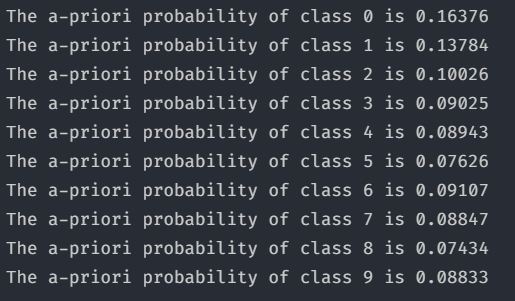
Βήμα 14

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



Βήμα 15

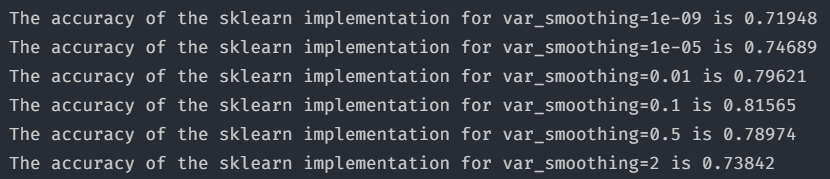
Για την υλοποίηση του Naive Bayes προστέθηκαν υλοποιήσεις των μεθόδων fit και predict. Το training έγινε χρησιμοποιώντας τις προηγούμενες συναρτήσεις υπολόγιζε τις a-priori πιθανότητες και τις μέσες τιμές και τυπικές αποκλίσεις των χαρακτηριστικών για κάθε κλάση. Συγκεκριμένα για την τυπική απόκλιση,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

Όπου οι είναι κανονικές κατανομές με μέση τιμή και τυπική απόκλιση ίση με αυτή που υπολογίσαμε στο training οπότε ο υπολογισμός τους γίνεται από την ως εξής

Παίρνοντας τα predictions πάνω στο test dataset υπολογίζουμε την εξής ακρίβεια



Παρακάτω φαίνονται για σύγκριση τα αποτελέσματα της έτοιμης υλοποίησης για διαφορετικά var\_smoothing



Το accuracy που έβγαλε η δικιά μας υλοποίηση ήταν πρακτικά ίδιο με τον default GaussianNB (). 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 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

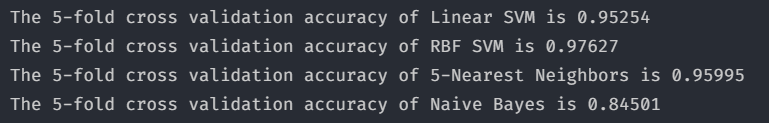


Το οποίο πρακτικά ταυτίζεται με το καλύτερο αποτέλεσμα που βρήκαμε με την έτοιμη υλοποίηση

Βήμα 17

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

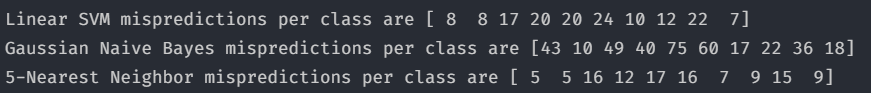
* Linear SVM
* Radial Basis Function SVM
* 5-Nearest Neighbors
* Naive Bayes



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



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



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



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

