Αναγνώριση Προτύπων
1η Εργαστηριακή Άσκηση
Οπτική Αναγνώριση Ψηφίων

Παρέλλη Μαρία
03115155
9ο Εξάμηνο

Χαρδούβελης Γεώργιος-Ορέστης
03115100
9ο Εξάμηνο

1 Εισαγωγή

Θέμα της άσκησης είναι η υλοποίηση ενός συστήματος οπτικής αναγνώρισης ψηφίων. Στα αρχεία που μας δίνονται, σε κάθε γραμμή περιέχονται χαρακτηριστικά για ένα ψηφίο. Στην πρώτη στήλη έχουμε για ποιό στοιχείο πρόκειται στην πραγματικότητα, ενώ κάθε άλλη στήλη αντιστοιχεί σε ένα χαρακτηριστικό. Συγκεκριμένα, κάθε στήλη περιγράφει σε ένα πιξελ του σκαναρισμένου στοιχείου και αντιστοιχεί στην απόχρωση του γκρι για αυτό.

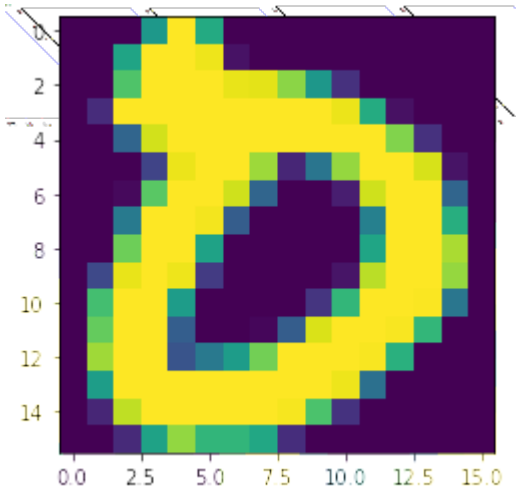
Η εργασία πραγματοποιήθηκε στο Google Colab και γενικά χρησιμοποιήθηκαν vectorized υλοποιήσεις στις μεθόδους μας.

Βήμα 1

Σε αυτό το βήμα έγινε η ανάγνωση και η επεξεργασία των αρχείων test.txt και train.txt. Διαβάστηκαν με την βοήθεια της βιβλιοθήκης πανδα και μετατρέποντας τα σε numpy πίνακες δημιουργήσαμε του πίνακες X_test, y_test, X_train, y_train, διαχωρίζοντας την πρώτη στήλη (του test και του train αντίστοιχα) για τους y πίνακες μιας και περιέχουν τα labels για τα ψηφία, ενώ τις υπόλοιπες στήλες για τους X πίνακες μιας και περιέχουν τα χαρακτηριστικά των ψηφίων.

Βήμα 2

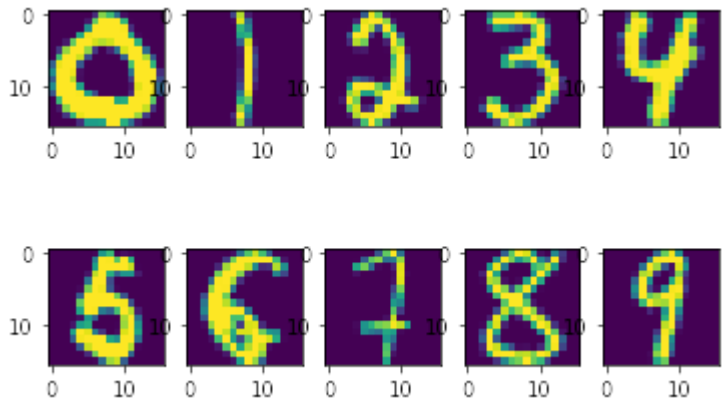
Επιλέγουμε την 131_{η} γραμμή από τον πίνακα X_train όπου περιέχει τα 256 χαρακτηριστικά του αντίστοιχου ψηφίου. Με τη συνάρτηση numpy.reshape οργανώνουμε τα χαρακτηριστικά σε έναν πίνακα $16x16$ και εύκολα μετά απεικονίζουμε το αποτέλεσμα με την συνάρτηση matplotlib.pyplot.imshow. Το αποτέλεσμα έχει ως εξής:



Βλέπουμε πως εμφανίζεται ένα μηδενικό, όπου επιβεβαιώνεται εφόσον το αντίστοιχο label του $131_{ου}$ ψηφίου είναι 0.

Βήμα 3

Εδώ, χρησιμοποιώντας το panda dataframe και την συνάρτηση groupby(), δημιουργούμε ένα dataframe με 10 τυχαία ψηφία με διαφορετικό label. Έτσι έχουμε dataframe με 10 σειρές, όπου κάθε μία έχει χαρακτηριστικά για ένα διαφορετικό ψηφίο από το 0 ως το 9. Όμοια με το βήμα 2 απεικονίζουμε τα ψηφία και προκύπτει το παρακάτω.



Βήμα 4

Ομαδοποιήθηκαν όλες οι γραμμές οι οποίες αντιστοιχούσαν σε ψηφίο με label 0. Ύστερα απομονώθηκε η στήλη που αντιστοιχεί στο pixel(10,10) και υπολογίστηκε ο μέσος όρος των στοιχείων της, ο οποίος βλέπουμε πως είναι ίσος με 0.43617587939698493.
Βλέποντας την θέση του pixel(10,10) σε παραπάνω απεικόνιση του ψηφίου 0, το αποτέλεσμα είναι εύλογο αφού πέφτει σε περιοχή κοντά στην κλειστή καμπύλη όπου σχηματίζει το ψηφίο, οπότε σε πολλές είναι μέρος της και σε άλλες δεν είναι. Συνεπώς, η μέση τιμή δεν είναι ιδιαίτερα μεγάλη.

Βήμα 5

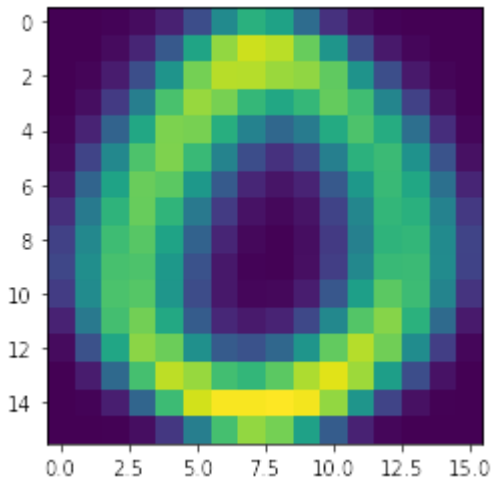
Από τα ίδια στοιχεία που υπολογίσαμε πάνω υπολογίζουμε και την διασπορά του ίδιου pixel, όπου προκύπτει 0.5344999053355629. Αυτό είναι επίσης εύλογο αποτέλεσμα αφού εφόσον είναι κοντά στην κλειστή καμπύλη του ψηφίου παίρνει συχνά αντιδιαμετρικές τιμές.

Βήμα 6

Έχοντας έτοιμο ήδη από τα παραπάνω πίνακα με τα χαρακτηριστικά του ψηφίου 0, γίνεται ο ίδιος υπολογισμός για όλες τις γραμμές και προκύπτουν μονοδιάστατοι πίνακες με τις μέσες τιμές και διακυμάνσεις κάθε pixel του ψηφίου μηδέν.

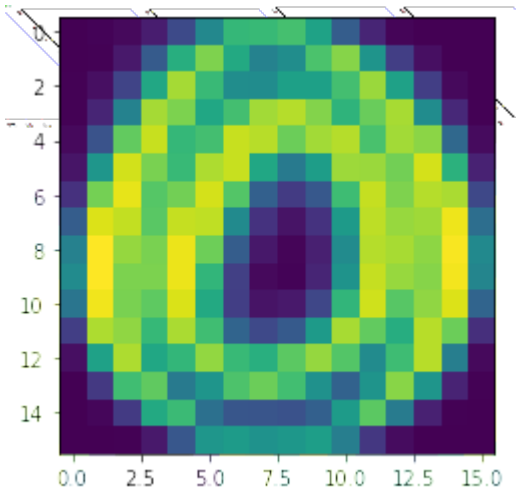
Βήμα 7

Έχοντας τις μέσες τιμές από το προηγούμενο βήμα, με όμοια τεχνική με το βήμα 2 απεικονίζουμε το ψηφίο 0. Το αποτέλεσμα έχει ως εξής:



Βήμα 8

Επαναλαμβάνουμε με τις διασπορές και προκύπτει το εξής:



Συγκρίνοντας, βλέπουμε πως και στις δύο περιπτώσεις, τα σημεία στις 4 άκρες καθώς και στο κέντρο είναι σταθερά (λογικό καθώς για την πλειονότητα των σκαναρισμένων εικόνων είναι 0).
Αντίθετα, στην κλειστή καμπύλη που σχηματίζει το μηδέν βλέπουμε διαφορές. Συγκεκριμένα, οι περιοχές που εμφανίζονται με ανοιχτό μπλε στο βήμα 8, έχουν κάποια ενδιάμεση τιμή και άρα σε ορισμένες εικόνες ήταν μαύρο και σε άλλες όχι. Εφόσον παίρνουμε την μέση τιμή έχουμε κάποιο ενδιάμεσο χρώμα. Στα αντίστοιχα σημεία, χρησιμοποιώντας τις διακυμάνσεις, είναι φανερό πως θα εμφανίζονται μεγάλες τιμές διακύμανσης και έτσι θα έχουμε έντονο κίτρινο χρώμα (που θα αντιστοιχούσε στο μαύρο χρώμα).
Απο την άλλη, στα σημεία που έχουμε έντονο κίτρινο χρώμα στο αποτέλεσμα με μέσες τιμές (άρα πολλές τιμές κοντά στο 1) έχουμε μικρή διακύμανση οπότε στο αποτέλεσμα μέσω διακυμάνσεων έχουμε ενδιάμεσο χρώμα.
Σε κάθε περίπτωση βεβαίως το αποτέλεσμα είναι φανερά το ψηφίο 0. Στην περίπτωση των μέσων τιμών είναι περισσότερο εμφανής η λεπτομέρεια.

Τέλος, αυτό που μπορούμε να συμπεράνουμε για τον τρόπο που ο κόσμος σχεδιάζει το ψηφίο 0 στηριζόμενοι στο dataset μας είναι πως το πάνω και κάτω μέρος που εμφανίζεται πιο συμπαγές με την μέση τιμή ενώ εμφανίζει μικρή διαφορά με την διασπορά σχεδιάζεται περίπου με τον ίδοι τρόπο σε μεγάλο ποσοστό ατόμων. Εντούτοις, στα πλάγια μέρη είναι που εμφανίζεται η μεγαλύτερη διαφορά στον τρόπο σχεδιασμού του ψηφίου και εκεί που γίνεται εμφανής ο διαφορετικός γραφικός χαρακτήρας ανάμεσα σε μια πληθώρα ατόμων.

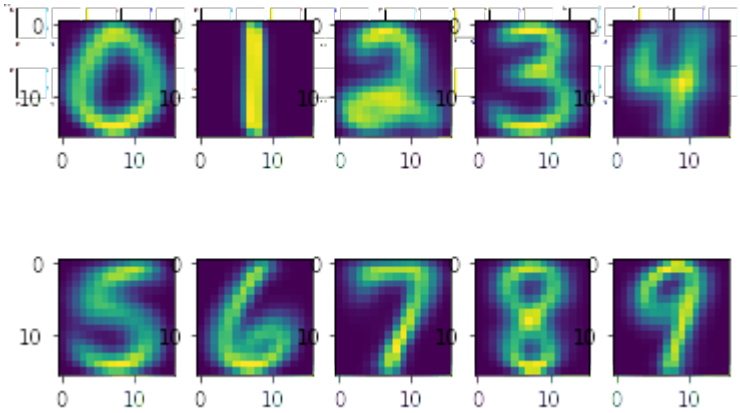
Βήμα 9

α

Επαναλαμβάνοντας το βήμα 6 για όλα τα διαφορετικά ψηφία, υπολογίζουμε εύκολα τις μέσες τιμές και διασπορές τους για κάθε pixel.

β

Χρησιμοποιώντας τα αποτελέσματα που προκύπτουν απο τις μέσες τιμές σχεδιάζουμε όλα τα ψηφία και το αποτέλεσμα φαίνεται παρακάτω.



Παρατηρούμε πως σχηματίζονται ευδιάκριτα και τα 10 ψηφία. Λίγο πιο θολό από τα υπόλοιπα είναι το 2, λόγω της καμπύλης στο κάτω μέρος.

Βήμα 10

Απομονώνουμε τα χαρακτηριστικά του 101ου στοιχείου από τον X_test και στη συνέχεια, υπολογίζουμε με την βοήθεια της συνάρτησης np.linalg.norm την ευκλείδεια απόσταση από την μέση τιμή κάθε πιθανού ψηφίου. Τυπώνοντας το ψηφίο από το οποίο παρατηρείται η μικρότερη απόσταση, βλέπουμε πως το αντιστοιχίζει στο ψηφίο 0.
Παράλληλα τυπώνουμε και το label του $101_{ου}$ στοιχείου και βλέπουμε πως ήταν όντως το ψηφίο 0. Έτσι η ταξινόμηση ήταν επιτυχής.

Βήμα 11

α

Επαναλαμβάνουμε την διαδικασία του βήματος 10 για κάθε γραμμή του στοιχείου X_test και υπολογίζουμε τον πίνακα y_pred που έχει σε έναν μονοδιάστατο πίνακα

τα αποτελέσματα της ταξινόμησης μας.

β

Συγκρίνοντας τους πίνακες y_pred (τις ταξινομήσεις μας) και y_test (με τα πραγματικά labels) με τη χρήση της συνάρτησης sklearn.metrics.accuracy_score, υπολογίζουμε το accuracy. Έτσι, τυπώνοντας το βλέπουμε πως το ποσοστό επιτυχίας είναι 0.8141504733432985. Αν λάβουμε υπόψη την πολυπλοκότητα του ταξινομητή μας είναι ένα αποδεκτό αποτέλεσμα.

Βήμα 12

Με βάση το outline που δόθηκε, υλοποιήθηκε ο EuclideanClassifier(), υλοποιώντας τις μεθόδους fit, predict και score.
Οι υλοποιήσεις ακολουθούν όμοια λογική με την υλοποίηση των παραπάνω βημάτων.

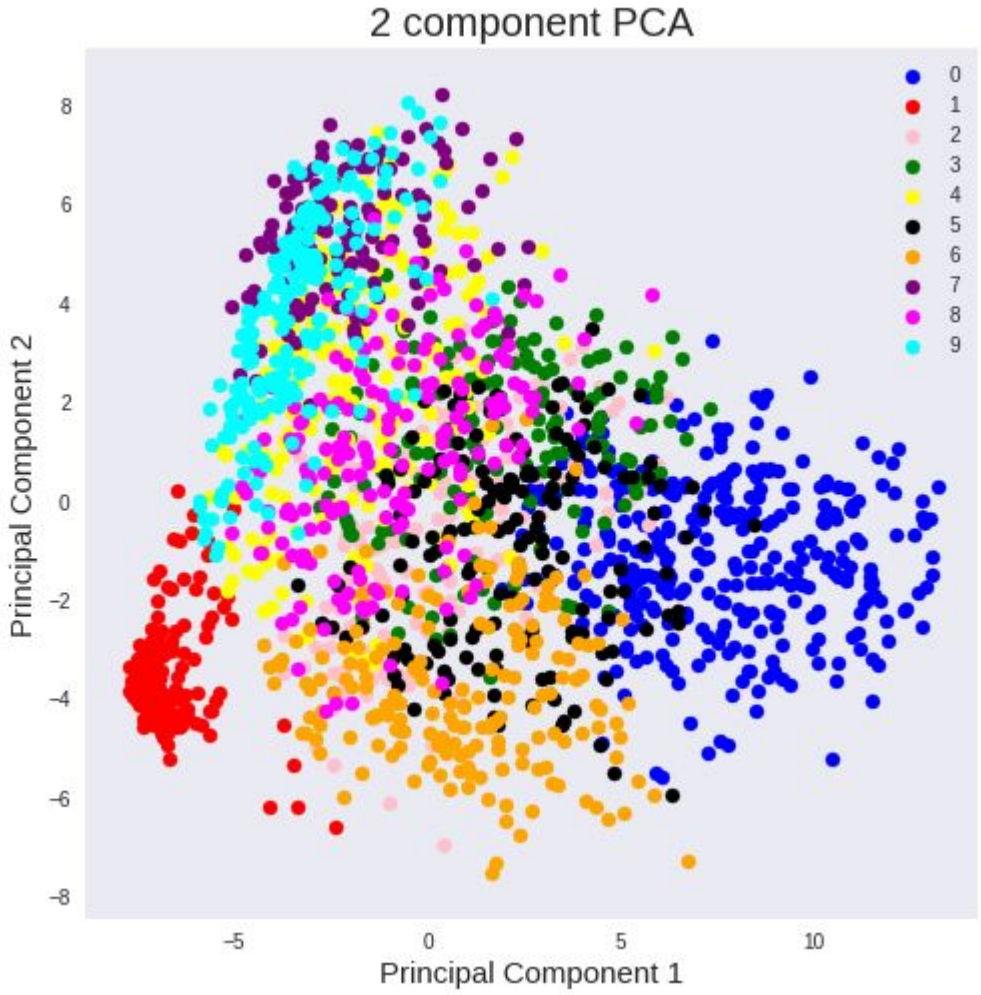
Βήμα 13

α

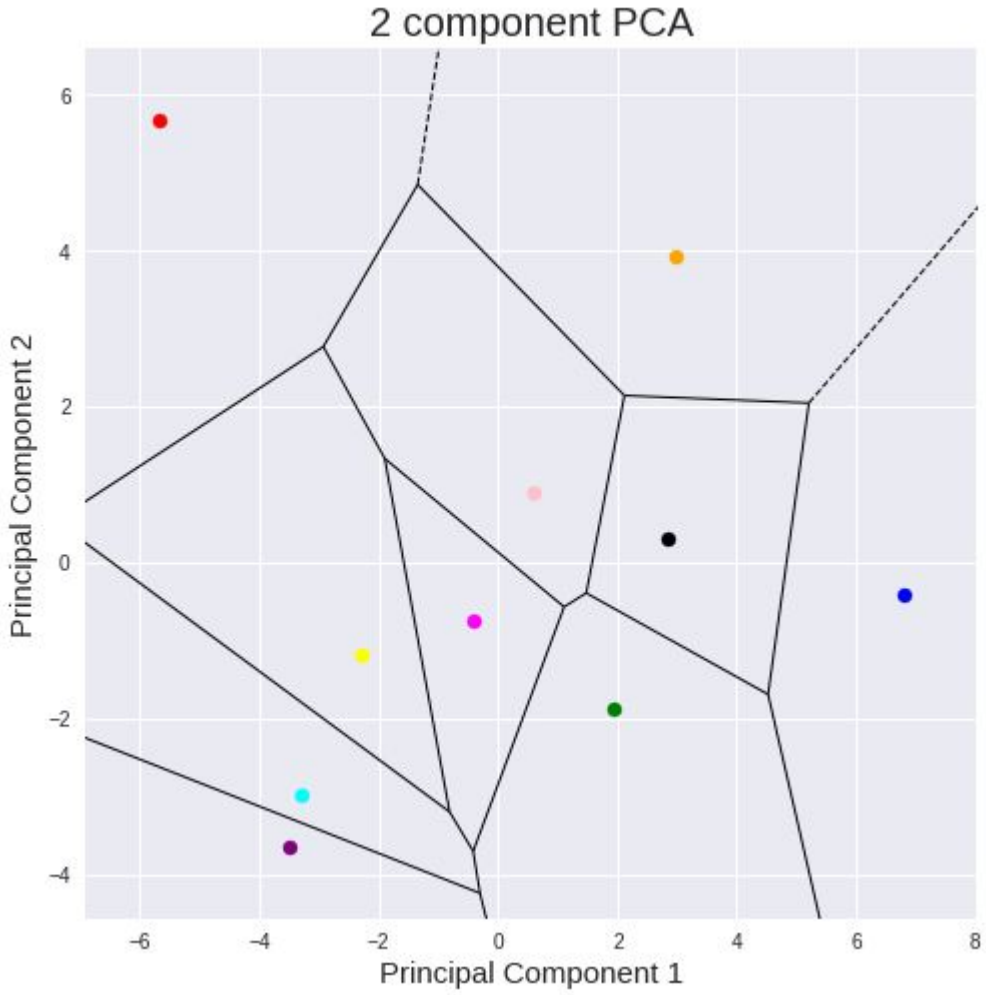
Χρησιμοποιώντας της συνάρτηση cross_val_score() της numpy βιβλιοθήκης και τον ταξινομητή που υλοποιήσαμε στο παραπάνω βήμα και τα train δεδομένα πραγματοποιήσαμε 5-fold cross-validation και υπολογίσαμε τα scores που προκύπτουν. Για να βγει το τελικό σχορε του ταξινομητή υπολογίστηκε η μέση τιμή των 5 παραπάνω τιμών όπου προκύπτει 0.8407163638677286.
Η προκύπτουσα τιμή είναι αναμενόμενη μιας και είναι συμβατή με το ποσοστό επιτυχίας που υπολογίσαμε στο βήμα 11.

β

Εδώ ζητείται ο σχεδιασμός της περιοχή απόφασης του ταξινομητή μας. Για να γίνει εφικτός ο σχεδιασμός δεδομένου πως έχουμε 256 διαστάσεις (όσα και χαρακτηριστικά) οφείλουμε να χρησιμοποιήσουμε κάποια μέθοδο μείωσης διαστατικότητας. Έτσι χρησιμοποιήσαμε την μέθοδο PCA ώστε να γίνει μείωση σε 2 διαστάσεις.
Χρησιμοποιώντας ως labels τον πίνακα y_pred με τις δικές μας ταξινομήσεις, απεικονίζουμε τα test data στον χώρο με βάση τα δύο κυρίαρχα χαρακτηριστικά που προέκυψαν από το PCA και καταλήγουμε στο εξής αποτέλεσμα:



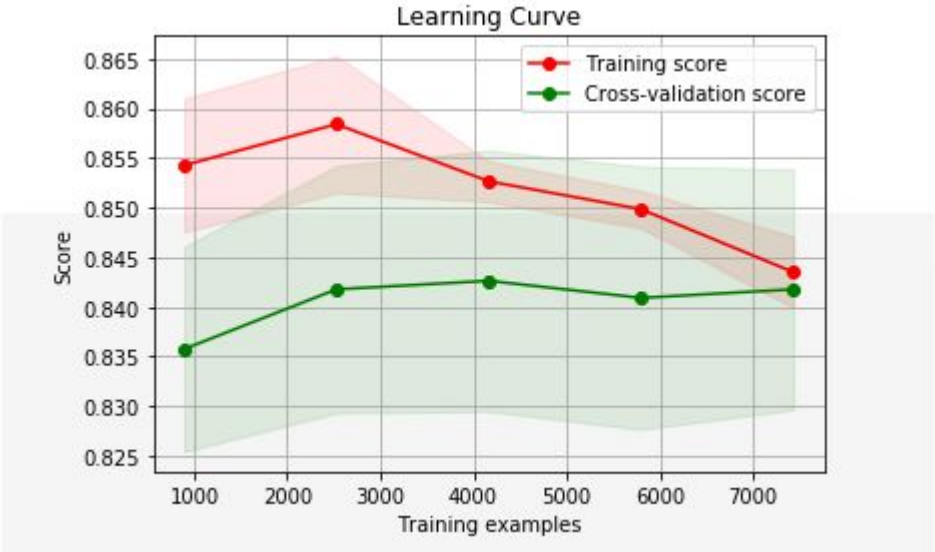
Η εικόνα είναι αρκετά χαώδης μιας και γίνεται απεικόνιση 10 διαφορετικών ψηφίων, ενώ ο αρχικός αριθμός των χαρακτηριστικών με βάση τον οποίο γίνεται η ταξινόμηση ήταν πολύ μεγάλος για να γίνει εμφανής απεικόνιση σε δύο διαστάσεις. Έτσι επαναλήφθηκε το παραπάνω και για τις μέσες τιμές των χαρακτηριστικών του κάθε target label, όπου στη συνέχεια απεικονίστηκαν σε ένα διάγραμμα Voronoi. Σε κάθε μέσο (σημείο) αντιστοιχεί μία περιοχη, η οποία περικλείει τα σημεία του επιπέδου των οποίων η απόσταση από το σημείο αυτό είναι μεγαλύτερη ή ίση από αυτή μεταξύ κάθε άλλου σημείου.



Έτσι δίνεται μια ιδέα για μια πιο ξεκάθαρη απεικόνιση των περιοχών απόφασης.

γ

Εδώ με την βοήθεια της learning_curve και χρησιμοποιώντας 5-fold cross-validation με τα train και test δεδομένα (θεωρήθηκε ότι είχαμε και τα δύο στη διάθεση μας σε ένα συνενωμένο data set), προκύπτουν εύκολα τα score για το training αλλά και για το cross validation.
Υπολογίζοντας την μέση τιμή των παραπάνω, απεικονίζουμε σε διάγραμμα τις καμπύλες εκμάθησης (με σκιές συμβολίζονται οι διακυμάνσεις των τιμών).



Βλέπουμε πως η απόσταση των καμπυλών που δηλώνει το generalization error όλο και μειώνεται και τελικώς οι δύο καμπύλες αρχίζουν και συγκλίνουν . Η απόσταση είναι ακόμη ενδεικτική της διακύμανσης η οποία δεν εμφανίζεται να είναι μεγάλη (low variance).
Τέλος, λόγω της απλότητας του μοντέλου, δεν προσαρμόζεται πλήρως στα training data (underfitting), άρα και για αυτό το score_training δεν είναι πολύ υψηλό. Συμπεραίνουμε λοιπόν ότι με το υπάρχον μοντέλο η αύξηση των δειγμάτων πιθανότατα δεν θα οδηγήσει σε σημαντικά καλύτερη απόδοση.

Βήμα 14

Σε αυτό το βήμα υπολογίζουμε τις a priori πιθανότητες για κάθε κατηγορία, δηλαδή την τιμή

$$\mathbb{P}(X = i),\ \forall i \in 1, 2...10$$

Για τον υπολογισμό αυτών μελετήσαμε την συχνότητα εμφάνισης του κάθε αριθμού, δηλαδή το πόσες φορές εμφανίστηκε στα δεδομένα μας διαιρεμένο με το σύνολο των δειγμάτων μας.

Βήμα 15

α

Σε αυτό το βήμα καλούμαστε να υλοποιήσουμε τον δικό μας Bayesian ταξινομητή. Έτσι υλοποιήθηκε ένας Naive Bayes ταξινομητής υποθέτοντας πως κάθε pixel ακολουθεί μια ανεξάρτητη κατανομή Gauss. Ο ταξινομητής υπολογίζει την πιθανότητα που το δείγμα που εξετάζει ανήκει σε κάθε label και το αντιστοιχεί

σε αυτό με την μεγαλύτερη. Αν συμβολίσουμε Y το label και X τον πίνακα των χαρακτηριστικών, σύμφωνα με τον νόμο του Bayes έχουμε:

$$\mathbb{P}(Y|X) = \frac{\mathbb{P}(X|Y)\mathbb{P}(Y)}{\mathbb{P}(X)}$$

όπου $\mathbb{P}(Y|X)$ η posterior πιθανότητα που θέλουμε και να υπολογίσουμε και $\mathbb{P}(X|Y)$ η a priori πιθανότητα.
Κάνοντας την υπόθεση πως κάθε χαρακτηριστικό (δηλαδή κάθε pixel) είναι ανεξάρτητο, έχουμε πως $\mathbb{P}(X|Y) = \prod_{i=0}^{255} \mathbb{P}(X_i|Y)$.

Έτσι η λύση που ταξινομεί το δείγμα δίνεται από την σχέση $\arg\max_Y \{\mathbb{P}(Y) \prod_{i=0}^{255} \mathbb{P}(X_i|Y)\}$, αφού ο όρος στον παρονομαστή είναι σταθερός οπότε για να βρούμε την μέγιστη τιμή μπορεί να απαλειφθεί.
Εφόσον υλοποιούμε Gaussian Naive Bayes Classifier έχουμε πως η δεσμευμένη πιθανότητα για κάθε χαρακτηριστικό υπολογίζεται από την σχέση:

$$\mathbb{P}(X_i|Y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp^{-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}}$$

Η διασπορά των pixel των ψηφίων έχει υπολογιστεί στο βήμα 9 της προπαρασκευής. Επειδή κάποιες τιμές ήταν μηδενικές κάνουμε smoothing προσθέτοντας μια αμελητέα τιμή στις διασπορές.
Τέλος, δεδομένου πως υπολογίζουμε γινόμενα, μπορούμε να βελτιστοποιήσουμε την υλοποίηση μετατρέποντας την σε αθροίσματα χρησιμοποιώντας **λογαρίθμους**.
Τα παραπάνω ενσωματώθηκαν σε συναρτήσεις fit και predict στην κλάση NaiveBayesianClassifier όπου δημιουργήσαμε, ώστε να έχουμε υλοποίηση συμβατή με scikit-learn, όμοια με τον τρόπο που υλοποιήθηκε το βήμα 12.

Το σκορ που προκύπτει είναι 0.7992027902341804.

β

Χρησιμοποιήσαμε τον έτοιμο Naive Bayes Classifier naive_bayes.GaussianNB της βιβλιοθήκης scikit-learn.
Το εκτιμούμενο σκορ είναι 0.7194818136522172.

γ

Οι δύο υλοποιήσεις ακολουθούν όμοια μεθοδολογία. Βέβαια, οι τιμές των διασπορών που υπολογίσαμε στο βήμα 9 ήταν από τον συνολικό όγκο δειγμάτων train και test. Αυτό και το γεγονός ότι χρησιμοποιήσαμε μεγαλύτερη παράμετρο smoothing (λείανση Gaussianης) είναι υπεύθυνο για την διαφορά στο σκορ των δύο ταξινομητών.

Βήμα 16

Το σκορ του παραπάνω ταξινομητή παίρνοντας όλες τις τιμές διασποράς για όλα τα χαρακτηριστικά για κάθε ψηφίο ίσες με 1 προκύπτει 0.8126557050323866, που είναι σημαντική βελτίωση σε σχέση με τα παραπάνω.
Πιθανόν αυτό συμβαίνει εφόσον δηλώνουμε μια μεγαλύτερη διασπορά, (που υποδηλώνει το πόσο απέχουν οι τιμές από τη μέση τιμή για το κάθε χαρακτηριστικό). Η πραγματική διασπορά, λόγω της φύσης του dataset για κάποια ψηφία προσεγγίζει το 0 και έτσι η κατανομή τείνει στο να γίνει κατακόρυφη. Λειάνοντάς την, ο ταξινομητής είναι προετοιμασμένος για μεγαλύτερες αποκλίσεις και ταξινομεί καλύτερα τα test δεδομένα.

Βήμα 17

Χρησιμοποιήθηκαν οι έτοιμες συναρτήσεις από το scikit-learn για την υλοποίηση Nearest Neighbors και SVM. Έγινε grid search προκειμένου να εντοπιστούν οι καλύτεροι παράμετροι για το μοντέλο μας. Στους κοντινότερους γείτονες, βέλτιστα αποτελέσματα με τη χρήση 5-fold cross validation επίφερε η παράμετρος $nearest\_neighbors = 1$ και κατόπιν $nn = 3$, και στον SVM ο πολυωνυμικός πυρήνας βαθμού 3. Συγκεκριμένα, όπως και σε παραπάνω βήματα, έγιναν fit τα train δεδομένα και μετά καλέστηκε η μέθοδος predict με τα train δεδομένα.
Η Nearest Neighbors υλοποιήθηκε για αριθμό γειτόνων ίσο με 3 και είχε σκορ 0.9469.
Βλέπουμε πως έχουμε ικανοποιητικό αποτέλεσμα. Γενικά, το κύριο πρόβλημα στην μέθοδο KNN είναι το μέγεθος του dataset, μιας και ένα μεγάλο dataset αυξάνει υπερβολικά τον χρόνο που τρέχει ο αλγόριθμος. Ο μικρός αριθμός δεδομένων μας καθιστά την παραπάνω μέθοδο αποδεκτή.

Στη συνέχεια, για την SVM μέθοδο δοκιμάσαμε διάφορες Kernels συναρτήσεις.

- για πολυωνυμική Kernel 3ου βαθμού συνάρτηση είχαμε σκορ 0.95366
- για γραμμική Kernel συνάρτηση είχαμε σκορ 0.92625
- για rbf (radial basis function) Kernel είχαμε σκορ 0.94718

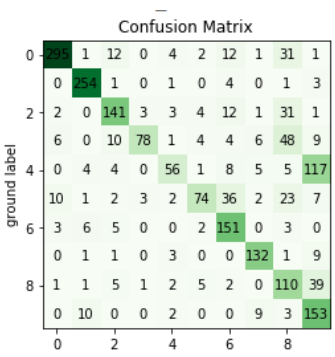
Γενικά βλέπουμε πως η πολυωνυμική συνάρτηση επιφέρει τα βέλτιστα αποτελέσματα Kernel SVM αλλά και ο γραμμικός πυρήνας λόγω του πλήθους των features δίνει ένα αρκετα ικανοποιητικό αποτέλεσμα.

Βήμα 18

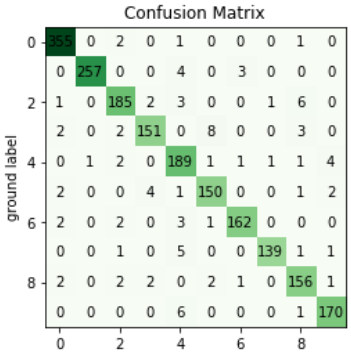
α

Η βασική ιδέα είναι πως συνδυάζοντας ταξινομητές, οι οποίο έχουν διαφορετικό τύπο λαθών, να συνδυαστούν και να δώσουν καλύτερα αποτελέσματα. Η τεχνική αυτή ονομάζεται ensembling και συνδυάζοντας classifiers δημιουργείται ένας

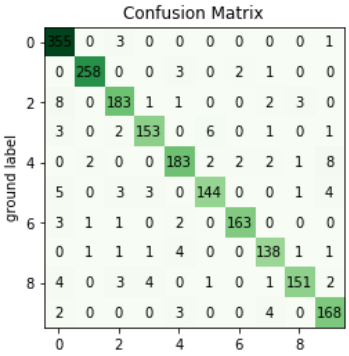
meta-classifier. Ο meta-classifier λειτουργεί με ψήφους, οπότε συνδυάζεται μονός αριθμός ταξινομητών ώστε να αποφεύγονται οι ισοπαλίες.
Συνδυάσαμε Naive Bayes Gaussian, SVM (με πολυωνυμική συνάρτηση Kernel) και 3NN ταξινομητές. Για να εξασφαλίσουμε διαφορετικό τύπο λαθών σχεδιάσαμε τους confusion matrices για τον κάθε ταξινομητή οι οποίοι φαίνονται παρακάτω.



(α′) Naive Bayes Confusion Matrix



(β′) SVM Poly Confusion Matrix



(γ′) KNN Confusion Matrix

Αρχικά εφαρμόσαμε τεχνική hard voting. Αυτό σημαίνει πως κάθε ταξινομητής ταξινομεί το δείγμα, και αν υπάρχει διαφωνία το τελικό label καθορίζεται από απλή ψηφοφορία πλειοψηφίας. Το σκορ του ταξινομητή αυτού προέκυψε 0.94768, που είναι καλύτερο απο τους μεμονωμένους ταξινομητές, με εξαίρεση τον SVM

που είχε το υψηλότερο σκορ.

Στη συνέχεια χρησιμοποιήσαμε τεχνική soft voting. Εδώ έναντι της παραπάνω απλής ψηφοφορίας, λαμβάνουμε από κάθε ταξινομητή την πιθανότητα το κάθε δείγμα να ανήκει σε κάποιο label, και ύστερα συνδυάζονται μεταξύ τους και το δείγμα ταξινομείται όπου παρατηρείται η συνολική μεγαλύτερη συνολική πιθανότητα.
Το σκορ του meta-classifier που δημιουργείται είναι 0.94818

Όπως περιμέναμε, στην περίπτωση του soft voting έχουμε βελτίωση στο σκορ, αν και είναι πολύ μικρή, γεγονός εύλογο αν σκεφτεί κανείς πως έχουμε ήδη πολύ καλό αποτέλεσμα για να υπάρξει μεγαλύτερη βελτίωση.

β

Σε αυτό το βήμα δημιουργείται ένα ensemble μέσω της τεχνικής Bagging. Συγκεκριμένα, χρησιμοποιούνται διαφορετικά μοντέλα τα οποία εκπαιδεύονται σε διαφορετικά υποσύνολα του dataset, ώστε να δημιουργήσουν διαφορετικούς ταξινομητές. Ύστερα αυτοι οι ταξινομητές συνεργάζονται για να κατατάξουν ένα δείγμα, είτε μέσω ψηφοφορίας είτε με τον μέσο όρο των προβλέψεων. Ο λόγος επιτυχίας της μεθόδου αυτή είναι ότι αναμένουμε ότι διαφορετικά μοντέλα θα παράγουν διαφορετικά είδη λαθών(ανεξάρτητα μεταξύ τους).
Στην υλοποιησή μας χρησιμοποιήθηκε η συνάρτηση της scikit-learn όπου χρησιμοποιεί τον μέσο όρο των προβλέψεων για την τελική απόφαση, ενώ ο ταξινομητής στον οποίον εφαρμόθηκε η μέθοδος ήταν ο Nearest Neighbors με αριθμό γειτόνων 3 όπου υλοποιήθηκε στο βήμα 17. Το σκορ που προέκυψε από αυτή την τεχνική ήταν 0.946686, βελτιωμένο σε σχέση με εκείνο του απλού 3NN ταξινομητή.

Βήμα 19

Εδώ χρησιμοποιήθηκε η βιβλιοθήκη pytorch με στόχο την υλοποίηση ενός νευρωνικού δικτύου.

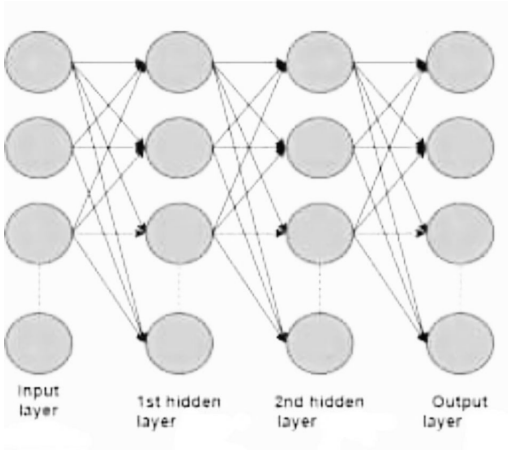
α

Αρχικά, υλοποιήθηκε ένας dataloader ο οποίος διαβάζει τα δεδομένα και τα φέρνει σε μορφή κατάλληλη για επεξεργασία με pytorch.
Ξεκινάμε δημιουργώντας μια κλάση που θα συμβολίζει το dataset μας. Υπάρχει ήδη η abstract κλάση Dataset στην Python, οπότε αρκεί να υλοποιήσουμε και να κάνουμε override τις συναρτήσεις __len__, που επιστρέφει το μήκος του dataset, __getitem__ που επιστρέφει το i-οστό στοιχείο (με όρισμα το i) και την __init__.
Ξεκινάμε με την υλοποίηση της συνάρτησης αρχικοποίησης (init) που διαβάζει το αρχείο με χρήση της βιβλιοθήκης panda.
Η len απλώς επιστρέφει το μήκος του πίνακα, ενώ η getitem παίρνει ως όρισμα

το i και γυρνάει μια τούπλα, με το πρώτο στοιχείο να είναι τα χαρακτηριστικά του στοιχείου σε έναν πίνακα (numpy) και το δεύτερο είναι το label του ψηφίου.

β

Υλοποιήθηκε ένα πλήρες συνδεδεμένο νευρωνικό δίκτυο, ως υποκλάση της nn.Module. Το νευρωνικό που φαίνεται στον κώδικα έχει 4 επίπεδα (το εισόδου, το εξόδου και 2 ενδιάμεσα / 'κρυφά' επίπεδα. Τα 2 ενδιάμεσα επίπεδα μετά απο δοκιμές επιλέξαμε να αποτελούνται από 100 νευρώνες το κάθε ένα. Ποιοτικά τα παραπάνω απεικονίζονται στο παρακάτω σχήμα.



Το επίπεδο εξόδου είναι ένα softmax layer, το οποίο αναπαριστά ενα 10-διάστατο διάνυσμα πιθανοτήτων-η έξοδος του κ-οστου νευρώνα αναπαριστά την πιθανότητα η είσοδος να αναπαριστά το κ-οστο ψηφίο.

$$p_k = \frac{e^{f_k}}{\sum_{k=1}^{10} e_{f_k}}$$

Για την υλοποίηση της κλάσης, κληρονομεί από την nn.Module, και χρειάζεται να γίνει overwrite στις μεθόδους init και forward. Στην init όπου γίνεται η αρχικοποίηση του νευρωνικού δικτύου, δημιουργείται ένα στιγμυότυπο της κλάσης και στην συνέχεια ορίζονται τα επίπεδα καθώς και τα μεγέθη τους.
Στην forward συνάρτηση δίνονται σαν όρισμα τα δεδομένα μας και όρίζονται οι συναρτήσεις ενεργοποίησης ανάμεσα σε κάθε επίπεδο. Στο παράδειγμα που φαίνεται στο κείμενο χρησιμοποιείται η ReLU, η οποία ορίζεται ως $f(x) = x^+ = max(x, 0)$ και επιλύει το πρόβλημα του κορεσμού που ενδεχομένως προκαλούν οι σιγμοειδείς συναρτήσεις. Στο τέλος επιστρέφεται λογαριθμική softmax συνάρτηση ενεργοποίησης, η οποία ορίζεται ως ο λογάριθμος της ποσότητας.
Αφού ορίστηκε το νευρωνικό συνεχίζουμε στην εκπαίδευση του. Ξεκινάμε υλοποιώντας έναν βελτιστοποιητή επικλινούς μεθόδου (stochastic gradient descent

optimize) με ρυθμό εκμάθησης 0.01 και momentum 0.9. Ορίζουμε ακόμη την συνάρτηση απωλειών (ως τον αρνητικό λογάριθμο της πιθανότητας απώλειας).

$$L(y) = -\log(y)$$

Έτσι ξεκινάμε την εκπαίδευση. Χρησιμοποιούμε τον dataloader που ορίσαμε παραπάνω όπου χωρίζει τα δεδομένα μας σε batches και τα προμηθεύει στο νευρωνικό. Οπότε έχουμε δύο βρόχους, έναν που ορίζει τις εποχές (στην περίπτωση μας 20) και ένας που διατρέχει τα δεδομένα μας σε batches. Στον παρόν κώδικα τα δεδομένα χωρίζονται σε 32 batches. Μέσα στον βρόχο, διαδοχικά μηδενίζουμε τις κλίσεις (gradients), καλείται η forward μέθοδος και υπολογίζεται το λάθος του αποτελέσματος μας. Τέλος γίνεται ένα backwards pass όπου επανυπολογίζονται οι κλίσεις (gradients) και ενημερώνονται τα βάρη του νευρωνικού.

Συνεχίζοντας, ελέγχουμε το νευρωνικό στα test δεδομένα με στόχο να μελετήσουμε την απόδοση του. Επεξεργαζόμαστε τα δεδομένα με τον ορισμένο dataloader και δουλεύοντας με batches, αφού καλέσουμε το νευρωνικό, υπολογίζουμε την απόκλιση και ελέγχουμε αν είναι το σωστό αποτέλεσμα.

Για τις δεδομένες παραμέτρους (20 εποχές, 32 batches, 4 επίπεδα, ReLU συνάρτηση ενεργοποιησης) είχαμε μέσο σφάλμα 0.0101 και 94% ακρίβεια.
Γίνανε δοκιμές και με άλλες παραμέτους -διαφορετικό αριθμό batches, διαφορετικός αριθμός επιπέδων, περισσότερες εποχές, διαφορετικές συναρτήσεις ενεργοποίησης όπως οι relu6, η σιγμοϊδής, η softplus, η *tanh*-, συνδυασμός τον οποίων έδωσε ίδια ή χειρότερη απόδοση.

δ

Όπως είδαμε και παραπάνω, το νευρώνικο δίκτυο που υλοποιήθηκε είχε ακρίβεια 94%, σκορ αρκετά καλό δεδομένου του μεγέθος των δεδομένων μας. Παρόλα αυτά, σε ορισμένους απλούς ταξινομητές εντοπίστηκε καλύτερη απόδοση. Αυτό συμβαίνει δεδομένου πως το πρόβλημα δεν είχε πολύ μεγάλη πολυπλοκότητα ενώ ο αριθμός των δεδομένων μας ήταν περιορισμένος. Συνεπώς, απλούστερα μοντέλα ήταν ικανά να ταξινομήσουν επαρκώς τα δεδομένα, ενώ το νευρωνικό δεν είχε τον απαραίτητο αριθμό δειγμάτων για να φτάσει στην μέγιστη επίδοσή του.