Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Αναγνώριση Προτύπων

$9^ο$ Εξάμηνο - Ροή Σ

Πρώτη Εργαστηριακή Αναφορά

Οπτική Αναγνώριση Ψηφίων

Κωνσταντίνος Κοψίνης - 031 17 062
kon.kopsinis@gmail.com

Δημήτρης Δήμος - 031 17 165
dimitris.dimos647@gmail.com

Αθήνα
Οκτώβριος, 2021

Περιεχόμενα

Περιγραφή Εργαστηρίου

Σε αυτή την εργασία υλοποιούμε ένα σύστημα που αναγνωρίζει ψηφία από εικόνες. Τα δεδομένα που χρησιμοποιούμε για τον σκοπό αυτό είναι γκρίζες εικόνες χειρόγραφων ψηφίων που έχουν σκαναριστεί και προέρχονται από την US Postal Service. Η παρούσα αναφορά συνοψίζει τις διαδικασίες που ακολουθήσαμε, αρχικά, στο προπαρασκευαστικό στάδιο του εργαστηρίου και στη συνέχεια στη κύριο μέρος του.

Η προπαρασκευή αφορά προεπεξεργασία των δεδομένων που θα χρησιμοποιηθούν ως είσοδος στην εκπαίδευση των μοντέλων ταξινόμησης, εξαγωγή ορισμένων στατιστικών χαρακτηριστικών τους και την κατασκευή ενός Ευκλείδιου ταξινομητή.

Το κύριο μέρος περιλαμβάνει, κατ'αρχάς, την κατασκευή ενός ταξινομητή Naive Bayes "με το χέρι". Στη συνέχεια, συγκρίνουμε ως προς την απόδοσή τους οριμένους ταξινομητές υλοποιημένους ήδη στη βιβλιοθήκη της scikit-learn. Συγκεκριμένα, οι ταξινομητές που μας ενδιαφέρουν είναι οι Naive Bayes, Nearest Neighbours και SVM (Support Vector Machine με διάφορους kernels). Έπειτα, επιτελούμε το ίδιο task ταξινόμησης συνδυάζοντας τους παραπάνω ταξινομητές με τεχνικές ensembling και boosting.

Στο τελευταίο στάδιο του εργαστηρίου, υλοποιούμε customly ένα νευρωνικό δίκτυο (πρακτικά πολλές εκδοχές μιας βασικής αρχιτεκτονικής fully-connected NN) προκειμένου να πραγματοποιήσουμε την ταξινόμηση των ψηφίων και να εξάγουμε αντίστοιχα συμπεράσματα περί απόδοσης.

Προπαρασκευαστικά Βήματα

Είσοδος Δεδομένων

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

Σχεδιασμός του ψηφίου στη θέση 131

Ένα δείγμα του training data set και, συγκεριμένα, στη θέση 131 (αρχίζοντας την αρίθμηση από το 0) είναι το παρακάτω:

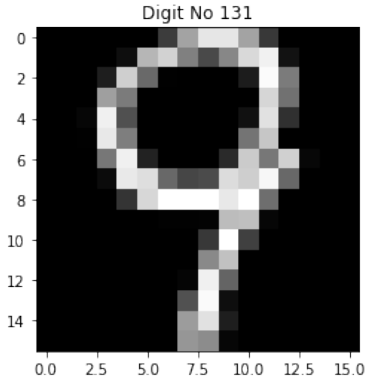

Figure 1: Ψηφίο υπ' αριθμόν 131

Ενδεικτική απεικόνιση ψηφίων από το train data

Για εποπτικούς λόγους, διαλέγουμε ένα sample από κάθε κλάση (0 έως 9) και το απεικονίζουμε:

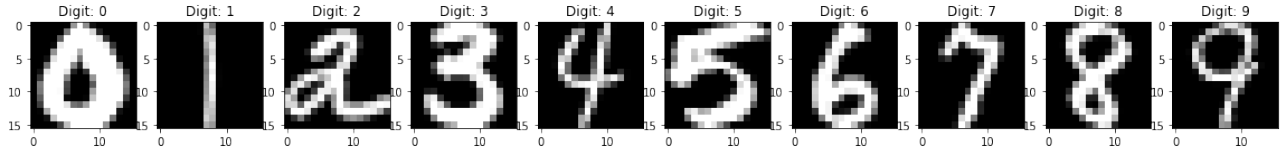

Figure 2: Ενδεικτικά samples από κάθε κλάση ψηφίων

Εξαγωγή Στατιστικών: Μέση Τιμή και Διασπορά

Στη συνέχεια, εξάγουμε τα επιθυμητά χαρακτηριστικά από το σύνολδο δεδομένων εκπαίδευσης, δηλαδή τη μέση τιμή και τη διασπορά. Συγκριμένα, αυτό που επιθυμούμε να πετύχουμε είναι ο υπολογισμός της μέσης τιμής και της διασποράς σε κάθε pixel απο κάθε κλάση.

Αρχίζουμε από υπολογισμούς σε επίπεδο pixel και υπολογίζουμε ενδεικτικά τη μέση τιμή και τη διασπορά του pixel (10,10) (αρχίζοντας την αρίθμηση από το (0,0)) για την κλάση του ψηφίου 0. Φάνηκε ότι:

mean value	-0.504
variance	0.525

Υπολογίζουμε τα ίδια χαρακτηριστικά, ομοίως, για κάθε pixel των εικόνων της κλάσης 0. Οπτικοποιούμε το αποτέλεσμα και παραθέτουμε τις παρακάτω εικόνες ως απόδειξη:

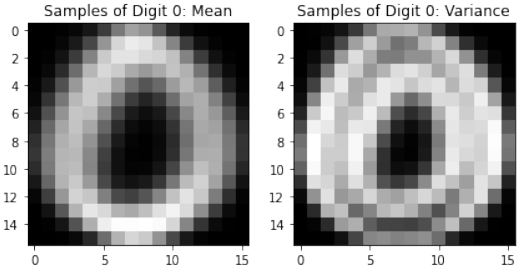

Figure 3: Mean Value and Variance

Η απεικόνιση των παραπάνω στατιστικών ιδιοτήτων έρχεται σε απόλυτη συμφωνία με το διαισθητικό τους νόημα.

Η μέση τιμή φαίνεται να έχει μεγάλες τιμές σε pixel τέτοια, που η εικόνα δίνει την εντύπωση πως όντως πρόκειται για το ψηφίο 0. Στις άκρες της εικόνας οι τιμές είναι 0, γεγονός λογικό αφού κάνενα ψηφίο της κλάσης 0 δεν έχει μη μηδενικό pixel στα περιθώρια.

Η διασπορά, επίσης, συμφωνεί με την προγενέστερη αντίληψή μας. Εν γένει, το εν λόγω μέγεθος λαμβάνει τιμές μεγάλες σε pixel που παρουσιάζουν αποκλίσεις, δηλαδή σε σημεία που διαφέρουν ανά τις εικόνες. Σε σημεία όπου υπάρχει απόλυτη συμφωνία (δηλαδή τα pixels δεν έχουν άλλες τιμές, αλλά συμφωνούν σε μία) η διασπορά είναι 0. Αυτό γίνεται αντιληπτό παρατηρώντας τα περίχωρα της εικόνας της διασποράς, όπου οι τιμές των pixel ειναι 0. Πράγματι, αυτά τα pixel ειναι 0 για κάθε εικόνα της κλάσης 0. Οι τιμές της διαποράς είναι ιδιαίτερα χαμηλές στα pixel που έχει υψηλές τιμές η μέση τιμή. Αυτό είναι λογικό, καθώς τα μέγιστα pixel της μέσης τιμής συμφωνούν ως επί το πλείστον. Τελικώς, παρατηρώντας το "0" που σχηματίζουν τα pixel της μέσης τιμής και τις αντίστοιχες τιμές της διασποράς, διαπιστώνουμε ότι εκεί η διασπορά παρουσιάζει τις μέγιστές τιμές της. Πράγματι, πρόκειται για τα pixel που διαφέρουν περισσότερο από μηδενικό σε μηδενικό του train dataset, άρα θα έχουν και μεγάλο βαθμό συμφωνίας. Άρα, σωστά η διασπορά παίρνει μεγάλες τιμές σε αυτά τα pixel.

Στη συνέχεια, παραθέτουμε τις εικόνες που σχηματίζονται υπολογίζοντας τις μέσες τιμές για κάθε κλάση ψηφίων:

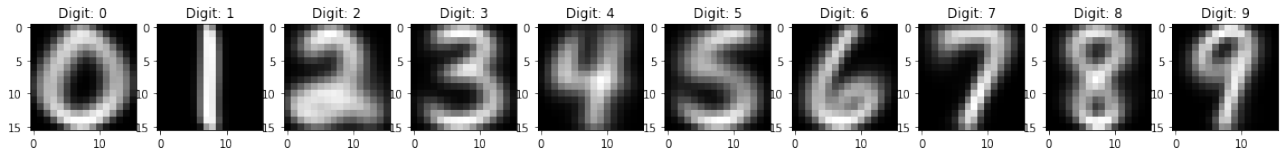

Figure 4: Μέση τιμή για κάθε κλάση ψηφίων

Από την παραπάνω απεικόνιση συμπεραίνουμε πως η μέση τιμή κάθε κλάσης μοίαζει με μια θολωμένη εκδοχή του αντίστοιχου ψηφίου.

Ταξινόμηση του ψηφίου της θέσης 101 του test set

Στη συνέχεια, επιδιώκουμε την ταξινόμηση του ψηφίου στη θέση 101 του test set. Για αυτή την ταξινόμηση χρησιμοποιούμε την ευκλείδια απόσταση ως κριτήριο. Υπολογίζουμε την εν λόγω απόσταση της εικόνας προς ταξινόμηση από τη μέση τιμή κάθε κλάσης και το ταξινομούμε στην κλάση με τη μικρότερη απόσταση.

Για τον υπολογισμό της απόστασης, θεωρούμε τις εικόνες ως flattened images, δηλαδή διανύσματα, και υπολογίζουμε την τιμή:

$$d[c] = \sqrt{\sum_{0}^{255} \left(x_{j(img)} - x_{j(c)} \right)^2}, \quad \forall c \in \{0, \ldots, 9\}$$

Το δείγμα ταξινομείται στην κλάση:

$$c = \arg\min_{c} d[c]$$

Στην περίπτωσή μας, το ψηφίο 101 - για το οποίο γνωρίζουμε πως ανήκει στην κλάση 6 - ταξινομείται στην κλάση 0. Το αποτέλεσμα είναι λανθασμένο, αλλά εν μέρει "λογικό" αν θεωρήσουμε πως τα ψηφία 0 και 6 έχουν αρκετά μεγάλο κομμάτι τους ίδιο. Δεν πρόκεινται για ψηφία με μεγάλη διαφορά μεταξύ τους, όπως για παράδειγμα θα ήταν το 1 με το 8.

Ταξινόμηση όλων των test data βάσει ευκλείδιας απόστασης

Επαναλαμβάνουμε την ίδια διαδικασία για όλα τα ψηφία του test set. Ορίζοντας σας ποσοστό επιτυχίας τις σωστές ταξινομήσεις προς το πλήθος των test data, λαμβάνουμε ποσοστό επιτυχίας ίσο με 81.42 %.

Δεδομένης της ευκολίας υλοποίησης του συγκεκριμένου ταξινομητή, θα λέγαμε πως ένα ποσοστό που ξεπερνά το 80% είναι σχετικά ικανοποιητικό.

Στο τελευταίο στάδιο της προπαρασκευής, υλοποιούμε τον Ευκλείδιο ταξινομητή στο format ενός estimator της βιβλιοθήκης scikit-learn, με σκοπό να εξάγουμε μερικά ακόμη αποτελέσματα.

5-Fold Cross Validation

Για την περαιτέρω αξιολόγηση του ταξινομητή, χρησιμοποιούμε 5-fold cross validation. Η εν λόγω μέθοδος, αγνοεί το σύνολο δεδομένων που έχουμε εμείς ορίσει ως test set και χρησιμοποιεί μόνο το train set για να αξιολογήσει.

Αρχικά, διαιρεί το train set σε k υποσύνολα. Ύστερα, θέτει το πρώτο σύνολο ως validation set και ενοποιεί τα υπόλοιπα k-1 ως train set. Εκπαιδεύει το μοντέλο του ταξινομητή χρησιμοποιώντας το train set και αξιολογεί τον ταξινομητή στο validation set. Επαναλαμβάνει την παραπάνω διαδικασία, χρησιμοποιώντας κάθε φορά διαφορετικό υποσύνολο του αρχικού train set ως validation. Αποθηκεύοντας κάθε φορά τα evaluation scores, στο τέλος υπολογίζει το μέσο ποσοστό επιτυχίας του ταξινομητή, υπολογίζοντας τον αριθμητικό μέσο όρο των επιμέρους scores.

Για τον δικό μας ταξινομητή, η παραπάνω μέθοδος είχε σαν αποτέλεσμα:

$$\text{cross-validation error} : 0.151420 \pm 0.001816$$

το οποίο ισοδυναμεί κατά προσέγγιση με ένα μέσο ποσοστό επιτυχίας: 84.86 %. Η 5-fold cross validation αξιολόγηση φανερώνει υψηλότερα ποσοστά επιτυχίας σε σχέση με την προηγούμενη αξιολόγηση πάνω στα test data.

Περιοχή Απόφασης και Καμπύλη Εκμάθησης

Για να πετύχουμε μια προσεγγιστική οπτικοποίηση της ταξινόμησης σε ένα επίπεδο, θα θάλαμε να σχεδιάσουμε τις περιοχές απόφασης του classifier. Φυσικά, κάτι τέτοιο δεν είναι εφικτό με δεδομένο ότι οι διαστάσεις των δεδομένων προς ταξινόμηση είναι 256. Είναι προφανές πως δεν δύναται να απεικονιστούν στον φυσικό κόσμο περιοχές που καθορίζονται από περισσότερα των 3 σημείων.

Για τους σκοπούς της άσκηση, λοιπόν, μειώνουμε τις διαστάσεις των δεδομένων μας από 256 σε 2. Προκειμένου να το πετύχουμε, χρησιμοποιούμε ανάλυση των διανυσμάτων εισόδου (που είναι εικόνες 256 pixel) σε Πρωταρχικές Συνιστώσες, ή όπως είναι αλλιώς γνωστή: Principal Component Analysis (PCA). Εφαρμόζοντας PCA στα διανύσματα εκπαίδευσης και καθιστώντας τον ταξινομήτη ικανό να ταξινομήσει δεδομένα 2 διαστάσεων (αυτό απαιτεί μικρές τροποποιήσεις στον κώδικα), παράγουμε την παρακάτω απεικόνιση περιχής απόφασης:

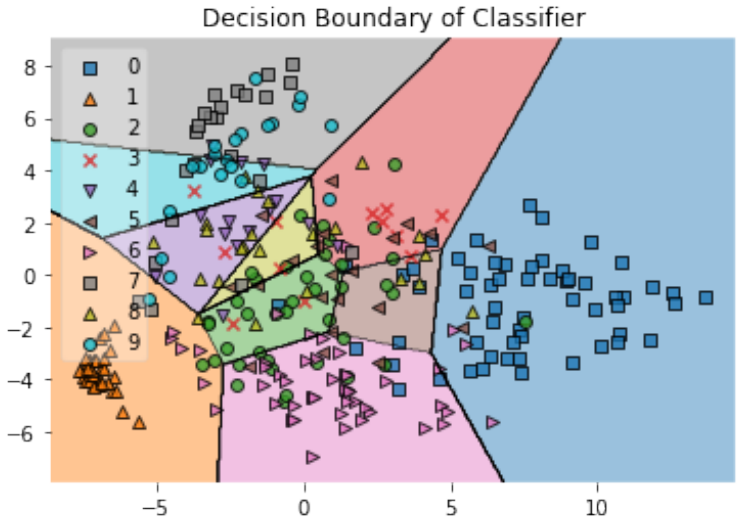

Figure 5: Περιοχή απόφασης ταξινομητή, μετά την μείωση της διαστατικότητας. Απεικονίζουμε μόνο 300 σημεία, για να φαίνονται καθαρά οι περιοχές απόφασης και σε σημεία υψηλής συγκέντρωσης

Παρατηρούμε πως η μεγάλη πλειονότητα των δειγμάτων έχει ταξινομηθεί σωστά. Όλα τα δείγματα της κλάσης 1, για παράδειγμα, έχουν ταξινομηθεί σωστά και "ασφαλώς" εντός των σωστών ορίων.

Παραθέτουμε και την καμπύλη εκμάθησης του ταξινομητη πάνω στο σύνολο X_train, για 5 διαφορετικά υποσύνολά του:

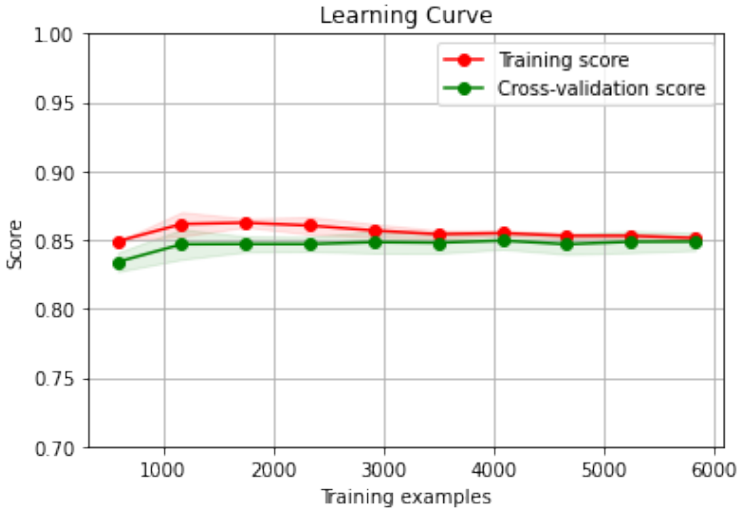

Figure 6: Καμπύλη εκμάθησης

Από το γράφημα παρατηρούμε ότι η αύξηση των δειγμάτων που χρησιμοποιούνται για την εκπαίδευση συνεπάγεται σταδιακή σύγκλιση των training score και cross-validation scores. Φαίνεται πως το ποσοστό του 85% πετυχαίνεται στα 6000 δείγματα περίπου, ενώ το score φαίνεται να έχει την τάση να σταθεροποιηθεί σε αυτό το επίπεδο.

Βήματα Κυρίου Μέρους

Naive Bayes Classifier

Σε πρώτο στάδιο του κυρίου μέρους κατασκευάζουμε από το μηδέν έναν ταξινομητή Naive Bayes. Αρχικά, προσδιορίζουμε τις a priori πιθανότητες της κάθε κλάσης, υπολογίζοντας τις εξής ποσότητες:

$$\mathcal{P}(y_i) = \frac{\#\text{ samples} \in}{\#\text{ samples}}, \forall i \in \{0, \ldots, 9\}$$

Τα αποτελέσματα είναι:

$\mathcal{P}(y_0)$	=	0.1638
$\mathcal{P}(y_1)$	=	0.1378
$\mathcal{P}(y_2)$	=	0.1003
$\mathcal{P}(y_3)$	=	0.0902
$\mathcal{P}(y_4)$	=	0.0894
$\mathcal{P}(y_5)$	=	0.0763
$\mathcal{P}(y_6)$	=	0.0911
$\mathcal{P}(y_7)$	=	0.0885
$\mathcal{P}(y_8)$	=	0.0743
$\mathcal{P}(y_9)$	=	0.0883

Με κόκκινο σημειώνεται η μεγαλύτερη πιθανότητα, που αντιστοιχεί το ψηφίο 0.

Για την υλοποίηση του Naive Bayes Classifier βασιζόμαστε στη θεωρία. Ο ταξινομητής Naive Bayes ταξινομεί κάθε δεδομένο βάσει της μέγιστης a posteriori πιθανότητας (Maximum a-posteriori Probability - MAP). Συγκεκριμένα, υπολογίζει τις a posteriori πιθανότητες και ταξινομεί στην κλάση για την οποία η εν λόγω πιθανότητα είναι η μεγαλύτερη. Βάσει του κανόνα Bayes, αυτή η πιθανότητα είναι:

$$\mathcal{P}(y|\vec{x}) = \frac{\mathcal{P}(y) \cdot \mathcal{P}(\vec{x}|y)}{\mathcal{P}(\vec{x})}$$

Στο υπολογισμό των παραπάνω ποσοτήτων, ο παρονομαστής δεν επιδρά, καθώς είναι ίδιος για κάθε y. Επομένως, η επιλογή απλοποιείται στην παρακάτω ποσότητα:

$$y = \arg\max_y \mathcal{P}(y) \cdot \mathcal{P}(\vec{x}|y)$$

Ο υπολογισμός του $\mathcal{P}(\vec{x}|y)$ καθορίζεται από τη εξάρτηση των μεταβλητών $x_1, x_2, \ldots, x_n$ του διανύσματος $\vec{x}$. Στην περίπτωση του Naive Bayes, γίνεται η υπόθεση ότι τα $x_1, x_2, \ldots, x_n$ (δηλαδή τα pixel της εκάστοτε εικόνας) είναι ανεξάρτητα, εξού και ο χαρακτηρισμός του ως Naive. Έτσι, θεωρούμε ότι κάθε πιθανότητα pixel δεδομένης της κλάσης y, περιγράφεται από μια κατανομή, την οποία και θέτουμε εμείς κάθε φορά ανάλογα με τον τύπο μεταβλητών, ώστε να μας εξυπηρετεί. Εφόσον έχουμε αριθμητικά δεδομένα, μια καλή επιλογή είναι να θεωρήσουμε πως η κατανομή πιθανότητας κάθε pixel είναι μια Κανονική Κατανομή με μέση τιμή και διασπορά που καθορίζονται από το συγκεκριμένο pixel και μόνο.

Άρα, όλα τα features είναι ανεξάρτητες μεταβλητές, κάθε μία από τις οποίες ακολουθεί κανονική κατανομή (independent & identically distributed). Αυτό συνεπάγεται τις ακόλουθες απλοποιήσεις:

$$\mathcal{P}(y|\vec{x}) = \frac{\mathcal{P}(y) \cdot \mathcal{P}(x_1|y) \cdot \mathcal{P}(x_2|y) \ldots \mathcal{P}(x_n|y)}{\mathcal{P}(\vec{x})}$$

και, συνεπώς:

$$y = \arg\max_y \mathcal{P}(y) \cdot \mathcal{P}(x_1|y) \cdot \mathcal{P}(x_2|y) \dots \mathcal{P}(x_n|y)$$

όπου έχουμε θεωρήσει ότι η κατανομές είναι κανονικές, άρα:

$$\mathcal{P}(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \cdot exp\Big\{ -\frac{(x_i - \mu_i)^2}{2\sigma_i^2} \Big\} \sim \mathcal{N}(\mu_i, \sigma_i^2)$$

Για να μην προκύψουν προβλήματα στους υπολογισμούς της διασποράς, με δεδομένο ότι σε πολλά pixel η διασπορά είναι μηδεν (αφού έχουν την ίδια τιμή όλα τους), προσθέτουμε μια μικρή σταθερά σε όλες τις διασπορές της τάξης του 10^{-9}. Την ίδια τεχνική φαίνεται να ακολουθεί και η έτοιμη υλοποίηση της scikit-learn.

Τα αποτελέσματα συγκρίνοντας τους δύο ταξινομητές (custom και έτοιμη υλοποίηση της scikit learn) είναι:

custom	71.9%
scikit-learn	71.95%

Το αποτέλεσμα του custom ταξινομητή εξαρτάται άμεσα από την επιλογή της smoothing παραμέτρου. Παρατηρούμε ότι αλλάζοντας την τάξη μεγέθους της (η πιο "σωστή" τάξη θα λέγαμε είναι κοντά σε 10^{-9}), αλλάζει και το τελικό score.

Naive Bayes με unit variance

Στη συνέχεια επαναταξινομούμε τα δεδομένα test, θεωρώντας ότι οι διακυμάνσεις κάθε pixel είναι ίσες με 1. Αυτή η κίνηση παρουσιάζει το εξής ενδιαφέρον: ο πίνακας συνδυακύμανσης για τα χαρακτηριστικά κάθε εικόνας γίνεται ο μοναδιαίος και, ενθυμούμενοι ότι η κατανομή που έχουμε θέσει τα pixel να ακολουθούν είναι κανονική, αυτό συνεπάγεται ότι ο Naive Bayes Classifier μετατρέπεται σε Ταξινομητής Ευκλείδιας απόστασης και τα decision surfaces είναι υπερεπίπεδα 1ης τάξης [1].

Μάλιστα το score του ταξινομητή με unit variance είναι πολύ κοντά σε αυτό του αρχικού μας Euclidean Classifier και ισούται με 81.27%.

Σύγκριση Ταξινομητών (Euclidean, Naive Bayes, SVM, KNN)

Σε αυτό το βήμα συγκρίνουμε την επίδοση ορισμένων ταξινομητών στα train δεδομένα με χρήση cross-validation:

Euclidean Distance	84.86%
Custom Naive Bayes	74.89%
Custom Naive Bayes (unit variance)	84.83%
sklearn Gaussian NB	74.83%
SVM (linear kernel)	95.25%
SVM (rbf kernel)	97.63%
KNN (3 neighbors)	96.39%
KNN (5 neighbors)	95.99%

Με κόκκινο χρώμα επισημαίνουμε το μέγιστο ποσοστό επιτυχίας. Αυτό το πετυχαίνει ο SVM (Support Vector Machine) ταξινομητής με πυρήνα rbf. Τα καλύτερα αποτελέσματα φαίνεται να πετυχαίνουν οι ταξινομητές SVM και KNN (K Nearest Neighbors).

Ensembling/Bagging/Boosting

Σε αυτό το βήμα, επιδιώκουμε περαιτέρω βελτίωση της απόδοσης της ταξινόμησης συνδυάζοντας επιμέρους ταξινομητές. Για αυτή την τεχνική, η οποία ονομάζεται ensembling, συνδυάζουμε ταξινομητές για τους οποίους γνωρίζουμε ότι τείνουν να ταξινομούν λανθασμένα συγκεκριμένα ψηφία, ώστε να πετυχουμε καλύτερο αποτέλεσμα. Για αυτόν τον σκοπό, αρχικά, παράγουμε πίνακες που υποδεικνύουν τη συχνότητα που κάθε ταξινομητής κάνει λάθος στην πρόβλεψη κάθε ψηφίου:

Classifier Type	0	1	2	3	4	5	6	7	8	9
Euclidean Distance	195	4	145	80	132	116	95	105	88	124
Custom Naive Bayes	180	7	148	275	465	322	49	49	182	78
Custom Naive Bayes (unit var)	190	4	144	80	133	119	93	105	90	125
sklearn Gaussian NB	183	8	148	274	465	323	49	48	184	77
SVM (linear)	0	0	0	0	1	0	0	0	0	0
SVM(rbf)	0	0	7	4	4	6	7	6	8	4
KNN (3 neighbors)	3	0	15	9	16	13	8	3	20	10
KNN (5 neighbors)	6	1	20	11	28	23	9	11	31	12

Ταξινομώντας τα παραπάνω δεδομένα σε φθίνουσα σειρά παίρνουμε τον παρακάτω πίνακα:

Classifier Type	0	1	2	3	4	5	6	7	8	9
Euclidean Distance	0	2	4	9	5	7	6	8	3	1
Custom Naive Bayes	4	5	3	8	0	2	9	7	6	1
Custom Naive Bayes (unit var)	0	2	4	9	5	7	6	8	3	1
sklearn Gaussian NB	4	5	3	8	0	2	9	6	7	1
SVM (linear)	4	9	8	7	6	5	3	2	1	0
SVM(rbf)	8	6	2	7	5	9	4	3	1	0
KNN (3 neighbors)	8	4	2	5	9	3	6	7	0	1
KNN (5 neighbors)	8	4	5	2	9	7	3	6	0	1

Voting Classifier

Η πρώτη τεχνική χρησιμοποιεί έναν meta-classifier ο οποίος συνδυάζει τις επιδόσεις των επιμέρους ταξινομητών, επιλέγοντας κάθε φορά κλάση ταξινόμησης με χρήση ψηφοφορίας. Τα ήδη ψηφοφορίας είναι δύο. Συγκεκριμένα:

- Hard Voting: η κλάση επιλέγεται ως η κλάση με τις περισσότερες ψήφους από τους επιμέρους ταξινομητές. Για να αποφευχθούν ισοπαλίες, χρησιμοποιούμε μονό αριθμό ταξινομητών.
- Soft Voting: η κλάση επιλέγεται ως η κλάση με το μεγαλύτερο άθροισμα πιθανοτήτων, δηλαδή για κάθε κλάση παράγουμε μια πιθανότητα ως άθροισμα πιθανοτήτων από κάθε ταξινομητή και επιλέγουμε αυτή με τη μεγαλύτερη.

Σε πρώτη φάση συνδυάζουμε τους ταξινομητές SVM: linear kernel και KKN: 3 neighbors, παρατηρώντας ότι ο πρώτος κάνει missclassify μόνο μία φορά το ψηφίο 4 και ο δεύτερος έχει το δεύτερο καλύτερο ποσοστό ευστοχίας με cross validation. Με soft voting πετυχαίνεται ποσοστό ευστοχίας 97.22%.

Καταφέρνουμε, πράγματι, μια μικρή βελτίωση του ποσοστού ευστοχίας, ανεβάζοντάς το από το 95.25% του απλού linear SVM σε 97.22%.

Στη συνέχεια, συνδυάζουμε τους SVM(linear), KNN(3 neighbors) και SVM(rbf) που αποτελούν την καλύτερη τριάδα εκτιμητών. Με αυτόν τον συνδυασμό πετυχαίνουμε:

- με Hard Voting: 97.52%
- με Soft Voting: 97.63%

Παρατηρούμε ότι τα ποσοστά ευστοχία είναι αρκετά μεγάλα και πολύ κοντά στο ποσοστό του βέλτιστου ταξινομητή. Αυτό είναι λογικό αν σκεφτεί κανείς ότι ο βέλτιστος ταξινομητής πετυχαίνει το ποσοστό του κάνοντας μόνο ένα λάθος.

Bagging Classifier

Στη συνέχεια χρησιμοποιούμε Bagging Classifier, η λειτουργία του οποίου περιγράφεται ως εξής: χωρίζουμε το training set σε πιθανώς επικαλυπτόμενα υποσύνολα καιχρησιμοποιούμε καθένα από αυτά για να εκπαιδεύσουμε έναν ταξινομητή βάσης. Στην περίπτωσή μας χρησιμοποιούμε τον ταξινομητή με το βέλτιστο ποσοστό ευστοχίας, δηλαδή τον SVM (rbf kernel). Η ταξινόμηση πραγματοποιείται μέσω ψηφοφορίας ή μέσου όρου των προβλέψεων των επιμέρους όμοιων ταξινομητών. Με cross-validation έχουμε:

5 estimators	97.59%
10 estimators	97.65%
15 estimators	97.63%

Bonus Βήμα

Σε αυτό το μέρος προσπαθούμε να ταξινομήσουμε το γνωστό dataset εκπαιδεύοντας νευρωνικά δίκτυα ποικίλων αρχιτεκτονικών και παραμέτρων με τη βοήθεια του PyTorch.

Dataloader

Αυτό το βήμα είναι προπαρασκευαστικό και ασχολείται με την μορφοποίηση των δεδομένων πριν την εισαγωγή τους στο Νευρωνικό. Συγκεκριμένα, υλοποιούμε ένα στιγμιότυπο της κλάσης Dataset (την κλάση DigitData) προκειμένου να ομαδοποιήσουμε τα ζεύγη δείγματος-τιμής και να τα μετατρέψουμε από numpy-arrays σε torch-tensors υποβάλλοντάς τα στον αντίστοιχο μετασχηματισμό (βλ. κλάση ToTensor). Έτσι, μπορούμε να τα περάσουμε σε έναν Dataloader για να τα χωρίσει στον επιθυμητό αριθμό τεμαχίων (batches) με τυχαία σειρά.

Σχεδιασμός αρχιτεκτονικής νευρωνικού

Εδώ προπαρασκευάζουμε τον σκελετό του νευρωνικού μας. Υλοποιούμε την κλάση LinearWActivation προκειμένου να επιλέγουμε εύκολα την επιθυμητή συνάρτηση ενεργοποίησης (activation function) των γραμμικών στρωμάτων του νευρωνικού (σιγμοειδής ή βηματική). Στη συνέχεια υλοποιούμε την κλάση MyFunnyNet που κατασκευάζει το νευρωνικό χρησιμοποιώντας αλεπάλληλα γραμμικά επίπεδα του επιθυμητού πλήθους νευρώνων και υπολογίζει την έξοδό του για ορισμένη είσοδο. Τέλος, δοκιμάζουμε να φτιάξουμε ένα νευρωνικό δύο επιπέδων και 100 νευρώνων και επαληθεύουμε την αρχιτεκτονική του τυπώνοντάς την.

Εκπαίδευση και δοκιμές

Έχοντας πλέον δεδομένα και νευρωνικό, μπορούμε να ξεκινήσουμε την εκπαίδευση. Υλοποιούμε τη συνάρτηση train για την εφαρμογή του αλγορίθμου μάθησης και την evaluate για την αξιολόγηση της απόδοσης του νευρωνικού. Μέσα από τη συνάρτηση train, μάλιστα, μπορούμε να περάσουμε και τις επιθυμητές παραμέτρους όπως το κριτήριο κόστους, τον ρυθμό μάθησης και τη συνάρτηση βελτιστοποίησης. Επαληθεύουμε τα παραπάνω εκπαιδεύοντας πιλοτικά ένα νευρωνικό με 2 επίπεδα των 100 νευρώνων σε 100 εποχές. Η επίδοση υπολογίστηκε: 50.92%.

Testing

Σύμφωνα με την εκφώνηση, πειραματιστήκαμε μελετώντας τη συμπεριφορά του νευρωνικού στο τρέχον σύνολο δεδομένων σχετικά με τον αριθμό των νευρώνων, τον αριθμό των layers και τον τύπο των μη γραμμικών activation. Για εξοικονόμηση χρόνου, επιλέξαμε η εκπαίδευση να γίνει μόνο σε 50 εποχές ώστε να περιοριστεί ο χρόνος της εκτέλεσης του προγράμματος. Αυτό σημαίνει ότι η πιθανότατα η επίδοση του συστήματος να μειωθεί. Λάβαμε τα παρακάτω αποτελέσματα:

Μελέτη ως προς τον αριθμό των νευρώνων

Network Layers	Sigmoid	ReLU
25 x 25	17.89%	92.33%
50 x 50	30.64%	92.97%
100 x 100	32.29%	92.87%
200 x 200	33.93%	93.02%

Παρατηρούμε ότι:

- Η ReLU οδηγεί σε δραστικά μεγαλύτερη επίδοση το νευρωνικό μας και συνεπώς για τη συγκεκριμένη εφαρμογή κρίνεται η πιο κατάλληλη.
- Η αύξηση των νευρώνων, προκαλεί και στις δύο περιπτώσεις αύξηση και στην απόδοση. Αυτό συμβαδίζει και με τη θεωρία, αφού περισσότεροι νευρώνες εισάγουν περισσότερες παραμέτρους στο σύστημα (συγκεκριμένα τα βάρη τους) και συνεπώς μπορεί να σχεδιάσει αποδοτικότερες τις περιοχές απόφασης.

- Η αύξηση των νευρώνων προκαλεί μεγαλύτερη διαφορά στην Sigmoid απ' ό,τι στη ReLU. Πρακτικά, η ReLU είναι αποδοτική ήδη με το 25x25 νευρωνικό και συνεπώς δεν χρειάζεται να τη χρησιμοποιήσουμε με περισσότερους νευρώνες, μιας και η βελτίωση είναι δυσανάλογη της πολυπλοκότητας που εισάγουμε.

Περαιτέρω έρευνα:

- **Sigmoid:** Όπως φαίνεται η 'Sigmoid' επηρεάζει αρνητικά την επίδοση του συγκεκριμένου συστήματος. Σε περίπτωση που επιμείνουμε στην χρήση της, θα πρέπει να δοκιμάσουμε να αυξήσουμε είτε τα στρώματα του δικτύου, είτε τις εποχές εκπαίδευσης. Το δεύτερο, ειδικά, μοιάζει επιτακτικό προκειμένου να δωθεί περισσότερος χρόνος στο νευρωνικό μας για να προσαρμοστεί στα δεδομένα, μιας και η 'Sigmoid' είναι ομαλή (smooth) και όχι απότομη.
- **ReLU:** Η ReLU είναι τόσο καλή στη δουλειά της που θα μπορούσαμε να δοκιμάσουμε έως και να απλουστεύσουμε την αρχιτεκτονική του νευρωνικού μας. Θα είχε ενδιαφέρον να μειώσουμε κι άλλο τον αριθμό των νευρώνων, τον αριθμό των layers ή και το πλήθος των εποχών εκπαίδευσης και να δούμε μέχρις ποιου σημείου εξακολουθούμε να παίρνουμε τις ίδιες μετρικές. Όσον αφορά την βελτίωση της επίδοσης και δεδομένου ότι οι 50 εποχές δεν είναι και ιδιαίτερα πολλές, το κλειδί βρίσκεται ασφαλώς στην αύξησή τους.

Μελέτη ως προς τον αριθμό των layers

Network Layers	Sigmoid	ReLU
100	78.82%	92.13%
100 x 100	30.59%	92.63%
100 x 100 x 100	17.89%	93.02%
100 x 100 x 100 x 100	17.89%	91.58%

Παρατηρούμε ότι:

- Η ReLU εξακολουθεί να οδηγεί σε δραστικά μεγαλύτερη επίδοση το νευρωνικό μας και παραμένει η ιδανική επιλογή μας.
- Για πρώτη φορά επιτύχαμε υψηλή επίδοση με την χρήση της Sigmoid και αυτή ήταν για 1 layer. Αυτό αποτελεί σοβαρή ένδειξη για περαιτέρω έρευνα μιας και η επιτυχία της συνδέεται άμεσα με τον αριθμό των layers. Ταυτόχρονα, βλέπουμε την επίδοση να μειώνεται δραματικά με την αύξηση των layers κάτι που αποτελεί ισχυρή ένδειξη για ένα πιθανό overfitting στα δεδομένα μας.
- Η περίπτωση της ReLU είναι εξίσου ενδιαφέρουσα διότι τη βλέπουμε εξίσου αποδοτική σε όλη την γκάμα των νευρωνικών όπου την αξιοποιήσαμε. Οι μικρές διαφορές που βλέπουμε είναι αμελητέες και πιθανότατα να έχουν να κάνουν με την τυχαία αρχικοποίηση των βαρών στην αρχή της εκπαίδευσης.

Περαιτέρω έρευνα:

- **Sigmoid:** Είναι πλέον προφανές ότι η Sigmoid αποδίδει σε νευρωνικά με τον ελάχιστο δυνατό αριθμό από ενδιάμεσα layers, δηλαδή ένα. Ακόμα κι έτσι, όμως, θα θέλαμε ιδανικά να αυξήσουμε κι άλλο το 78.82% που επιτύχαμε στο δεύτερο γύρο μετρήσεων. Για να γίνει αυτό, πρέπει να εκμεταλλευτούμε αφενός την θετική επίδραση της αύξησης των νευρώνων που ανέδειξε ο πρώτος γύρος μετρήσεων, καθώς και την αξιοποίηση μεγαλύτερου πλήθους εποχών εκπαίδευσης προκειμένου να δώσουμε χρόνο στο σφάλμα να μειωθεί κι άλλο.
- **ReLU:** Ο δεύτερος γύρος μετρήσεων δεν μας έδωσε κάποια περαιτέρω πληροφορία για την ReLU. Η αύξηση του πλήθους των layers δεν βελτίωσε την επίδοση του συστήματος και συνεπώς μπορούμε να περιοριστούμε σε ένα μόνο layer για μια απλούστερη αρχιτεκτονική. Παραμένουν τα ερωτήματα της μείωσης του αριθμού των νευρώνων και της αύξησης των κύκλων εκπαίδευσης που διατυπώσαμε και προηγουμένως.

Custom Neural Network Classifier

Σε αυτό το βήμα υλοποιούμε τον κώδικα εκπαίδευσης και αξιολόγησης του νευρωνικού μέσα από έναν ταξινομητή, τον CustomNNClassifier, συμβατό με την scikit-learn. Έτσι, γίνεται φανερό ότι τα νευρωνικά δίκτυα δεν διαφέρουν και πολύ στη γενική αρχή τους από τις παραδοσιακές αλγεβρικές μεθόδους ταξινόμησης. Τέλος, τρέξαμε εκ νέου τα ίδια πειράματα με πριν χωρίς να λάβουμε κάποιο αισθητά διαφορετικό αποτέλεσμα.

References

[1] C. M. Bishop, *"Pattern Recognition and Machine Learning". Springer*, 2006.