1η Εργαστηριακή Άσκηση Αναγνώρισης Προτύπων

Η εργασία αυτή εκπονήθηκε από τους φοιτητές της ΣΗΜΜΥ Κωνσταντίνο Τσαούση και Κωνσταντίνο Κωστόπουλο με ΑΜ 03117652 και 03117043 αντίστοιχα.

Βήμα 1

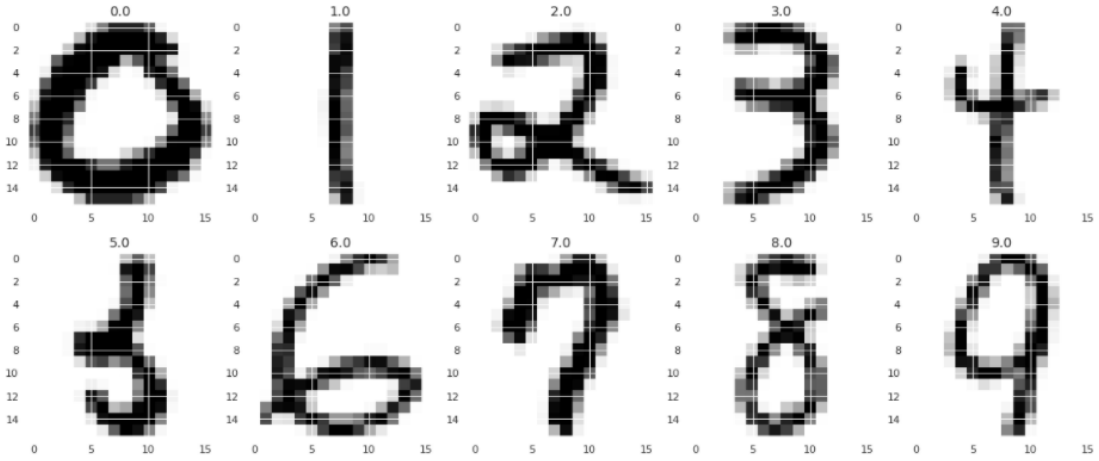
Χρησιμοποιώντας την *open* και την *split* της python διαβάζουμε τα αρχεία και τα τοποθετούμε σε πίνακες. Τα pixel values κυμαίνονται από -1 εώς +1, στον πραγματικό άξονα. Τα labels (το πρώτο στοιχείο της κάθε γραμμής του αρχείου) είναι ακέραιοι από 0 μέχρι το 9.

Βήμα 2

Εκτυπώνουμε την 131η εικόνα του train set. Ο αριθμός που βλέπουμε είναι ο 9 (εννιά)

Βήμα 3

Εντοπίζουμε τη θέση των ψηφίων κάθε label, και επιλέγουμε με τη *random.choice* κάποιο από αυτά. Προκύπτει:



Βήμα 4

Το pixel (10,10) για το ψηφίο 0 ανήκει σε μία κεντρική θέση. Η μέση τιμή είναι -0.504. Παρατηρούμε ότι η mean τιμή για το pixel (10,10) είναι πολύ μικρή (Κλίμακα [-1,1]). Αυτό είναι λογικό και αναμενόμενο δεδομένου του ότι πρόκειται για απεικονίσεις του ψηφίου 0, το οποίο έχει κενό στο κέντρο.

Βήμα 5

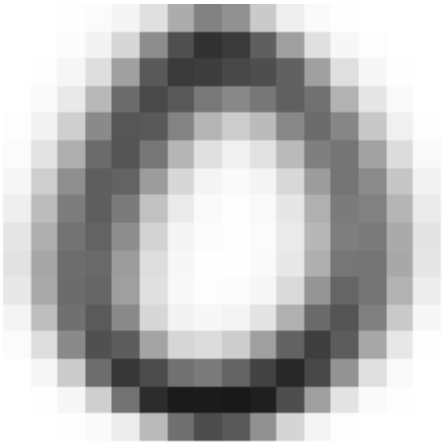
Χρησιμοποιώντας την built-in συνάρτηση *std* προκύπτει διασπορά 0.413.

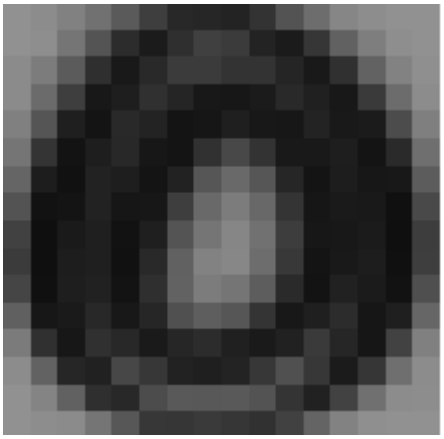
Βήμα 6

Ομοίως με τα παραπάνω βήματα υπολογίζουμε την μέση τιμή και την διασπόρα όλων των χαρακτηριστικών των pixel για το ψηφίο 0. Στο τέλος τυπώνοντας τις τιμές για το pixel (10,10) έχουμε τα ίδια αποτελέσματα.

Βήμα 7/8

Η απεικόνιση του ψηφίου 0 με τις τιμές της μέσης τιμής είναι πιο συμμετρική σε σύγκριση με των δειγμάτων και παρουσιάζει ομαλή διακύμανση στις τιμές. Επίσης, το πάχος είναι ενιαίο και έχει αυξηθεί. Η απεικόνιση με βάση τις τιμές της διασποράς έχει παρόμοια μορφή με της μέσης τιμής. Τα χρώματα είναι πιο έντονα, αφού η διασπορά έχει μεγαλύτερες τιμές εκεί που η τιμή των pixel είναι μεγαλύτερη.





Βήμα 9

Με μια for εφαρμόζουμε προηγούμενα σημεία του κώδικα για όλα τα ψηφία. Τελικά:

Βήμα 10

Χρησιμοποιώντας την ευκλείδια απόσταση, δηλαδή την νόρμα l2 και παίρνοντας τις εικόνες με τις μέσες τιμές των pixels του test set ως κέντρα, ταξινομούμε την 101η εικόνα με το label που ελαχιστοποιεί αυτή την νόρμα. Στην προκειμένη περίπτωση θέλουμε να είναι το 0. Παρατηρούμε ότι η πρόβλεψη είναι η σωστή.

Βήμα 11

Εφαρμόζοντας την ίδια τεχνική με το προηγούμενο ερώτημα ταξινομούμε όλα τα ψηφία των test δεδομένων σε μία από τις 10 κατηγορίες και έχουμε ποσοστό επιτυχίας 81.4% επί του συνολικού test set.

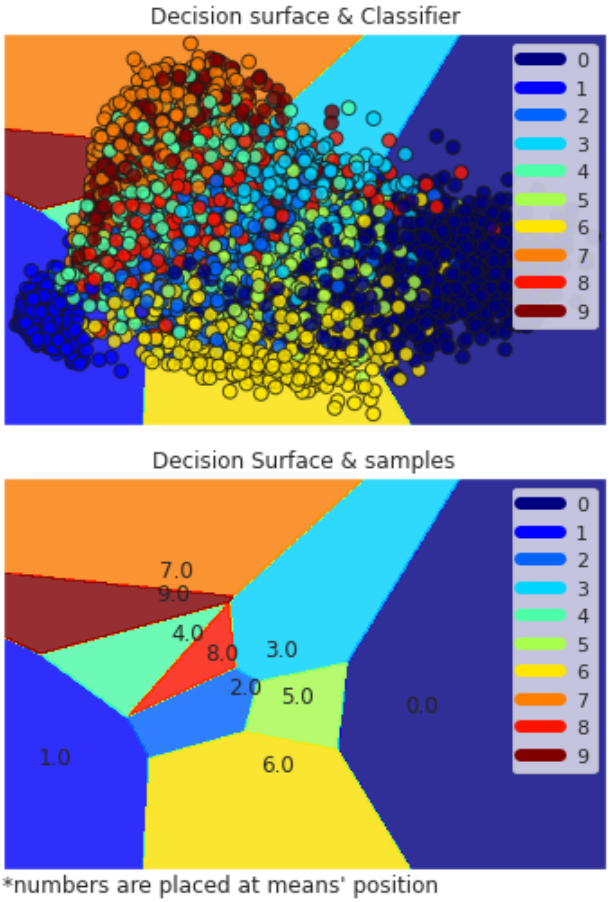
Βήμα 12

Υλοποιούμε τον ταξινομητή ευκλείδιας απόστασης σαν ένα scikit-learn estimator. Θα χρησιμοποιήσουμε τους ίδιους αλγόριθμους με τα προηγούμενα ερωτήματα. Στην συνάρτηση fit υπολογίζουμε την μέση τιμή όλων των κατηγοριών και το αποθηκεύουμε στο self.X_mean_. Στην predict όπως και το ερώτημα 11(α) ταξινομούμε όλα τα ψηφία που θα δίνονται στη συνάρτηση με βάση την Ευκλείδια απόσταση και επιστρέφει σε ποια κατηγορία ανήκει το κάθε ψηφίο. Στην score, αρχικά κάνουμε predict και έπειτα όπως στο ερώτημα 11(β) υπολογίζουμε το ποσοστό επιτυχίας.

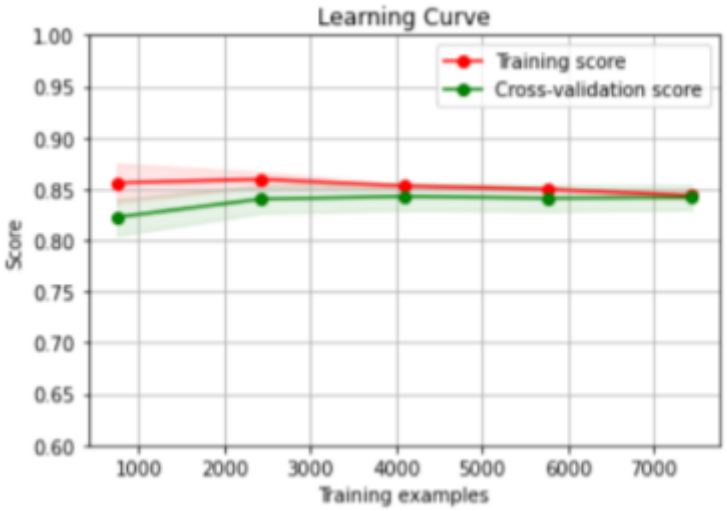
Βήμα 13

a) Χρησιμοποιούμε μόνο το train set για να υπολογίσουμε την απόδοση του classifier, καθώς δεν έχουμε πρόσβαση σε test data σε real world scenarios. Το 5-fold cross-validation (το οποίο χρησιμοποιούμε για τη μέτρηση αυτή) έγκειται στη διαίρεση του συνόλου δεδομένων σε 5 σετ. Κάθε υποσύνολο περιέχει διαφορετικές παρατηρήσεις. Η επιλογή των υποσυνόλων είναι τυχαία. Ένα από τα υποσύνολα χρησιμοποιείται ως σύνολο επικύρωσης και τα υπόλοιπα τέσσερα συνενώνονται και δημιουργούν το σύνολο εκπαίδευσης. Το μοντέλο εκπαιδεύεται χρησιμοποιώντας το σύνολο εκπαίδευσης και δοκιμάζεται έναντι του συνόλου επικύρωσης. Η διαδικασία επαναλαμβάνεται πέντε φορές, κάθε φορά χρησιμοποιώντας ένα διαφορετικό σύνολο ως σύνολο επικύρωσης και τα υπόλοιπα τέσσερα ως σύνολο εκπαίδευσης. Στο τέλος υπολογίζεται η μέση επίδοση του μοντέλου. Στον αλγόριθμο μας χρησιμοποιήσαμε την έτοιμη συνάρτηση cross_validate, και λάβαμε score 84%.

b) Όσον αφορά την περιοχή απόφασης του Euclidian Classifier, για να την υπολογίσουμε θα πρέπει να μειώσουμε τις 256 διαστάσεις σε 2. Αυτό το πετυχαίνουμε εφαρμόζοντας Principal Component Analysis (PCA).



c) Σχεδιάζουμε την καμπύλη εκμάθησης του ευκλείδειου ταξινομητή. Καθώς αυξάνονται τα δείγματα παρατηρούμε ότι το training score μειώνεται ενώ το cross validation score αυξάνει. Από τα 4000 δείγματα υπάρχει κορεσμός και το training σταματά, καθώς η αύξηση του score είναι αμελητέα.

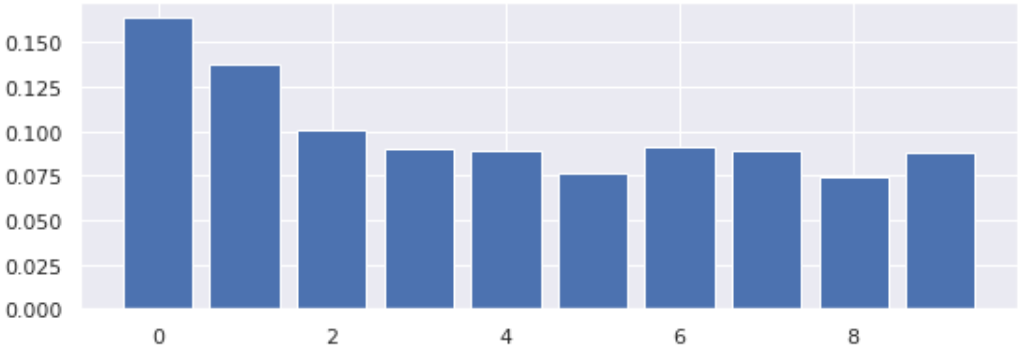


Βήμα 14

Ο υπολογισμός των a-priori πιθανοτήτων για την κάθε κλάση, πραγματοποιείται ως εξής. Προστίθενται οι φορές που εμφανίζεται η κάθε κατηγορία ψηφίου και την αποθηκεύουμε σε πίνακα. Τέλος, ο πίνακας αυτός διαιρείται με το συνολικό αριθμό των δειγμάτων, προκειμένου να εξαχθούν οι πιθανότητες για κάθε κλάση.

Δηλαδή από το Y_train βρίσκουμε για κάθε κλάση c το $P(Y_c) = \frac{N_c}{N}$. Προφανώς, περιμένουμε το άθροισμα τους να ισούται με 1.

Παρατηρούμε ότι τα samples με label 0 ειναι πολλά στο train σετ (και τα 1), με αποτέλεσμα η απριορι να ειναι μεγαλη, στα υπολοιπα οι απριορι ειναι καπως εξισταθμισμενες.



Βήμα 15

Α)

Η βασική ιδέα λειτουργίας του ταξινομητή είναι α) ο γνωστός νόμος του Bayes

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

β) η (naive) υπόθεση ότι τα χαρακτηριστικά είναι όλα ανεξάρτητα μεταξύ τους.

Ο Gaussian Naive Bayes ταξινομητής, δεδομένης μιας εικόνας, θεωρεί πως όλα τα pixels είναι ανεξάρτητα μεταξύ τους και κάθε ένα από αυτά ακολουθεί κανονική κατανομή. Συνεπώς, υπολογισμένων της μέσης τιμής και διασποράς για κάθε ένα από τα δέκα ψηφία, κάθε pixel μοντελοποιείται ως ανεξάρτητη κανονική κατανομή. Στην πραγματικότητα, η υπόθεση περί ανεξαρτησίας των εικονοστοιχείων της εικόνας δεν ισχύει (πχ. για τα γειτονικά pixel) ή τουλάχιστον είναι αρκετά απλοϊκή, ωστόσο έχει παρατηρηθεί πως ο ταξινομητής αυτός σημειώνει υψηλά ποσοστά ακρίβειας και για την ταξινόμηση ψηφίων.

Για μια μεταβλητή κλάσης y και ένα εξαρτώμενο διάνυσμα χαρακτηριστικών x1 μέχρι xb συμφωνα με το θεωρημα bayes ισχυει $P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots x_n \mid y)}{P(x_1, \ldots, x_n)}$

Με δεδομενη εισοδο το P(x1,...,χν) ειναι σταθερο συνεπως μπορουμε να χρησιμοποιησουμε

τον ακόλουθο κανόνα ταξινόμησης για τον Naive Bayes ταξινομητή:

$$\hat{y} = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

όπου P(y) είναι η a-priori πιθανότητα (σχετικη συχνοτητα του label y στο training set-υποθεση μας) για την εκάστοτε κλάση, ενώ οι πιθανότητες P(xi |y) δηλαδή η πιθανοφάνεια δηλαδη η πιθανοτητα του δειγματος με δεδομενη την υπόθεση μας και στην προκειμένη περίπτωση αφορούν το κάθε ένα εκ των 256 pixels δεδομένης της κάθε κλάσης, υπολογίζονται βάσει της κανονικής κατανομής:

(1)

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

όπου ξ είναι το pixel για την εικόνα που εξετάζουμε (0-255), μ_y είναι η μέση τιμή του για την εκάστοτε κλάση και σy είναι η τυπική του απόκλιση η οποία εκφράζει την τετραγωνική ρίζα της διακύμανσης. Κατά συνέπεια, το γινόμενο των δεσμευμένων πιθανοτήτων εξάγεται υπολογίζοντας το γινόμενο των κανονικών κατανομών κάθε εικονοστοιχείου, σύμφωνα με την μέση τιμή και στην διασπορά τους για κάθε κλάση.

Τοποθετώντας όλες τις κανονικές κατανομές για κάθε pixel σε έναν πίνακα και υπολογίζοντας το γινόμενό του έχουμε το ζητούμενο. Ωστόσο, για την επίτευξη της παραπάνω διαδικασίας, απαιτείται το σ και κατ' επέκταση η διακύμανση να είναι διάφορο του 0, εφόσον εμφανίζονται στον παρονομαστή της σχέσης. Η παραπάνω προϋπόθεση δεν ισχύει για το σύνολο των ψηφίων. Ακόμα, παρατηρήθηκε πως η

διακύμανση ορισμένων πιξελς ήταν πολύ μικρή και άρα η τυπική απόκλιση ακόμα μικρότερη, με συνέπεια να απειρίζουν τους δύο λόγους. Για να το αποφύγουμε αυτό προσθέτουμε σε όλες τις αποκλίσεις ένα κοινό μικρό offset της τάξης του 10^-6.

Ισοδύναμα μπορούμε να επιλέξουμε το label που έχει τη μεγαλύτερη λογαριθμική πιθανοφάνεια, δηλαδή να πάρουμε το λογάριθμο της έκφρασης (1)

$$\hat{y} = \text{argmax}_y\{log(P(y)) + \sum_{i=1}^{n} log(P(x_i \mid y))\} \quad = \quad \text{argmax}_y\{log(P(y)) + \sum_{i=1}^{n} log(\frac{1}{\sqrt{2\pi\sigma_y^2}}) + \sum_{i=1}^{n}\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)\}$$

μετασχηματισμός αυτός βολεύει γιατί πλέον υπολογίζουμε αθροίσματα αντί για γινόμενα και έχουμε μεγαλύτερη αριθμητική ευστάθεια, και στην υλοποιηση μας εχουμε χρησιμοποιησει αυτο τον τροπο.

Προκειμένου η υλοποίηση του Naive Bayesian ταξινομητή να είναι συμβατή με το scikitlearn, απαιτείται η κλάση CustomNBClassifier() να περιέχει τις συναρτήσεις :

- toNp() που μια λίστα γίνεται np.array
- fit() που υπολογίζεται η μέση τιμή και η διασπορά των χαρακτηριστικών για όλα τα ψηφία (0-9)
- predict() βάσει των α-πριορι πιθανοτήτων και γινομένων των κανονικών κατανομών
- score() που συγκρίνει τις προβλέψεις με τις πραγματικές τιμές και μας δίνει την ακρίβεια (accuracy)

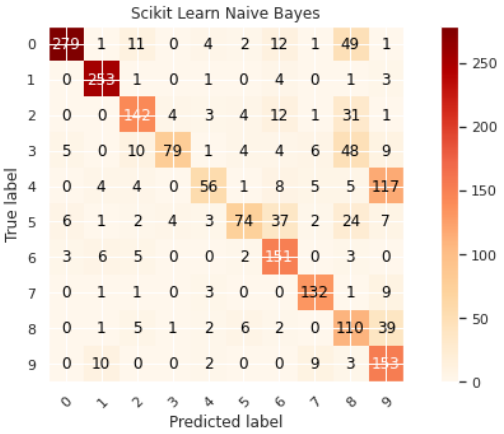
Β)

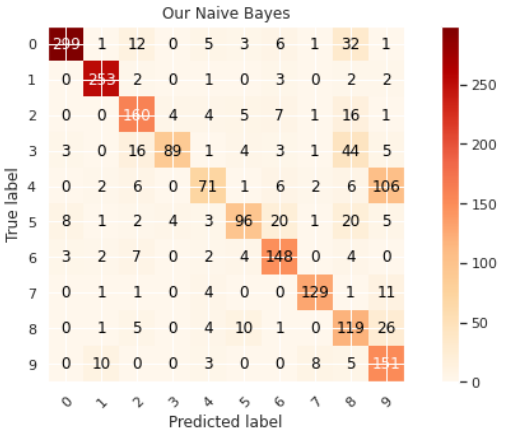
Το score υπολογίστηκε 0.7785, δηλαδή χειρότερο απο τον ταξινομητή Ευκλείδιας απόστασης .

Σχόλια: Παρατηρώντας τον τύπο και το πως δουλέυει ο ταξινομητής Bayes εύλογα θα μπορούσε κάποιος να πει ότι αποτελεί μια γενίκευση αυτού της Ευκλείδιας Απόστασης, με διαφορά οτι ο πρώτος λαμβάνει υπόψη του και τις κατανομές/διασπορές αλλά και της α πριορι πιθανότητες. Αν αυτές είναι ίδιες για όλα τα δείγματα και τα χαρακτηριστικά τους, τότε δουλεύον με τον ίδιο τρόπο. Η επιδείνωση σε σχέση με αυτον της Ευκλείδιας Απόστασης εχει πιθανη αιτια την αφελη προσεγγιση οτι οι κατανομες των Pixel ειναι iid η οποια δεν ισχυει οπως ειπωθηκε και παραπανω

Γ)

Η έτοιμη υλοποίηση του scikit-learn, δίνει score 0.712, αισθητά χαμηλότερο από τη δική μας υλοποίηση που είναι τώρα 0.7438. Βλέπουμε και από τις παρακάτω φωτογραφίες ότι το δικό μας και το Gaussian NB Classifier και τα δυο έχουν θέματα σύγχυσης στα ζεύγη ψηφίων 4 – 9 και 3 – 8. Φαίνεται, μάλιστα ότι το δικό μας είναι σχετικά καλύτερο, γεγονός που πιθανότατα προκλήθηκε απο τη διαφορετική υλοποίηση στα 0 stds.

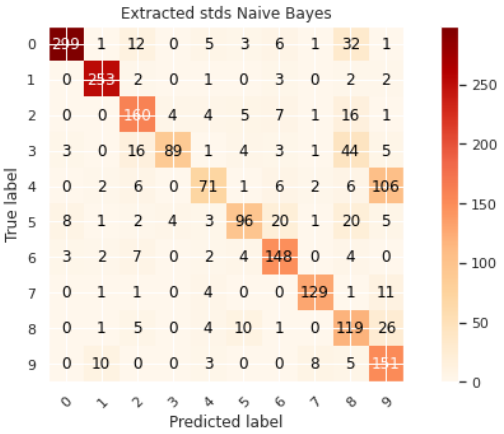


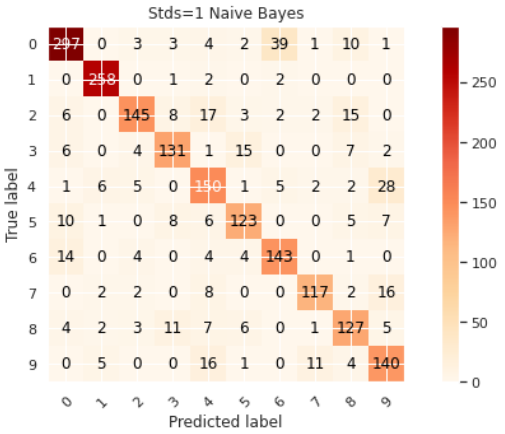


Βήμα 16

Έπειτα από την μικρή αλλαγή στον κώδικα (for k in range(256): var[k] = 1), το score που λαμβάνουμε ισούται με 0.8127 το οποίο υπερβαίνει κατα λίγο το αρχικό μας score, ενώ δοκιμάζοντας τη μέθοδο 5-fold-cross-validation λαμβάνουμε score 0.8405. Συνεπώς, η διακύμανση που τέθηκε ίση με ένα επηρέασε το score, αυξάνοντάς το.

Με διακύμανση 1 ο ταξινομητής δουλεύει σχεδόν σαν τον Euclidean με τη μόνη διαφορά την αφαίρεση των λογαριθμημένων a priori. Παρατηρούμε ότι ο ταξινομητής δουλεύει πολύ καλύτερα με std = 1, σημαντικά ελαχιστοποιώντας τις λανθασμένες προβλέψεις στα ζεύγη αριθμών 4 – 9 και 3 – 8.





Βήμα 17

Προκειμένου να συγκρίνουμε την επίδοση των ταξινομητών Naive Bayes, Nearest Neighbors, SVM (με linear και rbf kernels) εφαρμόζουμε την μέθοδο 5-fold-cross-validation, εντοπίζουμε τον ταξινομητή με την καλύτερη επίδοση και τον δοκιμάζουμε στο test set

και μετά στο train set. Τα scores για τους παραπάνω ταξινομητές με την μέθοδο 5-fold-cross-validation είναι τα ακόλουθα:

Για το KNN: Πειραματιζόμενοι με διαφορετικές τιμές του n, παρατηρούμε μια μείωση της ακρίβειας καθώς φτάνουμε σε μεγαλύτερους αριθμούς. Αυτό σημαίνει ότι η βαθμολογία είναι καλύτερη με μεγαλύτερη διακύμανση (πιο κοντά στο 1) , ενώ μπορούμε να συμπεράνουμε ότι το σύνολο εκπαίδευσης και το σύνολο δοκιμής έχουν παρόμοιες κατανομές.

Για το SVM: Καλύτερη απόδοση έχουν τα polynomial και έπειτα τα rbf, ενώ παρατηρούμε ότι τα linear δεν είναι τόσο καλά από άποψη ευστοχίας. Όσο ανεβαίνει το C μειώνεται το accuracy, συνεπώς προτιμούμε μικρές τιμές. Τέλος, για σιγμοειδή πυρήνα παρατηρούμε ότι τα αποτελέσματα είναι αρκετά απογοητευτικά.

- GaussianNB(smoothing = 0.1) : 0.8340
- KNeighborsClassifier(n_neighbors = 10) : 0.9352
- SVC(kernel = ”linear”) : 0.9555
- SVC(kernel = ”rbf”) : 0.9680
- SVC(kernel = ”poly”) : 0.9640
- SVC(kernel = ”sigmoid”) : 0.8382

Τα αντίστοιχα scores στο test set είναι ίσα με:

- GaussianNB(smoothing = 0.1) : 0.8156
- KNeighborsClassifier(n_neighbors = 10) : 0.9545
- SVC(kernel = ”linear”) : 0.9469
- SVC(kernel = ”rbf”) : 0.9720
- SVC(kernel = ”poly”) : 0.9690
- SVC(kernel = ”rbf”) : 0.8425

Παρατηρούμε, ότι όλοι οι ταξινομητές πετυχαίνουν υψηλά ποσοστά ακριβείας. Δεν θα μπορούσαμε να προβλέψουμε ποιος θα ήταν ο καλύτερος, λόγω των χειρόγραφων ψηφίων. Στην προκειμένη περίπτωση, για το υποσύνολο αυτό του dataset ο ταξινομητής SVM με linear kernel, ο οποίος σημείωσε το υψηλότερο score με την μέθοδο 5-fold-cross-validation εμφανίζει και το υψηλότερο score στο test set.

Βήμα 18

A)

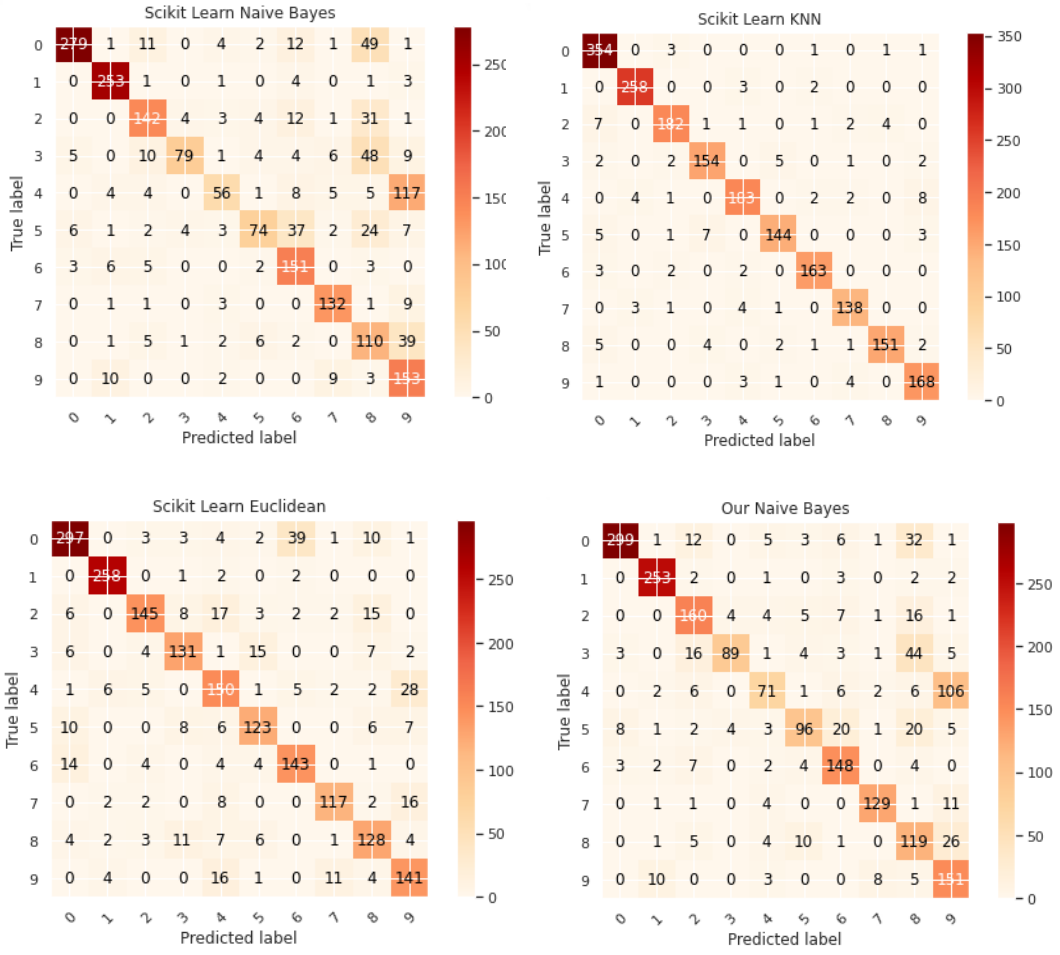
Πριν συνδυάσουμε τους ταξινομητές μας για να έχουμε τα καλύτερα αποτελέσματα, πρέπει να αναλύσουμε καλύτερα τα αποτελέσματά μας και να επιλέξουμε ποιος παράγοντας χρήζει βελτίωσης, όπως ποια κλάση γίνεται συνήθως λανθασμένα προβλέψιμη κ.λπ.

Για να αποκτήσουμε καλύτερη εικόνα των δεδομένων και των κανόνων ταξινόμησης, θα χρησιμοποιήσουμε τη λεπτομερή μετρική του πίνακα σύγχυσης, η οποία δείχνει τον αριθμό των δειγμάτων που δεν ταξινομούνται σωστά σε κάθε κλάση.

Κάθε γραμμή του πίνακα αντιπροσωπεύει τα δείγματα σε μια προβλεπόμενη κλάση, ενώ κάθε στήλη αντιπροσωπεύει τα δείγματα σε μια πραγματική κλάση (ή αντίστροφα).

Με την μέθοδο ensemble χρησιμοποιούνται συνδυασμοί ταξινομητών για τη βελτίωση των επιμέρους επιδόσεων. Αναμένουμε η προσέγγιση αυτή να μειώσει σημαντικά τον κίνδυνο ενός ιδιαίτερα κακού αποτελέσματος, καθώς και το σφάλμα πρόβλεψης, λόγω του συνδυασμού αποδοτικών μοντέλων τα οποία κάνουν διαφορετικά λάθη μεταξύ τους. Για τον εντοπισμό των διαφορετικών λαθών μεταξύ των ταξινομητών, τυπώνουμε τους confusion matrices τους. Χρησιμοποιούμε χάρτες θερμότητας για να έχουμε καλύτερη εικόνα του f1 score για κάθε ψηφίο.

Ο Gaussian βρίσκεται άνω αριστερά, ο KNN άνω δεξιά, ο Euclidean κάτω αριστερά και ο CustomNB κάτω δεξιά.



Οι Naive Bayes, KNN, CustomNB βλέπουν ως μεγαλύτερη αστοχία το ζεύγος 4 – 9 κι ο Euclidean το 0 – 6.

Λαμβάνουμε τους ταξινομητές KNN για n = 1,3, SVC poly για C = 10, 100 και τον rbf με C = 100.

Συνδυάζουμε τους ταξινομητές με το VotingClassifier του scikit-learn και μας παρέχεται επιλογή ανάμεσα σε hard και soft voting. Στην περίπτωση του hard voting (majority voting), καθε ταξινομητής ψηφίζει σε ποια κλάση θα κατατάξει το ψηφίο και η απόφαση που λαμβάνεται εκφράζει την απόφαση της πλειοψηφίας. Προκειμένου λοιπόν να έχουμε πλειοψηφία και όχι ισοπαλία, αναγκαίος κρίνεται ο μονός αριθμός των ταξινομητών που συνδυάζονται. Στην περίπτωση του soft voting, κάθε ταξινομητής δεν προβλέπει ένα label, αλλά μια πιθανότητα για (κατανομή πιθανότητας) την οποία ένα δείγμα ανήκει σε κάποια κλάση. Για παράδειγμα, για ένα συγκεκριμένο test δείγμα, το soft voting μπορεί να δίνει 30% πιθανότητα το test δείγμα να εντάσσεται στην κλάση 0 και 70% πιθανότητα να εντάσσεται στην κλάση 8. Αθροίζουμε, άρα, τις προβλεπόμενες πιθανότητες για κάθε label και δίνουμε ως αποτέλεσμα το label με την υψηλότερη πιθανότητα.

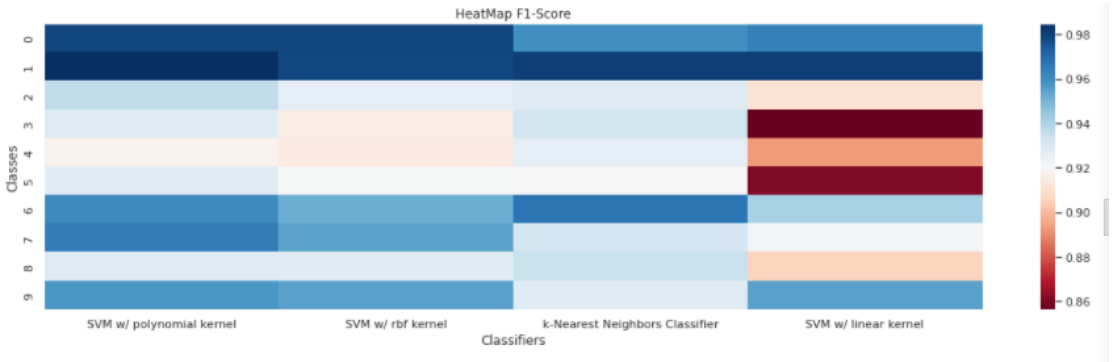
Το score που λαμβάνουμε για τα test δεδομένα για τον παραπάνω συνδυασμό με hard voting είναι ίσο με 0.9501 όσο περίπου αυτό του rbf kernel SVM. Με soft voting το score που λαμβάνουμε για τα test δεδομένα είναι ίσο με 0.9541. Όπως αναμέναμε, το hard voting έδωσε χαμηλότερο score.

B)

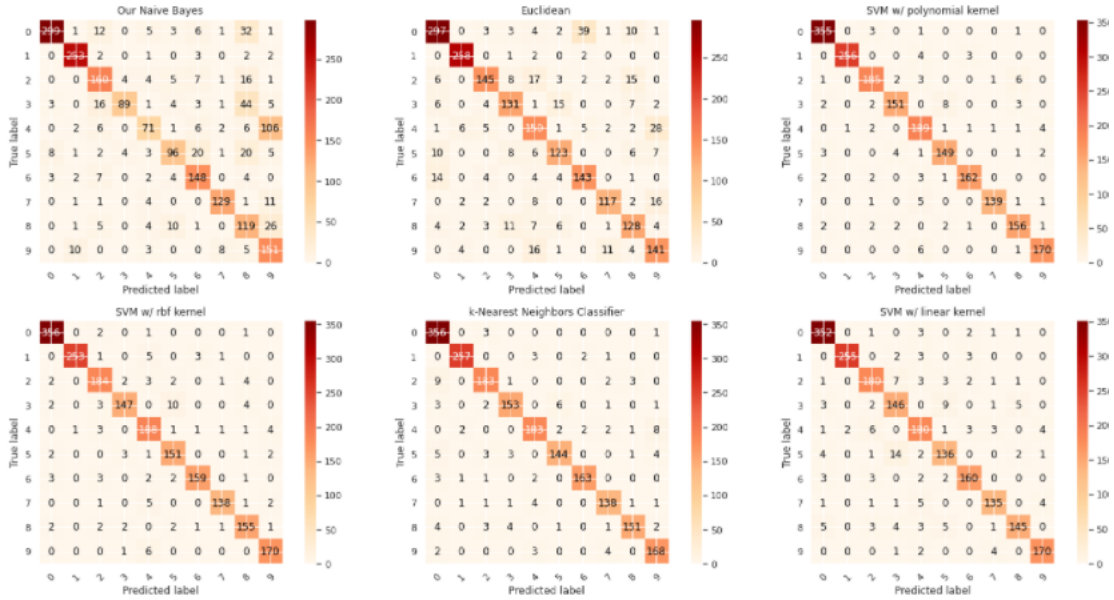
Ο αλγόριθμος bagging που, χαρακτηρίζεται ως αλγόριθμος παράλληλου συνδυασμού κατηγοριοποιητών, έχοντας δημιουργηθεί πολλαπλά σύνολα εκπαίδευσης από το αρχικό σύνολο δεδομένων, για κάθε ένα από αυτά εκπαιδεύει ένα διαφορετικό μοντέλο, με αποτέλεσμα, ο κάθε ταξινομητής να μην έχει εικόνα του συνόλου της πληροφορίας, αλλά για ένα μικρό μέρος της. Η τελική απόφαση κατηγοριοποίησης μιας νέας παρατήρησης λαμβάνεται με κάποια μορφή συνάθροισης των προβλέψεων των επιμέρους μοντέλων. Η παρατήρηση εκχωρείται στην κλάση που συγκέντρωσε τις περισσότερες ψήφους. Το bagging είναι κατάλληλο για ασταθείς ταξινομητές, δηλαδή όταν διαφορετικά tests μπορεί να δώσουν έντονα διαφορετικές απαντήσεις, και σκοπεύουν να χαμηλώσουν το variance. Ωστόσο, στην άσκησή μας έχουμε το αντίθετο, δηλαδή με χρήση του bagging παίρνουμε high variance αφού οι ταξινομητές μας είναι αρκετά σταθεροί. Επιλέγουμε τον καλύτερο εκ των ταξινομητών μας, αυτόν που σημείωσε υψηλότερο score στα test δεδομένα, τον rbf kernel SVM και χρησιμοποιώντας τον BaggingClassifier δημιουργούμε ένα ensemble. Το score που λαμβάνουμε για τα test δεδομένα ισούται με 0.9501, το υψηλότερο από το hard voting και από κάθε ταξινομητή ξεχωριστά. Τα decision trees είναι μέθοδος ταξινόμησης κατά την οποία επιδιώκεται η εξαγωγή κανόνων αποκλειστικά από τα στατιστικά χαρακτηριστικά δίχως expert knowledge, δίχως features. Ο συνδυασμός bagging και decision tree μας δίνει το random forest. Βλέπουμε ότι ο μόνο Decision Tree ταξινομητής βελτιώνεται (περίπου κατά 4 ποσοστιαίες μονάδες) με χρήση του bagging, γεγονός που οφείλεται στο ότι είναι αρκετά πιο ασταθής σε σχέση με τους υπόλοιπους (SVM, knn), με αποτέλεσμα στον Random Forest να είναι μειωμένο το Variance σε σχέση με τον Decision Tree.

Δοκιμάζουμε, λοιπόν, αρχικά με KNN για K=1 και 7 estimators και κάνουμε train σε κάθε estimator ξεχωριστά. Στη συνέχεια, κάνουμε το ensembling. Όπως προαναφέρθηκε, για το test φτάνουμε στο 95%, σκορ οριακά χειρότερο από αυτό του KNN με K=1, ενώ για το train προσεγγίζουμε το 100%. Για αυτόν τον ταξινομητή έχουμε λοιπόν:

Heat map:



Confusion Matrices:



Βήμα 19

A)

Χρησιμοποιώντας τον κώδικα από το tutorial της pytorch, φτιάχνουμε μία κλάση η οποία καλεί την Dataset και φορτώνει τα digits, ενώ από την torch.utils.data χρησιμοποιούμε την κλάση DataLoader, η οποία αναλάβει την ανάγνωση των δεδομένων και τον χωρισμό τους σε batches.

Ένα από τα προβλήματα που προκύπτουν στο learning είναι πώς θα διαχειριζόμαστε το μεγάλο όγκο δεδομένων. Συγκεκριμένα, μπορεί ο όγκος να είναι τέτοιος ώστε να μην χωράνε όλα ταυτόχρονα στη RAM.

Έτσι, χρησιμοποιούμε μεθόδους για να φορτώνουμε αποδοτικά τα δεδομένα κατά τη διάρκεια του training, αλλά και του testing, ώστε η διαδικασία του dataloading να μην αποτελέσει bottleneck για το συνολικό learning. Οι dataloaders φορτώνουν τα data σε batches, δηλαδή σε κομμάτια μη επικαλυπτόμενα του αρχικου dataset και μάλιστα αυτά μπορούν να δημιουργηθούν με παράλληλο τρόπο από το σύστημα (num workers).

Στο παρακάτω μέρος, θα χρησιμοποιήσουμε neural networks, στην εκπαίδευση των οποίων γίνεται τροφοδότηση του δικτύου με batches και όχι με το κάθε sample χωριστά. Για αυτό το λόγο, θα χρησιμοποιήσουμε Dataloaders για την τροφοδότηση των data. Επιπλέον, αξίζει να τονιστεί ότι ένας άλλος λόγος χρήσης του dataloader είναι η λειτουργία του shuffling, που στοχεύει στην ανάμειξη των αρχικών data και άρα στην ομογενοποίησή τους. Ο dataloader που χρησιμοποιούμε προέρχεται από την βιβλιοθήκη της Pytorch, ενώ το dataset που δημιουργούμε (και στο οποίο εφαρμόζεται ο dataloader) κληρονομεί απο το dataset της pytorch. Στα πλαίσια της εφαρμογής μας, ο όγκος των δεδομένων είναι μικρός και έτσι δεν χρειάζεται να φορτώνουμε από τον δίσκο, αλλά από έναν πίνακα στην RAM. Έτσι, το indexing, που υλοποιήσαμε, δεν είναι indexing σε αρχεία, αλλά σε καταχωρήσεις πίνακα.

B)

Το μοντέλο έχει ως εξής. Αρχικά ισοπεδώνουμε την εικόνα 16*16 σε ένα διάνυσμα 256 στοιχείων (ή 256 χαρακτηριστικά). Στη συνέχεια περνάμε τα 256 στοιχεία εισόδου από το πρώτο κρυφό στρώμα για να τα μετατρέψουμε σε 180 διαστάσεις, στη συνέχεια το δεύτερο κρυφό στρώμα τα μετατρέπει σε 100 διαστάσεις και το τρίτο σε 50. Τέλος, το στρώμα εξόδου το οποίο θα το μετατρέψει σε ένα διάνυσμα 10 διαστάσεων, δηλαδή τον αριθμό των κλάσεων των δεδομένων μας (0-9 ψηφία).

Ο μετασχηματισμός μεταξύ των στρωμάτων γίνεται με γραμμικά στρώματα ή πλήρως συνδεδεμένα/αφινικά στρώματα. Σε αυτά τα στρώματα κάθε στοιχείο σε ένα στρώμα συνδέεται με κάθε στοιχείο στο επόμενο.

Κάθε σύνδεση μεταξύ ενός νευρώνα σε ένα στρώμα και ενός νευρώνα στο επόμενο στρώμα έχει ένα βάρος που σχετίζεται με αυτήν. Η είσοδος σε έναν νευρώνα είναι το άθροισμα των σταθμισμένων τιμών όλων των νευρώνων του προηγούμενου στρώματος που συνδέονται με αυτόν, συν έναν σταθμισμένο όρο προκατάληψης, όπου η τιμή της προκατάληψης είναι πάντα 1. Στη συνέχεια, ο νευρώνας εφαρμόζει μια συνάρτηση ενεργοποίησης σε αυτό το σταθμισμένο άθροισμα. Αυτή η συνάρτηση ενεργοποίησης είναι μια μη γραμμική συνάρτηση που επιτρέπει στο νευρωνικό δίκτυο να μαθαίνει μη γραμμικές συναρτήσεις μεταξύ εισόδων και εξόδων.

Κατωτέρω ορίζουμε το MLP μας, το οποίο αποτελείται από τρία γραμμικά στρώματα. Παίρνουμε πρώτα τη δέσμη εικόνων εισόδου και τις ισοπεδώνουμε ώστε να μπορούν να περάσουν στα γραμμικά στρώματα. Στη συνέχεια τις περνάμε από το πρώτο γραμμικό επίπεδο, το input_fc, το οποίο υπολογίζει το σταθμισμένο άθροισμα των εισόδων και στη συνέχεια εφαρμόζουμε στοιχειωδώς τη συνάρτηση ενεργοποίησης ReLU (rectified linear unit). Αυτό το αποτέλεσμα περνάει στη συνέχεια από ένα άλλο γραμμικό στρώμα, το hidden_fc, εφαρμόζοντας και πάλι την ίδια συνάρτηση ενεργοποίησης στοιχειωδώς. Τέλος, το αποτέλεσμα περνάει από το τελευταίο γραμμικό επίπεδο, το output_fc. Επιστρέφουμε όχι μόνο την έξοδο αλλά και το δεύτερο κρυφό στρώμα καθώς θα κάνουμε κάποια ανάλυση σε αυτό αργότερα.

Δεν υπάρχει φόρμουλα που να μας λέει πόσα επίπεδα ή πόσους νευρώνες πρέπει να έχουμε σε κάθε επίπεδο. Ο γενικός κανόνας πάντως είναι ότι τα στρώματα που βρίσκονται κοντά στην είσοδο πρέπει να έχουν μεγαλύτερο αριθμό νευρώνων για να εξάγουν γενικά χαρακτηριστικά (π.χ. γραμμές, καμπύλες, ακμές), και ο αριθμός των νευρώνων μειώνεται σε κάθε επόμενο στρώμα, συνδυάζοντας τα χαρακτηριστικά που μαθαίνουν τα προηγούμενα στρώματα σε πιο υψηλού επιπέδου χαρακτηριστικά (π.χ. η

τομή δύο γραμμών κάνει ένα σταυρό, πολλαπλές καμπύλες κάνουν έναν κύκλο). Θέλουμε επομένως μια συμπίεση της πληροφορίας σε κάθε επίπεδο.

Στην συνέχεια, χρησιμοποιώντας τον κώδικα για την υλοποίηση του απλού νευρωνικού δικτύου και τροποποιώντας την είσοδο ώστε να δέχεται το σύνολο των pixels, λαμβάνει την flattened εικόνα, από δισδιάστατη δηλαδή 16x16 την δέχεται ως διάνυσμα 1x256.

Γ)

Έπειτα, εκπαιδεύουμε και αξιολογούμε το νευρωνικό μας δίκτυο με τρόπο που είναι γνωστός ξανά από το tutorial, δηλαδή έχουμε μια εξωτερική επανάληψη για κάθε εποχή και μία εσωτερική 9 επανάληψη για το dataset. Έχοντας κάνει όλα τα παραπάνω, μπορούμε να γράψουμε το νευρωνικό δίκτυο σε μορφή συμβατή με το scikit learn, οπου η συνάρτηση fit με παραμέτρους epochs, batch size και learning rate περιέχει την for loop η οποία εκπαιδεύει το νευρωνικό. Η συνάρτηση predict, αναλαμβάνει την επανάληψη στο test set και επιστρέφει το σύνολο των προβλέψεων. Τέλος, στην score, υπολογίζεται το accuracy, δηλαδή, το σύνολο των σωστών προβλέψεων προς τον αριθμό των δεδομένων του test set.

Δ)

Έχοντας υλοποιήσει το νευρωνικό σαν ενα scikit learner μπορούμε να εφαρμόσουμε cross validation και grid search. Δοκιμάζουμε 4 configurations και βλέπουμε τα εξής: Για μεγάλο learning rate (0.5) και λίγες εποχές (10) έχουμε όπως αναμενόταν κακά αποτελέσματα (cv score 40%, έναντι 97% των άλλων).

Παρατηρούμε ότι υψηλό lr και πολλές εποχές χαλάει το accuracy (λόγω overfitting και πολυπλοκότητας), ενώ χαμηλό lr και λίγες εποχές πάλι έχει χαμηλό accuracy γιατί δεν εκπαιδεύεται σωστά το νευρωνικό δίκτυο. Αυτό μπορεί να ισοσταθμιστεί βάζοντας σχετικά μεγάλο learning rate και λίγες εποχές ή μικρό lr και πολλές εποχές. Προφανώς lr κοντά στο 1 δεν έχει νόημα.

Τα καλύτερα αποτελέσματα (ακρίβεια 94%) πετυχαίνει ο συνδυασμός

{'layers_size': [256, 250, 180, 10], 'epochs': 30, 'lr': 0.05, 'n_layers': 3},

ο οποίος αντιστοιχεί στο δίκτυο με τους περισσότερους νευρώνες. Ωστόσο για να αντισταθμίσουμε την πολυπλοκότητα του δικτύου τις εποχές σε 30. Η τελευταία δοκιμή είχε 100 εποχές και μικρό lr αλλά επίσης λιγότερους νευρώνες από την βέλτιστη και πέτυχε ελαφρώς χειρότερο σκορ από τη βέλτιστη. Οι δύο αυτές αρχιτεκτονικές είχαν 3 επίπεδα. Η δεύτερη δοκιμή είχε μονο 2 επίπεδα και μικρότερο αριθμό νευρώνων από την τρίτη δοκιμή, η οποία πέτυχε το υψηλότερο σκορ. Επίσης έχει ίδιο lr με την τρίτη, αλλά μόνο 10 εποχές. Παρόλα τα αποτελέσματα είναι πολύ παρόμοια. Άρα το βέλτιστο μοντέλο είναι υπερβολικά πολύπλοκο και το πρόβλημα μπορεί να λυθεί με παρόμοια αποτελέσματα με απλούστερο μοντέλο ή/και λιγότερη εκπαίδευση (κορεσμός).