ένα δείγμα n χαρακτηριστικών $\mathbf{x} = [x_1, x_2, ..., x_n]$ και μια δεδομένη κλάση y τότε ισχύει ότι:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(x_1, \ldots x_n \mid y) \cdot P(y)}{P(x_1, \ldots, x_n)} \tag{1}$$

όπου ο όρος $P(y)$ αντιπροσωπεύει την a-priori πιθανότητα της κλάσης y ενώ ο όρος $P(x_1, \ldots x_n \mid y)$ εκφράζει τη δεσμευμένη πιθανότητα του δείγματος χαρακτηριστικών $\mathbf{x} = [x_1, x_2, ..., x_n]$ δεδομένης της κλάσης y.

Ωστόσο, πέρα από το Μπευζιανό θεώρημα, ο ταξινομητής Naive Bayes κάνει μια επιπλέον σημαντική παραδοχή για τα n χαρακτηριστικά ενός δείγματος x. Συγκεκριμένα, υποθέτει ότι αυτά είναι ανεξάρτητα μεταξύ τους και ότι επηρεάζουν εξίσου το τελικό αποτέλεσμα (όλα τα features συνεισφέρουν με την ίδια βαρύτητα στην τελική απόφαση).

Αξιοποιώντας λοιπόν την υπόθεση ανεξαρτησίας των χαρακτηριστικών, μπορούμε να γράψουμε:

$$P(x_1, \ldots x_n) = P(x_1) \cdot P(x_2) \cdot \ldots \cdot P(x_n) = \prod_{i=1}^{n} P(x_i) \tag{2}$$

$$P(x_1, \ldots x_n \mid y) = P(x_1 \mid y) \cdot P(x_2 \mid y) \cdot \ldots \cdot P(x_n \mid y) = \prod_{i=1}^{n} P(x_i \mid y) \tag{3}$$

Με αντικατάσταση των σχέσεων (2) και (3) στην (1) λαμβάνουμε:

$$P(y \mid x_1, \ldots, x_n) = \frac{\prod_{i=1}^{n} P(x_i \mid y) \cdot P(y)}{\prod_{i=1}^{n} P(x_i)} \tag{4}$$

Εφόσον ο παρανομαστής παραμένει σταθερός για ένα δεδομένο δείγμα εισόδου, μπορούμε να τον αγνοήσουμε, καθώς δεν μας ενδιαφέρει για την ταξινόμηση, δηλαδή:

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y) \tag{5}$$

Τώρα θέλουμε να ορίσουμε το πιθανοτικό μοντέλο ταξινόμησης. Για το σκοπό αυτό υπολογίζουμε την πιθανότητα $P(y \mid x_1, \ldots, x_n)$ για κάθε διαφορετικό y και κρατάμε την κλάση εκείνη που μεγιστοποιεί την πιθανότητα. Μαθηματικά αυτό μπορεί να εκφρασθεί ως εξής:

$$\hat{y} = \arg\max_{y} P(y) \prod_{i=1}^{n} P(x_i \mid y) \tag{6}$$

Επομένως, αυτό που μένει τώρα είναι να υπολογίσουμε την a-priori πιθανότητα $P(y)$ κάθε κλάσης y καθώς και να προσδιορίσουμε τον όρο $P(x_i \mid y)$, που εκφράζει τη δεσμευμένη πιθανότητα ενός χαρακτηριστικού x_i του δείγματος εισόδου x, δεδομένης κλάσης y.

Οι a-priori πιθανότητες $P(y)$ προσδιορίζονται με τον τρόπου που εξηγήθηκε στο Βήμα 14. Για τις υπο-συνθήκη πιθανότητες $P(x_i \mid y)$, θεωρούμε ότι τα χαρακτηριστικά x_i $(i = 1, 2, \ldots, n)$ ενός δείγματος, δεδομένης μιας κλάσης y, ακολουθούν μια μονοδιάστατη Γκαουσιανή κατανομή, δηλαδή θεωρούμε ότι ισχύει:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi {\sigma_y}^2}} \cdot exp\left(-\frac{(x_i - \mu_y)^2}{2{\sigma_y}^2}\right) \tag{7}$$

Θα μπορούσαμε να είχαμε θεωρήσει και κάποια διαφορετική κατανομή, εκτός της κανονικής, ωστόσο χρησιμοποιούμε τη Γκαουσιανή καθώς μοντελοποιεί αρκετά αποτελεσματικά τέτοιου είδους προβλήματα. Γνωρίζομε ότι το γινόμενο n μονοδιάστατων Γκαουσιανών συναρτήσεων οδηγεί σε μια πολυδιάστατη Gaussian, διάστασης n:

$$P(x_1, \ldots x_n \mid y) = \prod_{i=1}^{n} P(x_i \mid y) = \frac{1}{\sqrt{2\pi}^n |\Sigma_y|^{\frac{1}{2}}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \mu_\mathbf{y})^T {\Sigma_y}^{-1} (\mathbf{x} - \mu_\mathbf{y})\right\} \tag{8}$$

όπου $\mathbf{x} = [x_1, x_2, ..., x_n]$ ένα δείγμα 256 χαρακτηριστικών του συνόλου δεδομένων, μ_y το διάνυσμα μέσης τιμής για την κλάση y και Σ_y ο διαγώνιος 256×256 πίνακας συνδιακύμανσης της κλάσης y.

Με βάση λοιπόν τη σχέση (8), βλέπουμε πως κάθε δείγμα $\mathbf{x} = [x_1, x_2, ..., x_n]$ του συνόλου train, δεδομένης μιας κλάσης y, ακολουθεί μια n-διάστατη Γκαουσιανή κατανομή. Αντικαθιστώντας τώρα τη σχέση (8) στο κριτήριο απόφαση (6), το τελευταίο παίρνει την ακόλουθη μορφή:

$$\hat{y} = \arg\max_y \left(P(y) \cdot \frac{1}{\sqrt{2\pi}^n |\Sigma_y|^{\frac{1}{2}}} \exp\left\{ -\frac{1}{2}(\mathbf{x} - \mu_y)^T {\Sigma_y}^{-1} (\mathbf{x} - \mu_y) \right\} \right) \tag{9}$$

Παρατηρούμε ωστόσο, ότι η παραπάνω σχέση δεν είναι ιδαίτερα αποδοτική, λόγω του υψηλού υπολογιστικού κόστους που προκύπτει από την εκθετική συνάρτηση της Γκαουσιανής κατανομής. Για να μειώσουμε την πολυπλοκότητα των πράξεων, μπορούμε να αξιοποιήσουμε τις ιδιότητες της λογαριθμικής συνάρτησης ln:

$$\hat{y} = \arg\max_y \left(P(y) \cdot \frac{1}{\sqrt{2\pi}^n |\Sigma_y|^{\frac{1}{2}}} \exp\left\{ -\frac{1}{2}(\mathbf{x} - \mu_y)^T {\Sigma_y}^{-1} (\mathbf{x} - \mu_y) \right\} \right) \Rightarrow$$

$$\hat{y} = \arg\max_y \left(\ln \left[P(y) \cdot \frac{1}{\sqrt{2\pi}^n |\Sigma_y|^{\frac{1}{2}}} \exp\left\{ -\frac{1}{2}(\mathbf{x} - \mu_y)^T {\Sigma_y}^{-1} (\mathbf{x} - \mu_y) \right\} \right] \right) \Rightarrow$$

$$\hat{y} = \arg\max_y \left(\ln\left[P(y)\right] + \ln \left[\frac{1}{\sqrt{2\pi}^n |\Sigma_y|^{\frac{1}{2}}} \right] + \ln \left[\exp\left\{ -\frac{1}{2}(\mathbf{x} - \mu_y)^T {\Sigma_y}^{-1} (\mathbf{x} - \mu_y) \right\} \right] \right) \Rightarrow$$

$$\boldsymbol{\hat{y} = \arg\max_y \left(\ln\left[P(y)\right] - \frac{n}{2} \ln(2\pi) - \frac{1}{2}|\Sigma_y| - \frac{1}{2}(\mathbf{x} - \mu_y)^T {\Sigma_y}^{-1} (\mathbf{x} - \mu_y) \right)} \tag{10}$$

Η παραπάνω εξίσωση εκφράζει ένα ισοδύναμο κριτήριο απόφασης του ταξινομητή μας και έχει μικρότερη υπολογιστΙκή πολυπλοκότητα. Έτσι, λοιπόν θα αξιοποιηθεί για την κατηγοριοποίηση κάθε test δείγματος σε μία από τις 10 κατηγορίες.

Υλοποίηση σε python

Με τρόπο ανάλογο του βήματος 12, υλοποιούμε τον ταξινομητή σε python σαν ένα **scikit-learn estimator**. Συγκεκριμένα, ορίζουμε την κλάση **CustomNBClassifier**, η οποία κληρονομεί από τις **BaseEstimator** και **ClassifierMixin**. Ο κατασκευαστής της κλάσης (**__init__**) αρχικοποιεί τους πίνακες X_mean_ και X_var_, των μέσων τιμών και διασπορών όλων των ψηφίων (0-9), καθώς επίσης και τον πίνακα των a-priori πιθανοτήτων κάθε κλάσης, ορίζει τον αριθμό των διαθέσιμων κατηγοριών (10) ενώ τέλος ορίζει και μια boolean μεταβλητή που καθορίζει αν ο πίνακας συνδιακύμανσης κάθε κλάσης θα περιέχει μοναδιαίες διασπορές ή όχι. Στη συνέχεια υλοποιούμε τις ακόλουθες μεθόδους:

- **fit**: Δέχεται σαν ορίσματα το train set X μαζί με τα αντίστοιχα labels y και υπολογίζει τους πίνακες X_mean_ και X_var_ των μέσων τιμών και διασπορών όλων των κατηγοριών.

Για την εύρεση των διανυσμάτων μέσης τιμής και διασποράς μιας κατηγορίας, αρχικά εντοπίζουμε τις θέσεις όλων των δειγμάτων των οποίων το label ισούται με την εν λόγω κατηγορία. Στη συνέχεια, διατρέχουμε τα feature vectors των δειγμάτων αυτών και με χρήση των συναρτήσεων **mean** και **var** της numpy υπολογίζουμε τα διανύσματα μέσης τιμής και διασποράς αυτής της κατηγορίας. Έπειτα τα αποθηκεύουμε στους πίνακες X_mean_ και X_var_ αντίστοιχα. Επαναλαμβάνουμε τη διαδικασία για κάθε κατηγορία. Προς αποφυγή του μηδενισμού της ορίζουσας του πίνακα συνδιακύμνασης κάθε κλάσης, προσθέτουμε μια μικρή σταθερά (10^{-5}) στις μηδενικές τιμές του πίνακα. Να σημειωθεί πως σε περίπτωση που η boolean μεταβλητή use_unit_variance δοθεί ως True τότε αντί για τον υπολογισμό της διασποράς κάθε κλάσης με την προαναφερθείσα διαδικασία, θέτουμε απλώς τη διασπορά των χαρακτηριστικών κάθε κλάσης ίση με τη μονάδα.

- **predict**: Δέχεται σαν όρισμα ένα σύνολο ελέγχου X και για κάθε δείγμα του υπολογίζει τις πιθανότητες αυτό να ανήκει σε μία από τις 10 κατηγορίες (0-9). Οι πιθανότητες αυτές καθορίζονται από το κριτήριο απόφασης στο οποίο καταλήξαμε και αποθηκεύονται σε έναν δισδιάστατο πίνακα criterion, μεγέθους 10 × test samples. Με χρήση της συνάρτησης **argmax** της numpy κρατάμε τη μέγιστη τιμή που αντιστοιχεί σε κάθε στήλη του πίνακα (πιθανότερη κατηγορία κάθε sample), όπως υποδηλώνει η (10), οπότε τελικά η συνάρτηση επιστρέφει ένα μονοδιάστατο πίνακα predictions με τις προβλέψεις για κάθε test δείγμα.
- **score**: Δέχεται σαν όρισμα το test set X μαζί με τις αντίστοιχες επισημειώσεις y. Καλεί την μέθοδο predict και αναθέτει σε κάθε δείγμα ένα label $\hat{y}$. Για τον υπολογισμό του accuracy score διαιρεί τον αριθμό των σωστών προβλέψεων (εκείνες για τις οποίες το $\hat{y}$ συμπίπτει με το y) με τον συνολικό αριθμό των δειγμάτων του συνόλου δοκιμής.

Score Custom Naive Bayes ταξινομητή

Για τον υπολογισμό του σκορ του Naive Bayes ταξινομητή που κατασκευάσαμε, ορίζουμε τη συνάρτηση **evaluate_custom_nb_classifier** η οποία δέχεται ως ορίσματα ένα σύνολο δεδομένων X, τα αντίστοιχα labels y καθώς και μια παράμετρο folds (με default τιμή 5) που καθορίζει τον αριθμό των διαμερίσεων για το cross validation. Η συνάρτηση ορίζει ένα μοντέλο **CustomNBClassifier** και έπειτα υπολογίζει το score του αξιοποιώντας τη συνάρτηση **evaluate_classifier** που ορίσαμε στο βήμα 13 (α).

Καλούμε λοιπόν τη συνάρτηση με ορίσματα: το X_dataset, το y_dataset (τα σύνολα αυτά έχουν προκύψει με concatenation των αρχικών συνόλων εκπαίδευσης και ελέγχου) και τον αριθμό των folds, οπότε λαμβάνουμε score ίσο με **76.525%**.

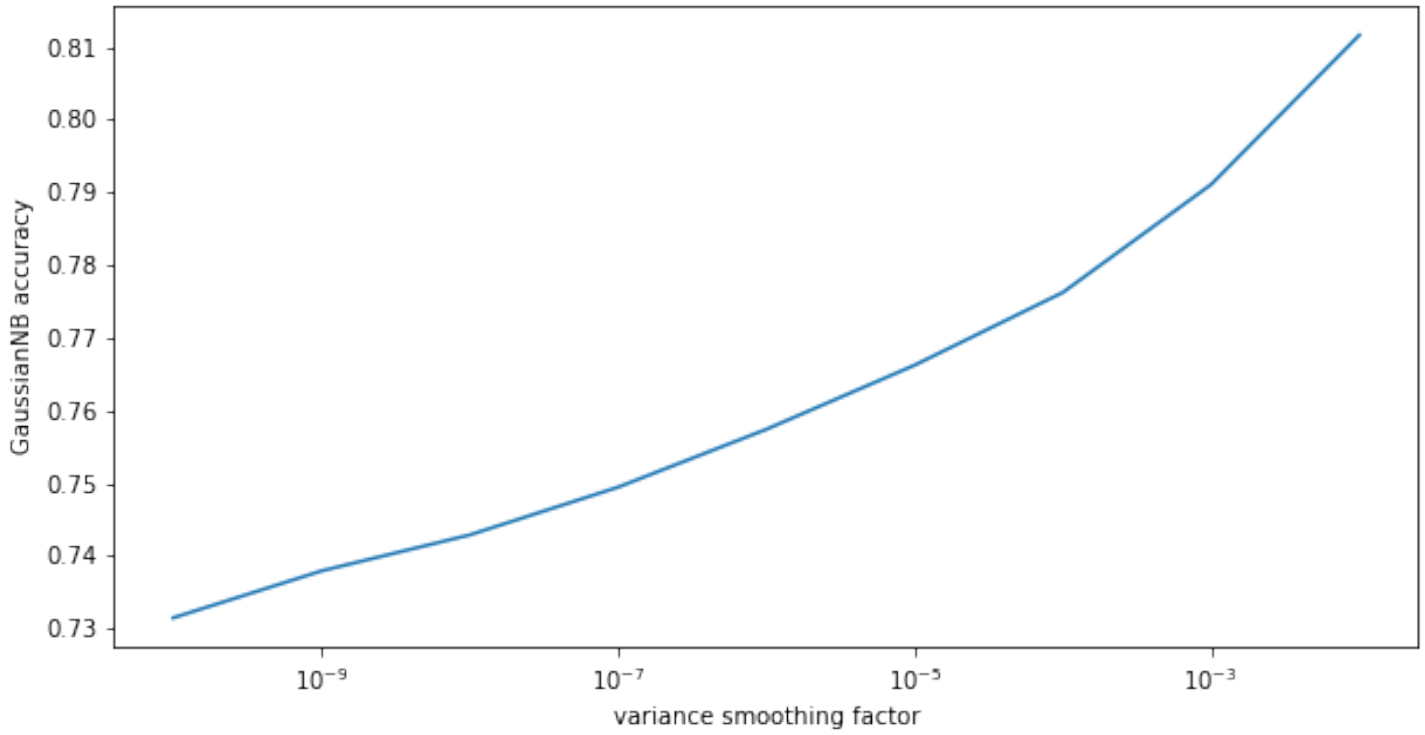
Score scikit-learn Naive Bayes ταξινομητή

Θέλουμε τώρα να συγκρίνουμε τον Naive Bayes ταξινομητή που κατασκευάσαμε με την έτοιμη

υλοποίηση του **scikit-learn**. Για τον υπολογισμό του σκορ του ταξινομητή του scikit-learn ορίζουμε τη συνάρτηση **evaluate_sklearn_nb_classifier**, η οποία δέχεται ακριβώς τα ίδια ορίσματα με τη συνάρτηση **evaluate_custom_nb_classifier** που ορίσαμε προηγουμένως. Η συνάρτηση ορίζει ένα μοντέλο **GaussianNB** και έπειτα υπολογίζει το score του, αξιοποιώντας τη συνάρτηση **evaluate_classifier** που ορίσαμε στο βήμα 13 (α).

Καλούμε λοιπόν τη συνάρτηση με ορίσματα: το X_dataset, το y_dataset και τον αριθμό των folds (5), οπότε λαμβάνουμε score ίσο με **73.792%**.

Παρατηρούμε ότι η δική μας υλοποίηση υπερτερεί σε ακρίβεια ως προς την έτοιμη του scikit learn. Μετά από μελέτη του documentation, διαπιστώσαμε ότι η μόνη διαφορά μεταξύ του **CustomNBClassifier** και της υλοποίησης του **GaussianNB** είναι το γεγονός ότι η τελευταία δέχεται μια παράμετρο **var_smoothing** για διαχείριση αριθμητικής ευστάθειας. Συγκεκριμένα, η τιμή αυτής της παραμέτρου πολλαπλασιάζεται με τη μέγιστη τιμή του πίνακα διασπορών και η σταθερά που προκύπτει προστίθεται σε κάθε στοιχείο του. Για λόγους σύγκρισης, τροποποιούμε την κλάση **CustomNBClassifier** και δημιουργούμε μια νέα, την **SmoothCustomNBClassifier**, η οποία εφαρμόζει την προαναφερθείσα διαδικασία και λειτουργεί αντίστοιχα με την υλοποίηση του scikit learn. Επίσης, για να εξετάσουμε την επίδραση της παραμέτρου **var_smoothing** στην επίδοση του ταξινομητή **GaussianNB**, σχεδιάζουμε το ακόλουθο διάγραμμα:



Παρατηρούμε πως όσο μεγαλώνει η τιμή της παραμέτρου **var_smoothing** τόσο αυξάνεται το score του Naive Bayes ταξινομητή.

Μοναδιαία διασπορά στα χαρακτηριστικά (unit variance)

Ορίζουμε ξανά ένα μοντέλο **CustomNBClassifier**, μόνο που τώρα θέτουμε την τιμή **unit_variance** σε **True** (αντί για False που ήταν στο βήμα 15). Έτσι υποθέτουμε ότι η διασπορά για όλα τα χαρακτηριστικά, για όλες τις κατηγορίες ισούται με 1. Καλούμε λοιπόν τη συνάρτηση **evaluate_classifier** με ορίσματα: το μοντέλο, τα σύνολα X_dataset, y_dataset και τον αριθμό των folds (5). Έτσι, λαμβάνουμε το score του ταξινομητή, το οποίο στη συγκεκριμένη περίπτωση είναι ίσο με **84.024%**. Παρατηρούμε πως είναι μεγαλύτερο από το αντίστοιχο score του Naive Bayes ταξινομητή που υπολόγιζε τις πραγματικές τιμές των διασπορών (78.193%). Αυτό είναι αναμενόμενο καθώς, θέτονας όλες τις διασπορές στην ίδια τιμή, εξαλείφουμε εντελώς την εξάρτηση που υπάρχει μεταξύ τους, οπότε ο ταξινομητής βασίζεται πλέον μόνο στις μέσες τιμές των χαρακτηριστικών. Αυτό μάλιστα δικαιολογεί και το γεγονός ότι το ποσοστό ακρίβειας είναι σχεδόν ταυτόσημο με αυτό του Ευκλείδειου ταξινομητή που εξετάσαμε σε προηγούμενα βήματα. Μάλιστα αν αγνοήσουμε τις a-priori πιθανότητες τότε ο ταξινομητής μας ταυτίζεται απόλυτα με τον Ευκλείδειο ταξινομητή.

Βήμα 17: Σύγκριση ταξινομητών

Συγκρίνουμε τώρα την επίδοση των ταξινομητών Naive Bayes, Nearest Neighbors και SVM (με διαφορετικούς kernels). Για το σκοπό αυτό δημιουργούμε συναρτήσεις οι οποίες ορίζουν το αντίστοιχο μοντέλο, χρησιμοποιώντας τις έτοιμες υλοποιήσεις του scikit-learn, και υπολογίζουν το 5-fold-cross-validation score καλώντας την evaluate_classifier του βήματος 13 (α).

K Nearest Neighbors

Αρχικά υλοποιούμε τον ταξινομητή k κοντινότερων γειτόνων. Σύμφωνα με αυτόν, ένα νέο δείγμα ταξινομείται στην κλάση εκείνη που ανήκει η πλειοψηφία των k πλησιέστερων γειτόνων του. Εάν $k = 1$, τότε το δείγμα απλώς αντιστοιχίζεται στην κλάση αυτού του μεμονωμένου γείτονα που βρίσκεται πλησιέστερα σε αυτό σύμφωνα με κάποια απόσταση (πχ. Ευκλείδεια). Μοντέλα όπως αυτό του k-NN είναι γνωστά σαν μη γενικευμένα καθώς απλά θυμούνται όλα τα δεδομένα εκπαίδευσης. Ωστόσο, παρά την απλότητά τους είναι αρκετά επιτυχημένα σε πολλές εφαρμογές ταξινόμησης όπως αυτή της αναγνώρισης χειρόγραφων ψηφίων.

Πράγματι, το ποσοστό που λαμβάνουμε είναι ίσο με **96.13 %** το οποίο είναι αρκετά υψηλό. Σημειώνουμε πως η τιμή $k = 3$ του αριθμού των κοντινότερων γειτόνων προέκυψε ύστερα από πειραματισμό και είναι αυτή για την οποία λάβαμε το μεγαλύτερο score.

Linear SVM και Rbf SVM

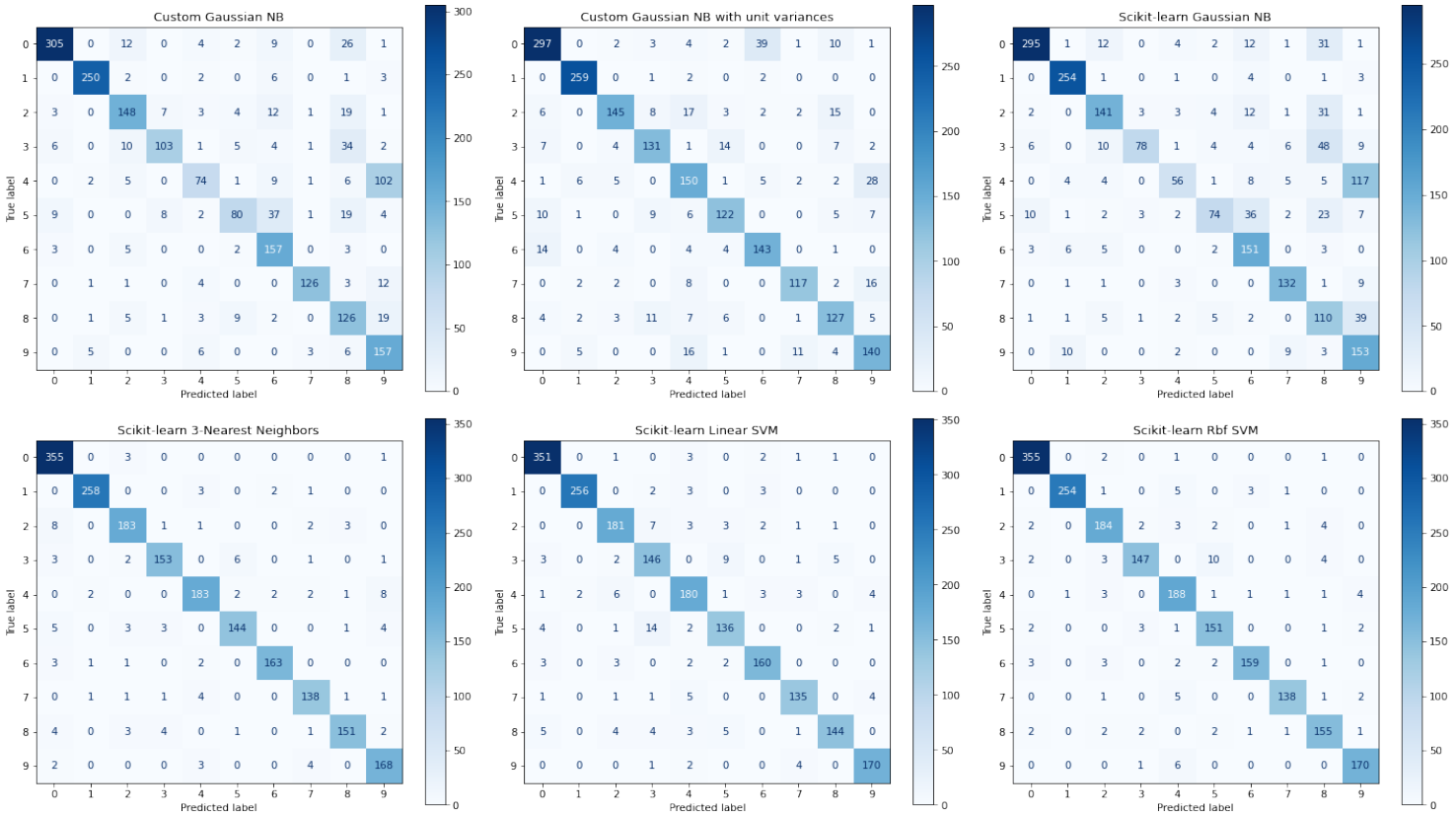
Στη συνέχεια, δοκιμάζουμε τον ταξινομητή SVM με διαφορετικούς πυρήνες. Οι μηχανές

διανυσμάτων υποστήριξης όπως λέγονται, βρίσκουν το βέλτιστο υπερεπίπεδο με το μεγαλύτερο περιθώριο ταξινόμησης. Στην περίπτωση των δισδιάστατων και τρισδιάστατων δεδομένων αυτό είναι μια ευθεία ή ένα επίπεδο αντίστοιχα. Η εύρεσή του είναι εύκολη στην περίπτωση των γραμμικώς διαχωρίσιμων δεδομένων. Αντιθέτως, όταν τα δεδομένα δεν είναι γραμμικώς διαχωρίσιμα χρησιμοποιούνται κάποιες συναρτήσεις, που ονομάζονται πυρήνες (kernels), οι οποίες μετατρέπουν το μη διαχωρίσιμο πρόβλημα σε διαχωρίσιμο. Μεταξύ των πυρήνων που χρησιμοποιούνται συχνά είναι οι 'linear', 'rbf' και 'poly'. Μερικά από τα πλεονεκτήματα των μηχανών αυτών είναι ότι είναι αποτελεσματικά σε χώρους μεγάλων διαστάσεων και ότι δεν χρειάζεται να θυμούνται όλα τα δεδομένα εκπαίδευσης παρά μόνο τα διανύσματα υποστήριξης (support vectors).

Τα ποσοστά που λαμβάνουμε για τον SVM ταξινομητή με 'linear' και 'rbf' πυρήνες αντίστοιχα είναι **94.66 %** και **97.19 %**, γεγονός που τους καθιστά ιδιαίτερα επιτυχημένους στο πρόβλημα ταξινόμησης που εξετάζουμε.

Confusion Matrix

Για την καλύτερη εποπτεία της επίδοσης των επιμέρους ταξινομητών αλλά και την μεταξύ τους σύγκριση χρησιμοποιούμε τον πίνακα σύγχυσης (**Confusion Matrix**). Το στοιχείο (i,j) του πίνακα αυτού υποδηλώνει τον αριθμό των δειγμάτων που ταξινομήθηκαν στην κλάση j, ενώ στην πραγματικότητα ανήκουν στην κλάση i.



Παρατηρούμε ότι ο SVM ταξινομητής με πυρήνα 'rbf' προβλέπει σωστά τα περισσότερα ψηφία των δεδομένων δοκιμής. Το γεγονός αυτό επιβεβαιώνει και το μεγαλύτερο score του συγκεκριμένου ταξινομητή σε σύγκριση με τους υπόλοιπους. Εξίσου καλά λειτουργεί και ο 3-NN, ο οποίος έχει σχεδόν την ίδια απόδοση με τον Rbf SVM. Αναφορικά με τους Gaussian Naive Bayes ταξινομητές παρατηρούμε πως αυτός της scikit-learn μπερδεύει το ψηφίο 4 με το 9 αφού 117 δείγματα που κανονικά ανήκουν στην κλάση 4 ταξινομήθηκαν εσφαλμένα από αυτόν στην κλάση 9. Επιπλέον, συγκρίνοντας την δική μας υλοποίηση του συγκεκριμένου ταξινομητή με την έτοιμη του Scikit-learn συμπεραίνουμε πως ο δικός μας αναγνωρίζει πιο αποτελεσματικά τα ψηφία 3,4 και 5.

Βήμα 18: Ensembling

Η βασική ιδέα του βήματος αυτού είναι ο συνδυασμός κάποιων ταξινομητών με αρκετά καλή επίδοση με στόχο να επιτευχθεί επίδοση υψηλότερη των επιμέρους. Αυτή η τεχνική είναι γνωστή ως **ensembling**, ενώ είναι σημαντικό οι ταξινομητές που θα συνδυαστούν να χαρακτηρίζονται από διαφορετικό τύπο λαθών. Για το πρόβλημά μας οι ταξινομητές με το μεγαλύτερο score, σύμφωνα με το Βήμα 17, είναι οι Rbf SVM, 3-NN και Euclidean. Επιπλέον, με βάση το Confusion Matrix επιβεβαιώνουμε ότι δεν ταξινομούν λάθος τα ίδια ψηφία. Συγκεκριμένα, μόνο ο δικός μας Ευκλείδειος ταξινομητής τείνει να ταξινομεί λανθασμένα το ψηφίο 4 στην κατηγορία 9, ενώ οι άλλοι δύο προβλέπουν σωστά την πλειοψηφία των δειγμάτων.

(α′) Αρχικά πειραματιζόμαστε με τον **VotingClassifier** του scikit-learn ο οποίος συνδυάζει τους επιμέρους ταξινομητές βάζοντάς τους να ψηφίσουν για το αποτέλεσμα. Ο συνδυασμός τους μπορεί να γίνει σε **hard** ή **soft voting**. Στην περίπτωση του πρώτου κάθε μεμονωμένος ταξινομητής ψηφίζει για μια κλάση και η πλειοψηφία κερδίζει. Στο **soft voting** ο κάθε ταξινομητής παρέχει μια τιμή πιθανότητας για ένα συγκεκριμένο δείγμα να ανήκει σε μία κατηγορία. Οι προβλέψεις σταθμίζονται κατάλληλα και προστίθενται. Στη συνέχεια, η κλάση με το μεγαλύτερο άθροισμα σταθμισμένων πιθανοτήτων κερδίζει την ψηφοφορία.

Για την υλοποίηση των παραπάνω ορίζουμε την συνάρτηση **evaluate_voting_classifier** η οποία δέχεται για ορίσματα ένα dataset X μαζί με τα αντίστοιχα labels y και τον αριθμό των folds. Αφού ορίσει τους επιμέρους ταξινομητές καλεί την **VotingClassifier** για τον συνδυασμό τους. Ο υπολογισμός του 5-fold cross-validation-score γίνεται με χρήση της συνάρτησης evaluate_classifier του βήματος 13 (α).

Συνδυάζοντας τους τρείς ταξινομητές με το μεγαλύτερο score σε **hard** και **soft voting**, τα ποσοστά που λαμβάνουμε είναι αντίστοιχα **96.27 %** και **96.11 %** τα οποία δεν είναι υψηλότερα από το **97.19 %** του Rbf SVM. Το γεγονός αυτό δεν μας παραξενεύει αφού ο

VotingClassifier δεν είναι εγγυημένο ότι παρέχει καλύτερη απόδοση από οποιοδήποτε μοντέλο του συνόλου. Ωστόσο, ένας τέτοιος μετα-ταξινομητής μπορεί να προσφέρει χαμηλότερη διακύμανση (variance) στην ακρίβεια η οποία μπορεί να είναι επιθυμητή δεδομένης της υψηλότερης σταθερότητας ή εμπιστοσύνης του μοντέλου. Να σημειώσουμε ότι ο αριθμός των ταξινομητών που επιλέγουμε πρέπει να είναι μονός προκειμένου να αποφύγουμε τυχόν 'ισοπαλίες' μεταξύ τους.

(β′) Εξετάζουμε τώρα την περίπτωση του **BaggingClassifier** ως μία εναλλακτική μέθοδο για την δημιουργία ενός ensemble. Η τεχνική bagging, αφορά στο χωρισμό του training set σε υποσύνολα τα οποία προκύπτουν επιλέγοντας τυχαία δεδομένα από το αρχικό σύνολο εκπαίδευσης με αντικατάσταση. Αυτό σημαίνει ότι πολλά δείγματα μπορεί να επαναλαμβάνονται στο ίδιο ή σε διαφορετικά υποσύνολα ή ακόμα και να μην χρησιμοποιούνται καθόλου. Στη συνέχεια, επιλέγεται ένας ταξινομητής ο οποίος εκπαιδεύεται σε καθένα από αυτά τα υποσύνολα. Η τελική απόφαση βγαίνει μέσω ψηφοφορίας ή μέσου όρου των επιμέρους προβλέψεων.

Για την υλοποίηση του **BaggingClassifier** χρησιμοποιούμε την ομώνυμη συνάρτηση του scikit-learn ενώ ο ταξινομητής που επιλέγουμε από τα προηγούμενα βήματα είναι ο **Rbf SVM** που είχε την μεγαλύτερη απόδοση.

Το ποσοστό που λαμβάνουμε με την τεχνική **Bagging** είναι **96.87 %**, το οποίο είναι μικρότερο από εκείνο του Rbf SVM. Επομένως, παρατηρούμε ότι και σε αυτή την περίπτωση δεν παίρνουμε κάποιο καλύτερο αποτέλεσμα. Ωστόσο, τέτοιες μέθοδοι χρησιμοποιούνται συχνά καθώς βοηθάνε στην μείωση του variance και την αποφυγή του overfitting.

Βήμα 19: Neural Network

(α′) Υλοποιούμε έναν **dataloader** για να αναλάβει την ανάγνωση των δεδομένων και τον χωρισμό σε batches. Πρώτο βήμα αποτελεί η δημιουργία της κλάσης **DigitDataset** που κληρονομεί από την **torch.util.data.Dataset**. Συγκεκριμένα, ορίζουμε τον κατασκευαστή της κλάσης (__**init**__), ο οποίος μετατρέπει τα δεδομένα εισόδου της από numpy.ndarray σε tensors, καθώς επίσης και τις μεθόδους **len** και **getitem**. Από αυτές η πρώτη επιστρέφει το μέγεθος του dataset, ενώ η δεύτερη δέχεται ένα index το οποίο χρησιμοποιείται για να πάρουμε το δείγμα στην αντίστοιχη θέση (η αρίθμηση ξεκινάει από το μηδέν). Αυτό θα είναι tuple της μορφής (sample, label) όπου κάθε ένα από τα sample, label είναι ένας tensor. Επιπλέον, ορίζουμε (εκτός κλάσης) την συνάρτηση dataloader η οποία αναλαμβάνει τον χωρισμό ενός dataset σε batches μεγέθους batch_size (default = 32) το καθένα. Στην περίπτωση που για την συγκεκριμένη τιμή του batch_size τα δεδομένα δεν γίνεται να χωριστούν ακριβώς, τότε όσα περισσεύουν προστίθενται μαζί σε ένα batch το οποίο φυσικά θα έχει μικρότερο μέγεθος από τα υπόλοιπα. Η συνάρτηση αυτή επιστρέφει

μία λίστα όπου το κάθε στοιχείο της αφορά ένα batch και είναι ένα tuple με τα 256 χαρακτηριστικά όλων των δειγμάτων του και τα αντίστοιχα labels σε μορφή tensors.

(β′) Στη συνέχεια υλοποιούμε ένα **fully connected** νευρωνικό δίκτυο ως μία κλάση η οποία κληρονομεί από την **nn.Module**. Στον κατασκευαστή δίνουμε τις διαστάσεις D_in (εισόδου), D_out (εξόδου) και H (κρυφή διάσταση), ενώ ορίζουμε τα επίπεδα του δικτύου (input, hidden και output layers). Η είσοδος ενός επιπέδου είναι πάντα η έξοδος του αμέσως προηγούμενού του. Η αρχιτεκτονική του δικτύου φαίνεται στην μέθοδο forward όπου και επιλέγουμε την relu ως συνάρτηση ενεργοποίησης.

(γ′) Για την εκπαίδευση και το evaluation του νευρωνικού δικτύου χωρίζουμε αρχικά το dataset σε train και validation για το training. Αυτό γίνεται μέσω της συνάρτησης **random_split**. Αφού ορίσουμε το μοντέλο μας, επιλέγουμε για optimizer τον **Stochastic Gradient Descent (SGD)** και για loss function την **CrossEntropy**. Κατόπιν, εκπαιδεύουμε το δίκτυό μας για 200 εποχές. Σε κάθε εποχή, διατρέχουμε ένα ένα τα batches των δεδομένων εκπαίδευσης, υπολογίζουμε το loss και ανανεώνουμε τα βάρη. Αφού τα διατρέξουμε όλα, αξιολογούμε το μοντέλο μας με βάσει τα validation data. Δηλαδή, σε κάθε εποχή υπολογίζουμε τις μετρικές training loss, validation loss και validation accuracy. Μόλις ολοκληρωθούν και οι 200 και τα βάρη λάβουν τις τελικές τους τιμές, αξιολογούμε την επίδοση του νευρωνικού στα δεδομένα test.

(δ′) Αξιολογώντας το νευρωνικό δίκτυο που ορίσαμε πάνω στα test δεδομένα, λαμβάνουμε ποσοστό ακρίβειας ίσο με **91.88%**, τιμή αρκετά υψηλή αν αναλογιστούμε πως το δίκτυο μας αποτελείται μόνο από Fully Connected επίπεδα. Πειραματιζόμαστε με διαφορετικό loss function, optimizer, αριθμό εποχών αλλά και αρθμό hidden layers, ωστόσο η ακρίβεια του μοντέλου μας δεν βελτιώνεται ιδιαίτερα. Αξίζει ωστόσο να αναφερθεί ότι σε περίπτωση που ορίζαμε ένα Συνελικτικό Νευρωνικό Δίκτυο (CNN), τότε το accuracy θα έφθανε σε ακόμα μεγαλύτερα επίπεδα.