ΗΜΜΥ ΕΜΠ

Αναγνώριση Προτύπων (2022-2023)

1η Εργαστηριακής Άσκησης

Ηλιόπουλος Γεώργιος:	03118815	giliopoulos301@gmail.com
Σερλής Εμμανουήλ Αναστάσιος:	03118125	manosserlis@gmail.com

Θέμα: Οπτική Αναγνώριση Ψηφίων

Σκοπός 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.

Βήμα 1: Εισαγωγή των δεδομένων

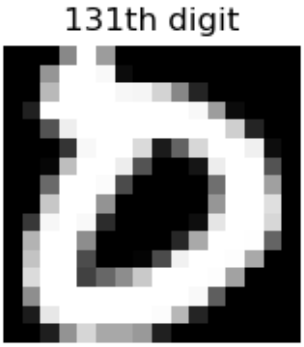
Στο βήμα έγινε η μεταβίβαση των δεδομένων από `txt` αρχεία σε πίνακες. Αρχικά τα δεδομένα περάστηκαν ως ένα data frame κάνοντας χρήση της βιβλιοθήκης pandas και στη συνέχεια μετατράπηκαν σε `NumPy arrays` διάστασης $N_{samples} \times 256$, όπου 256 είναι ο αριθμός των χαρακτηριστικών κάθε ψηφίου που ταυτίζεται με τον αριθμό των pixel (εικόνες 16×16). Από τους πίνακες απομονώθηκε η πρώτη στήλη που περιείχε το label κάθε δείγματος σε `ένα NumPy array y` διάστασης $N_{samples} \times 1$, και τα υπόλοιπα στοιχεία σε ένα `NumPy array` διάστασης $N_{samples} \times 256$, όπου 256 είναι ο αριθμός των χαρακτηριστικών κάθε ψηφίου που ταυτίζεται με τον αριθμό των pixel (εικόνες 16×16).

Έτσι μετά το πρώτο βήμα έχουμε 4 NumPy arrays:

Όνομα πίνακα	Περιγραφή	Διάσταση
`X_train`	Features των δεδομένων εκπαίδευσης	7291×256
`y_train`	Labels των δεδομένων εκπαίδευσης	7291×1
`X_test`	Features των δεδομένων εκπαίδευσης	2007×256
`y_test`	Labels των δεδομένων εκπαίδευσης	2007×1

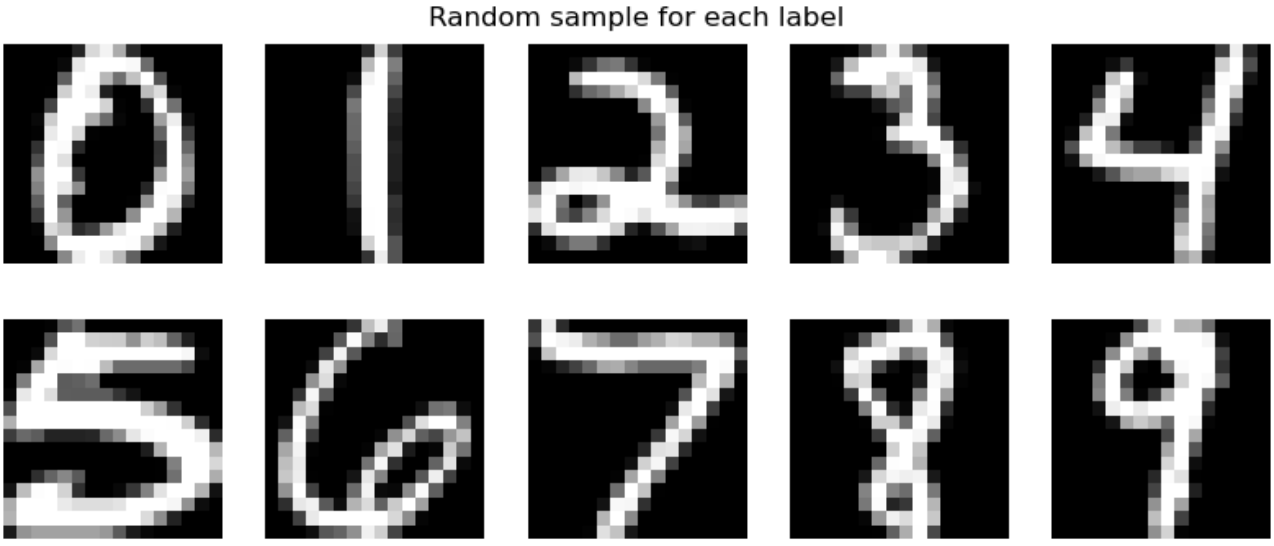
Βήμα 2: Απεικόνιση 131ου ψηφίου

Για να δούμε πως μοιάζουν οπτικά τα δεδομένα μας θα απεικονίσουμε το 131ο δεδομένο του train set. Επιλέγουμε την σειρά 130 (λαμβάνοντας υπόψιν πως η μέτρηση ξεκινάει από το 0) του πίνακα `X_train` και το κάνουμε `reshape(16, 16)` για να οργανώσουμε τα pixel σε ένα grid 16×16.



Βήμα 3: Απεικόνιση ενός τυχαίου δείγματος από κάθε κλάση

Στη συνέχεια παρουσιάζεται ένα τυχαίο δείγμα από κάθε label. Για να επιλεχθεί τυχαία ένα δείγμα από κάθε κλάση δημιουργούμε μία λίστα `samples` μήκους 10 (μία θέση για την κάθε κλάση) που κάθε στοιχείο της είναι ένας NumPy array διατάσεων $N_{each\ class\ \#samples} \times 256$ που έχει τα feature όλων των δειγμάτων που ανήκουν σε αυτή τη κλάση. Έτσι με `random.randint(0, len(samples[i]) - 1)` μπορούμε να επιλέξουμε τυχαία ένα δείγμα για κάθε ψηφίο.



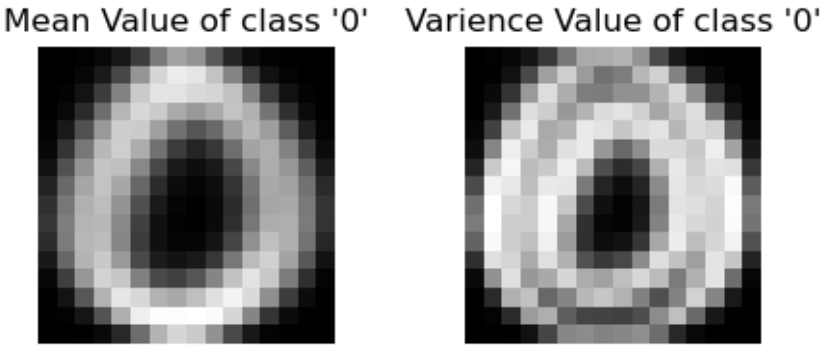
Βήμα 4 – 5: Υπολογισμός μέση τιμής και διασποράς του pixel (10, 10) των μηδενικών

Αφού έχουμε αποθηκευμένα όλα τα δείγματα κάθε κλάσης σε πίνακες ο υπολογισμός της μέσης τιμής και της διασποράς του pixel (10, 10) όλων των δειγμάτων "0" με χρήση των `np.mean` και `np.var`. Λαμβάνοντας υπόψι πως η αρίθμηση ξεκινάει από το 0 το pixel βρίσκεται στη θέση $10 \cdot 16 + 10 - 1 = 169$ και όχι 170.

	μέση τιμή	**διασπορά**
pixel (10, 10) όλων των μηδενικών	`-0.8568`	`0.1699`

Βήμα 6 - 8: Υπολογισμός και απεικόνιση μέσης τιμής και διασποράς όλων των pixel όλων των μηδενικών

Για τον υπολογισμό της μέσης τιμής και της διασποράς όλων pixel των δειγμάτων από την κλάση των μηδενικών κινηθήκαμε ανάλογα με τα βήματα 4 και 5 αλλά οι np.mean και np.var εφαρμόζονται σε όλα τα pixel (`np.mean(samples[0], axis=0)`).

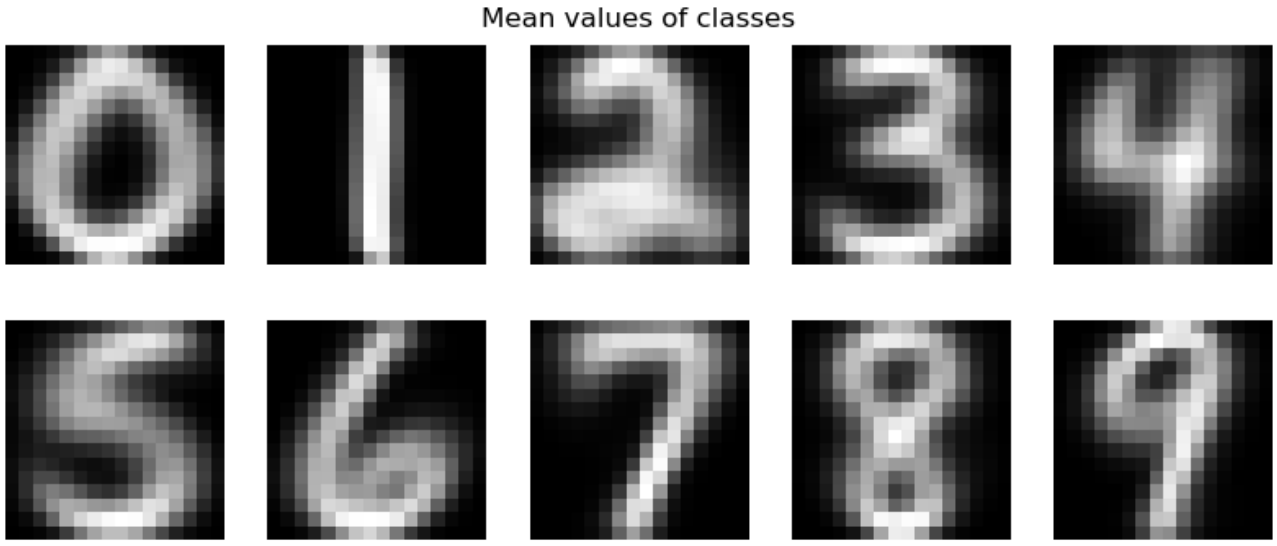


Στην απεικόνιση της μέσης τιμής παρατηρούμε πως παίρνουμε μία πιο θολή εκδοχή του ψηφίου "0", το οποίο είναι λογικό καθώς είναι ο μέσος όρων όλων των χειρόγραφων μηδενικών. Για την διασπορά παρατηρούμε πως τα pixel του περιγράμματος έχουν μεγαλύτερη τιμή, αφού σε αυτά τα pixel η διασπορά είναι υψηλότερη.

Και στις δύο περιπτώσεις το μηδενικό είναι ευδιάκριτο.

Βήμα 9: Υπολογισμός και απεικόνιση μέσης τιμής και διασποράς όλων κλάσεων

Για τον υπολογισμό της μέσης τιμής και της διασποράς όλων pixel των δειγμάτων από όλες τις κλάσεις κινηθήκαμε με ανάλογο τρόπο. Παρακάτω φαίνονται οι μέσες τιμές από όλες τις κλάσεις.



Παρατηρούμε πως σε όλες τις κλάσεις τα ψηφία είναι ευδιάκριτα. Το ψηφίο “1” είναι πιο ευδιάκριτο από τα υπόλοιπα ψηφία, γεγονός που υποδεικνύει περισσότερη ομοιογένεια στην γραφή του “1”.

Βήμα 10: Ταξινόμηση του 101ου ψηφίου του test set βάσει του ευκλείδειου ταξινομητή

Για τον υπολογισμό της Ευκλείδειας απόστασης των χαρακτηριστικών του 101ου ψηφίου χρησιμοποιήθηκε η ακόλουθη εντολή:

```
dist = np.linalg.norm(samples_mean - X_test[digit], ord=None, axis=1)
```

Το `np.linalg.norm` όρισμα `ord=None` υπολογίζει την Frobenius νόρμα της διαφοράς (2η κανονική νόρμα) η οποία ταυτίζεται με την Ευκλείδεια απόσταση.

Ευκλείδεια απόσταση	$d(p,q) = \sqrt{(p_1 - q_1)^2 + \cdots + (p_m - q_m)^2}$, όπου p η πραγματική τιμή ενός pixel και q η μέση τιμή του (συνολικά m pixel)
Frobenius νόρμα	$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{m}\|a_{ij}\|^2}$, όπου a_{ij} η διαφορά της μέσης τιμής από την πραγματική τιμή ενός pixel

Έτσι με τα τον υπολογισμό της ευκλείδειας απόστασης μεταξύ των pixel του 101ου ψηφίου και της μέσης τιμής κάθε κλάσης παίρνουμε με την συνάρτηση np.argmin την κλάσης όπου ελαχιστοποιείται η απόσταση.

```
The digit we want to classify is the 101th didgit of the test
set and as we can see below it is a '0'.
The euclidean classifier classifies it as a '0' which is True.
```

Η ταξινόμηση του 101ου ψηφίου έγινε σωστά ως ‘0’.

Βήμα 11: Ταξινόμηση όλων των ψηφίων βάσει του ευκλείδειου ταξινομητή και υπολογισμός ποσοστού επιτυχίας

Για την ταξινόμηση όλων των ψηφίων σε μία από τις 10 κατηγορίες αρχικά υπολογίζουμε την ευκλείδια απόσταση κάθε ψηφίου από τις μέσες τιμές κάθε κατηγορίας και στη συνέχεια επιλέγουμε την ελάχιστη. Οι προβλέψεις για κάθε ψηφίο αποθηκεύονται σε μία λίστα `predicts`.

Για το υπολογισμό του ποσοστού επιτυχίας του ταξινομητή που φτιάξαμε μετράμε πόσες φορές η λίστα predicts ισούται με τη λίστα των labels και στη συνέχεια το διαιρούμαι με το σύνολο των ψηφίων προς ταξινόμηση.

Ποσοστό επιτυχίας Ευκλειδιου ταξινομητή	81.42%

Βήμα 12: Υλοποίηση του ευκλείδειου ταξινομητή ως μία κλάση σαν ένα scikit-learn estimator

Τώρα καλούμαστε να υλοποιήσουμε τον παραπάνω ταξινομητή σε μία κλάση. Χρησιμοποιούμε τις ίδιες εντολές απλά πλέον τις βάζουμε μέσα σε συναρτήσεις της κλάσης `EuclideanDistanceClassifier`. Η κλάση μας έχει 3 συναρτήσεις ώστε να λειτουργεί ως ένας scikit-learn estimator και να μπορούμε να χρησιμοποιήσουμε τις έτοιμες συναρτήσεις από την `scikit-learn`.

Συνάρτηση	return	Περιγραφή
`fit(self, X, y)`	`self`	Δέχεται ως είσοδο δύο πίνακες `X` και `y` με δείγματα και τα labels τους αντίστοιχα και επιστρέφει τον ταξινομητή εκπαιδευμένο
`predict(self, X)`	`predicts`	Δέχεται ως όρισμα έναν πίνακα `X` με δείγματα προς ταξινόμηση και επιστρέφει τις προβλέψεις
`score(self, X, y)`	`acc`	Δέχεται ως όρισμα δύο πίνακες `X` και `y` με δείγματα προς ταξινόμηση και τα label τους και επιστρέφει το ποσοστό επιτυχίας

Βήμα 13α: Υπολογισμός score του ευκλείδειου ταξινομητή με χρήση 5-fold cross-validation

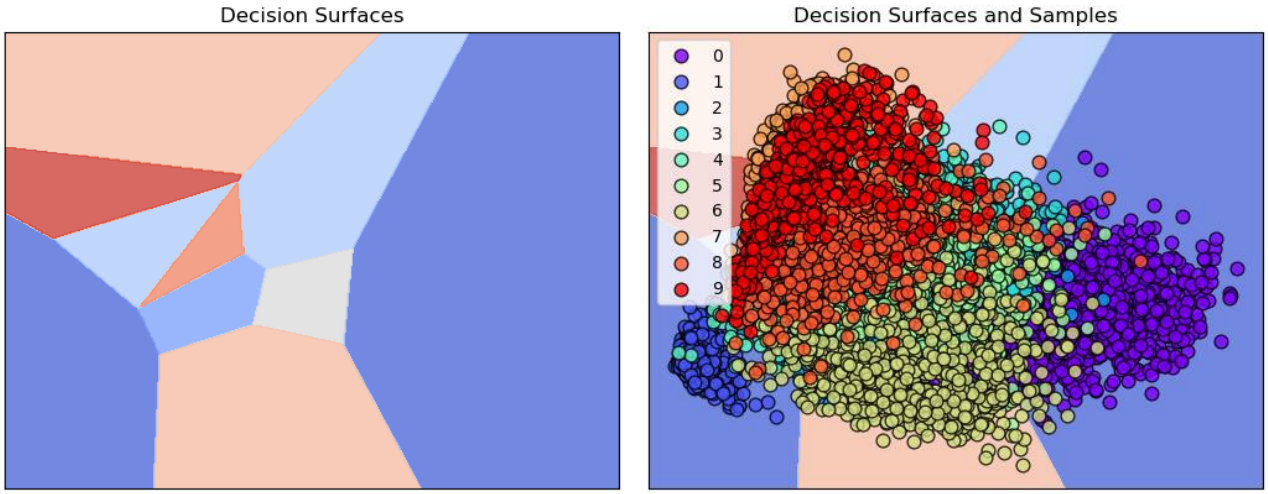
Τώρα θα εφαρμόσουμε την *5-fold-cross-validation* μέθοδο για να αξιολογήσουμε καλύτερα τον ταξινομητή. Βάσει αυτής της μεθόδου ανακατεύουμε το train set και το χωρίζουμε σε k (για εμάς $k = 5$) μέρη και γίνονται k εκπαιδεύσεις. Σε κάθε εκπαίδευση λαμβάνουμε το 1 μέρος ως trin set και τα υπόλοιπα $k - 1$ μέρη ως test set. Εν τέλει λαμβάνουμε ως ποσοστό επιτυχίας το μέσο όρος των επιμέρους ποσοστών. Για να υλοποιηθεί αυτή η μέθοδος χρησιμοποιήθηκε η συνάρτηση `cross_validate`.

```
The score of the Euclidean Classifie with 5-fold cross-validation is 85.14%.
Before 5-fold cross-validation it was 81.42%.
So it is more by 3.73%.
```

Η μικρή αύξηση του ποσοστού επιτυχίας οφείλεται στο γεγονός πως τώρα το test set μας έχει πολύ περισσότερα δείγματα από το προηγούμενο test set.

Βήμα 13β: Περιοχές απόφασης του ευκλείδειου ταξινομητή

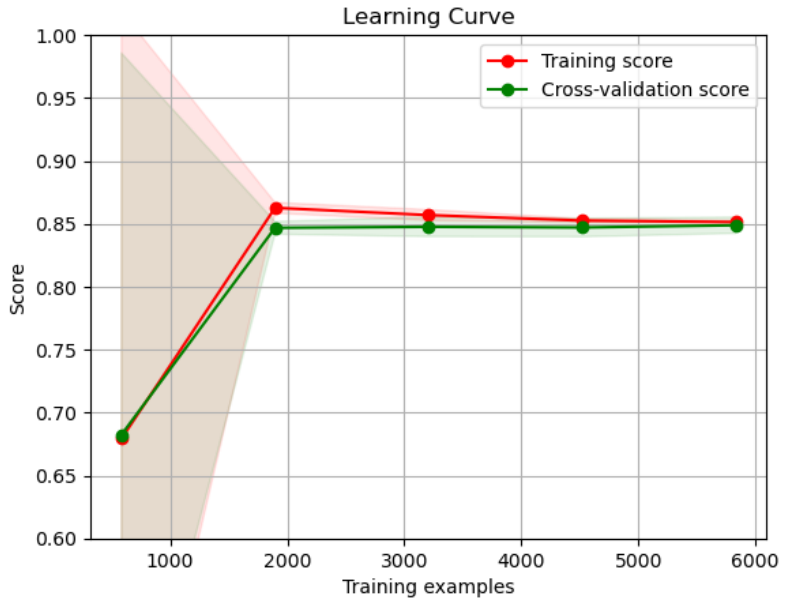
Η σχεδίαση των περιοχών απόφασης με 256 χαρακτηριστικά δεν είναι δυνατή και για αυτό χρησιμοποιούμε *Principal Component Analysis* (*PCA*) για να μειώσουμε σε 2 χαρακτηριστικά. Αφού μειώσουμε σε 2 χαρακτηριστικά κάνουμε fit τον ταξινομητή μας και απεικονίζουμε τις περιοχές απόφασης με την συνάρτηση `plot_clf` που μας δίνεται.



Παρατηρούμε πως τα δείγματα έχουν ταξινομηθεί κατά κύριο λόγο σωστά στις κατηγορίες τους.

Βήμα 13γ: Καμπύλη εκμάθησης του ευκλείδειου ταξινομητή (learning curve)

Χρησιμοποιώντας την συνάρτηση `plot_learning_curve` που μας δίνεται μπορούμε να αναπαραστήσουμε τις καμπύλες εκμάθησης για διαφορετικό αριθμό train samples.



Όσο αυξάνεται ο αριθμός των δειγμάτων το training score πέφτει γιατί το πρόβλημα δυσκολεύει, αφού έχει να ταξινομήσει περισσότερα χειρόγραφα ψηφία και εν τέλει

σταθεροποιείται και δεν μπορεί να βελτιωθεί περαιτέρω. Αντίθετα το cross-validation score αυξάνεται γιατί έρχεται αντιμέτωπο με μεγαλύτερη γκάμα περιπτώσεων. Παρατηρούμε πως με την αύξηση των δειγμάτων τα δύο ποσοστά συγκλίνουν. Η αύξηση δηλαδή των δειγμάτων εκπαίδευσης άνω των 6000 φαίνεται πως δεν έχει νόημα.

Βήμα 14: Υπολογισμός a priori πιθανοτήτων

Σε αυτό το βήμα, υπολογίστηκε η a priori πιθανότητα για καθένα από τα 9 διαφορετικά ψηφία, ως ο λόγος της συχνότητας εμφάνισης κάθε ψηφίου προς το συνολικό πλήθος ψηφίων στο training set. Αυτό αντικατοπτρίζεται μέσω της σχέσης:

$$\mathrm{P}(y_i) = \frac{\#samples\_i}{\#samples\_all}, i \in (0,9)$$

Τα αποτελέσματα ανά κλάση φαίνονται στον κάτωθι πίνακα:

Ψηφίο	$\boldsymbol{P(y_i)}$ **in %**
0	**16.37**
1	13.78
2	10.02
3	9.02
4	8.94
5	7.65
6	9.1
7	8.84
8	7.43
9	8.83

Βήμα 15 και 16: Υλοποίηση Naive Bayes Classifier και σύγκριση με υλοποίηση της scikit-learn.

Σε αυτό το βήμα πραγματοποιείται η υλοποίηση ενός Naive Bayes Classifier, σε μορφή συμβατή με scikit-learn, όπως και στο Βήμα 12. Συγκεκριμένα, για κάθε εικόνα, ο Classifier καλείται να υπολογίσει την posterior πιθανότητα να ανήκει σε καθεμία από τις 9 διαφορετικές κλάσεις ψηφίων, μέσω του κανόνα του Bayes:

$$P(y_i|x) = \frac{P(x|y_i) * P(y_i)}{P(x)}$$

όπου P(y_i) είναι η a priori πιθανότητα ανά κλάση. Η πιθανότητα P(x) για την εμφάνιση του διανύσματος εισόδου είναι ίδια ανά κλάση, συνεπώς δεν επιδρά στον υπολογισμό της a posterior πιθανότητας.

Όσον αφορά την πιθανότητα $P(x|y_i)$, στην υλοποίηση του naive bayes, υποθέτουμε ότι η κατανομή για κάθε pixel x_j είναι κανονική και ανεξάρτητη από τις κατανομές των υπολοίπων pixels. Η μέση τιμή και η διασπορά που χρησιμοποιείται ανά κλάση και pixel για τον υπολογισμό του $P(x|y_i)$ ισούται με τις τιμές που υπολογίστηκαν από το βήμα 9 (σε όλο το training set). Έτσι, λαμβάνουμε τις κάτωθι σχέσεις για την πιθανότητα $P(x|y_i)$:

$$P(x|y_i) = \prod_{j=1}^{256} P(x_j|y_i),\ i \in (0,9) \text{ με}$$

$$P(x_j|y_i) = \frac{1}{\sqrt{2*\pi*{\sigma_j}^2}} * e^{\frac{(x_j-\mu_j)^2}{2*{\sigma_j}^2}},\ j \in (1,256)$$

Τελικά, η νικήτρια κλάση δίνεται μέσω του argmax operator μέσω της σχέσης:

$$y = \text{argmax}\ P(y_i),\ i \in (0,9)$$

Στην συνέχεια, πραγματοποιείται σύγκριση ανάμεσα στην custom υλοποίηση του Naive Bayes Classifier – με τις τιμές διασποράς του βήματος 9 και με μοναδιαίες τιμές διασποράς – και στην δοθείσα υλοποίηση από την scikit-learn. Τα αποτελέσματα στο testing set φαίνονται παρακάτω.

Classifier	Validation Accuracy
Custom Naive Bayes	73.04%
Custom Naive Bayes (unit variance)	81.27%
Scikit-Learn Naive Bayes	71.95%

Παρατηρούμε ότι το μεγαλύτερο ποσοστό ευστοχίας επιτυγχάνεται από τον custom NB classifier με μοναδιαίες τιμές συνδιακύμανσης, μιας και σε αυτή την περίπτωση ο Naive Bayes μετατρέπεται σε Ευκλείδειο Ταξινομητή ο οποίος καθορίζεται μονάχα από τις μέσες τιμές ανά κλάση.

Βήμα 17: Σύγκριση ταξινομητών

Σε αυτό το βήμα, πραγματοποιείται σύγκριση των ταξινομητών των βημάτων 12, 15 και 16 με τους ταξινομητές SVM και knn. Συγκεκριμένα, για τους τελευταίους, χρησιμοποιήθηκαν οι υλοποιήσεις της scikit-learn και η σύγκριση βασίστηκε στο accuracy τόσο με 5-cross validation όσο και χωρίς. Τα αποτελέσματα αναγράφονται στον κάτωθι πίνακα:

Classifier	5 cross-validation	Testing accuracy
Euclidean	97.67%	81.42%
Custom Naive Bayes	77.73%	73.04%
Custom Naive Bayes (unit variance)	85.15%	81.27%
Scikit-Learn Naive Bayes	97.68%	71.95%
SVM (linear kernel)	99.99%	92.63%
SVM (rbf kernel)	99.38%	94.72%
KNN (n=5)	97.68%	94.47%

Παρατηρούμε ότι το καλύτερο accuracy τόσο με 5-cross validation όσο και στο testing set δίνεται από τον ταξινομητή SVM (με linear και με rbf kernel αντίστοιχα). Ο επόμενος ταξινομητής με συγκρίσιμα ποσοστά ακρίβειας είναι ο KNN, ενώ οι ταξινομητές Naive Bayes και Euclidean υπολείπονται σημαντικά στις παραπάνω μετρικές.

Βήμα 18: Τεχνική Ensembling

Σε αυτό το βήμα καλούμαστε να συνδυάσουμε μερικούς από τους ταξινομητές που υλοποιήθηκαν παραπάνω, ώστε να αυξήσουμε το συνολικό accuracy. Για να γίνει η εν λόγω επιλογή – πέρα από τις επιμέρους μετρικές για κάθε ταξινομητή – οφείλουμε να συνδυάσουμε ταξινομητές που χαρακτηρίζονται από διαφορετικό τύπο λαθών. Παρακάτω, παρουσιάζονται τα λάθη των ταξινομητών ανά ψηφίο καθώς και η κλάση στην οποία παρατηρούνται τα περισσότερα λάθη:

Classifier	0	1	2	3	4	5	6	7	8	9	Worst Class
Euclidean	**62**	5	53	35	50	37	27	30	38	36	0
Custom Naive Bayes	60	10	58	82	**139**	84	15	18	50	25	4
Custom Naive Bayes (unit variance)	**62**	5	53	35	50	38	27	30	39	37	0
Scikit-Learn Naive Bayes	64	10	57	88	**144**	86	19	15	56	24	4

SVM (linear kernel)	8	8	17	20	20	**24**	10	12	22	7	5
SVM (rbf kernel)	4	10	14	**19**	12	9	11	9	11	7	3
KNN (n=5)	5	5	16	12	**17**	16	7	9	15	9	4

Με βάση τα παραπάνω αποτελέσματα, αποφασίστηκε να χρησιμοποιηθούν οι 3 ταξινομητές με την μεγαλύτερη ακρίβεια από το βήμα 17, δηλαδή οι 2 SVM classifiers και ο KNN, μιας και ο καθένας έχει αυξήμενα λάθη σε διαφορετικά ψηφία (3, 4 και 5). Συγκεκριμένα, οι εν λόγω επιμέρους ταξινομητές συνδυάζονται σε έναν VotingClassifier, ο οποίος αποτελεί έναν μεταταξινομητή. Σημειώνεται ότι το πλήθος των ταξινομητών πρέπει να είναι περιττό ώστε να μπορεί ο VotingClassifier (με hard voting) να λάβει μία απόφαση σε κάθε test case και να αποφευχθεί η περίπτωση ισοπαλίας στις ψήφους (όπως θα μπορούσε να συμβεί με άρτιο πλήθος classifiers).

Παρακάτω, παρατίθενται τα αποτελέσματα του VotingClassifier με soft και hard voting. Με soft voting, υπολογίζεται η πιθανότητα ανά κλάση από κάθε επιμέρους ταξινομητή και οι πιθανότητες αυτές αθροίζονται. Η νικήτρια κλάση είναι αυτή με την μεγαλύτερη αθροιστική πιθανότητα. Με hard voting, κάθε ταξινομητής υπολογίζει ανεξάρτητα την νικήτρια κλάση και αυτή με τις περισσότερες ψήφους είναι η συνολική νικήτρια (εξού και η ανάγκη για μόνο πλήθος sub-classifiers)

Type of voting	Testing Accuracy
Soft Voting	**95.11%**
Hard Voting	94.91%

Παρατηρούμε ότι και στους 2 τρόπους ψηφοφορίας, λαμβάνουμε παρεμφερή αποτελέσματα, με καλύτερο accuracy να παρατηρείται στην τεχνική soft voting.

Τέλος, υλοποιήθηκε και ένας BaggingClassifier, ο οποίος γίνεται ensemble με έναν άλλον classifier. Η λειτουργία του έγκειται στον διαχωρισμό του training set σε επιμέρους υποσύνολα και την εφαρμογή του ταξινομητή σε καθένα από αυτά. Η τελική απόφαση βγαίνει μέσω ψηφοφορίας ή μέσου όρου των προβλέψεων των επιμέρους ταξινομητών. Στο δοθέν dataset, ο BaggingClassifier έγινε ensemble με τον SVM classifier (με rbf kernel) λόγω των υψηλών accuracy scores που διαθέτει αλλά και με τον DecisionTreeClassifier, με σκοπό την υλοποίηση του RandomForest.

Ensemble Classifier	Testing Accuracy
SVM (rbf kernel)	94.71%
Decision Trees	87.14%

Σε επίπεδο testing accuracy, παρατηρούμε ότι ο προκύπτων RandomForest υπολείπεται σημαντικά σε σχέση με τον συνδυασμό BaggingClassifier με SVM.

Βήμα 19: Εισαγωγή στα Νευρωνικά Δίκτυα

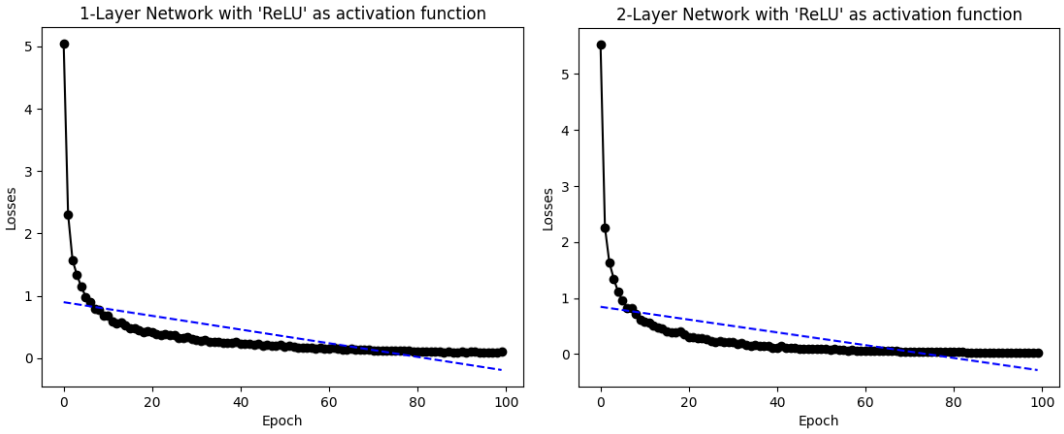
α) Για την αναγνώριση των δεδομένων υλοποιήσαμε μία κλάση DigitDataset που κληρονομεί τα χαρακτηριστικά της κλάσης Dataset. Η κλάση έχει 3 συναρτήσεις:

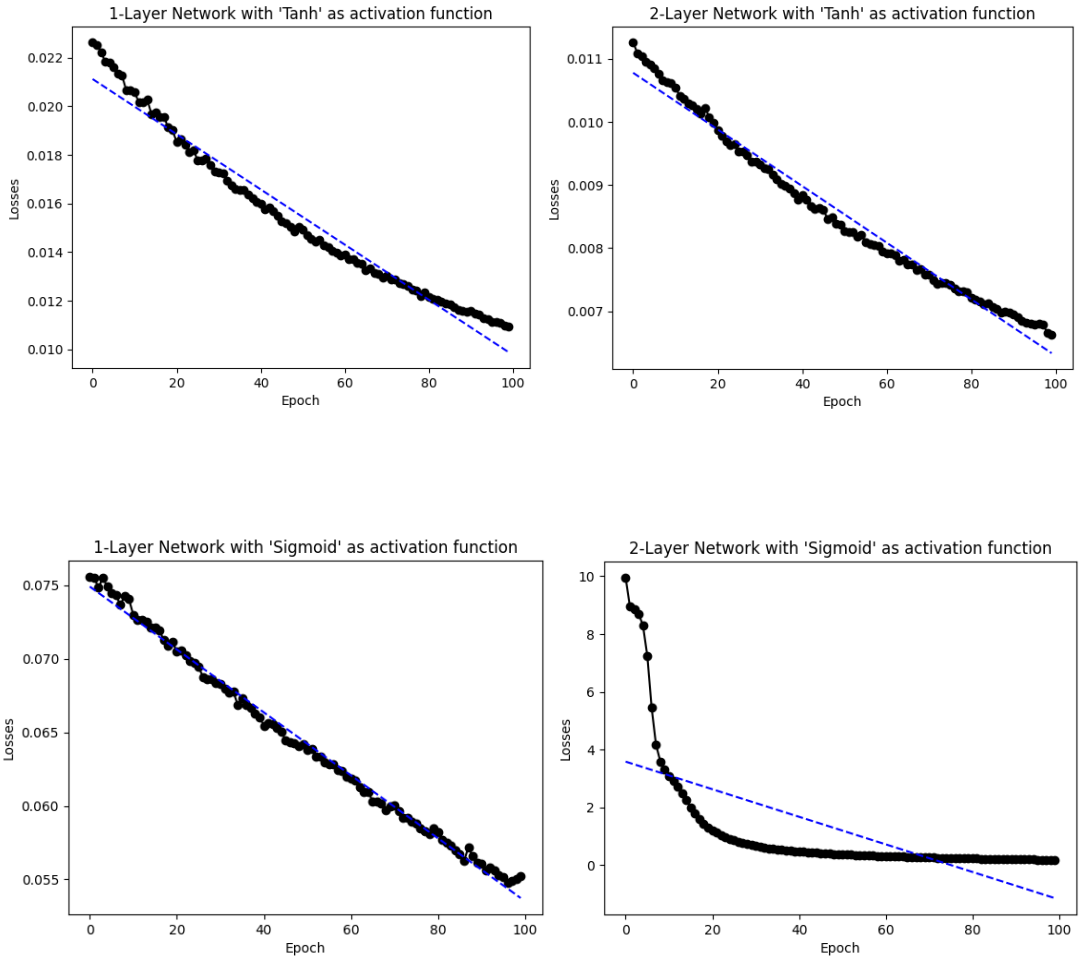
Συνάρτηση	Περιγραφή
`__len__(self)`	Επιστρέφει το πλήθος των δεδομένων
`__getitem__(self, idx)`	Επιστρέφει ένα στοιχείο των δεδομένων μας

Έπειτα για τον χωρισμό σε batches χρησιμοποιήθηκε η συνάρτηση `DataLoader`.

β) Υλοποιήσαμε 6 νευρωνικά δίκτυα τα οποία διαφοροποιούνται τόσο ως προς το πλήθος των hidden layers όσο και ως προς την συνάρτηση ενεργοποίησης. Πιο αναλυτικά κατασκευάστηκαν ως υποκλάσεις της `nn.Module` νευρωνικά δίκτυα με 1 ενδιάμεσο επίπεδο (*1-layer*) και με 2 ενδιάμεσα επίπεδα (*2-layer*) με 3 συναρτήσεις ενεργοποίησης το καθένα. Οι συναρτήσεις ενεργοποίησης είναι οι *ReLU*, *Sigmoid* και *Tanh*.

Παρακάτω μπορούμε να δούμε τα αποτελέσματα της προπόνησης (ως προς το loss metric) για τα νευρωνικά δίκτυα που υλοποιήθηκαν, με 50 κόμβους ανά middle layer και με 100 εποχές προπόνησης.





Παρατηρούμε ότι τα χαμηλότερα loss scores σε όλες τις εποχές επιτυγχάνονται από τα νευρωνικά με συνάρτηση ενεργοποίησης Tanh. Από την άλλη πλευρά, η ύπαρξη ενός ή δύο ενδιάμεσων layer δεν επηρεάζει συστηματικά το training loss, μιας και στην περίπτωση της χρήσης sigmoid και ReLU ως activation functions, αυξημένο loss παρατηρείται στα 2-layer δίκτυα. Απεναντίας, για την Tanh, στην περίπτωση του 2-layer NN το loss ανά εποχή μειώνεται.

γ) Στη συνέχεια έγινε εκπαίδευση των νευρωνικών σε εποχές και έγινε evaluation, τα αποτελέσματα του οποίου είναι αμφισβητήσιμα αφού βγάζουν accuracy < 0.5, που ακόμα και ένα εντελώς τυχαίος ταξινομητής θα έβγαζε. Παράλληλα, δεν έγινε μεταφορά της υλοποίησης σαν κλάση ταξινομητή συμβατή με την scikit-learn. Στον παρακάτω πίνακα, φαίνονται στα accuracy scores στο testing set.

#Layers	Activation Function	Testing accuracy
1-layer	ReLU	7.42%
2-layer	ReLU	5.97%

1-layer	Tanh	11.6%
2-layer	Tanh	11.06%
1-layer	Sigmoid	**12.55%**
2-layer	Sigmoid	3.73%

δ) Παρατηρούμε ότι κατά μέσο όρο τα υψηλότερα accuracy scores λαμβάνονται από την συνάρτηση ενεργοποίησης Tanh, πράγμα εύλογο λόγω των χαμηλών loss values κατά την προπόνηση των αντίστοιχων δικτύων. Το υψηλότερο score δίνεται από το 1-layer NN με sigmoid activation functions. Τέλος, η χρήση της ReLU οδηγεί συστηματικά σε accuracy < 10%.