ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ: ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΜΑΘΗΜΑ: ΑΝΑΓΝΩΡΙΣΗ ΠΡΟΤΥΠΩΝ

1ο ΕΡΓΑΣΤΗΡΙΟ

ΟΝΟΜΑΤΕΠΩΝΥΜΟ : ΣΤΑΜΑΤΗΣ ΑΛΕΞΑΝΔΡΟΠΟΥΛΟΣ
ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ: 03117060 (el17060)

ΟΝΟΜΑΤΕΠΩΝΥΜΟ : ΠΟΛΥΤΙΜΗ-ΑΝΝΑ ΓΚΟΤΣΗ
ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ:03117201 (el17201)

Εξάμηνο: 9ο

Πίνακας περιεχομένων

Στόχος Εργασίας **3**

Βήμα 1: Φόρτωση Δεδομένων **3**

Βήμα 2: Σχεδιασμός ενός ψηφίου από τα train δεδομένα **4**

Βήμα 3: Σχεδιασμός ενός τυχαίου δείγματος από κάθε κλάση **4**

Βήμα 4: Υπολογισμός μέσης τιμής χαρακτηριστικών για συγκεκριμένο pixel ενός ψηφίου **5**

Βήμα 5: Υπολογισμός διασποράς χαρακτηριστικών για συγκεκριμένο pixel ενός ψηφίου **5**

Βήμα 6: Υπολογισμός μέσης τιμής και διασποράς κάθε pixel για συγκεκριμένο ψηφίο **6**

Βήμα 7: Σχεδιασμός ψηφίου χρησιμοποιώντας τις τιμές της μέσης τιμής **7**

Βήμα 8: Σχεδιασμός ψηφίου χρησιμοποιώντας τις τιμές της διασποράς **8**

Συμπεράσματα για τα βήματα 7,8: *8*

Βήμα 9: Υπολογισμός μέσης τιμής και διασποράς όλων των ψηφίων **9**

Βήμα 10: Ταξινόμηση ψηφίου των test δεδομένων με Euclidean Classifier **9**

Βήμα 11: Ταξινόμηση test set δεδομένων με βάση τον Euclidean Classifier **10**

Βήμα 12: Υλοποίηση Euclidean Classifier σαν ένα scikit-learn estimator **11**

Βήμα 13: 5 -Fold Cross-Validation, Περιοχή Απόφασης, Learning Curve **11**

α) 5 -Fold Cross-Validation *11*

β) Σχεδιασμός Περιοχής Απόφασης Ευκλείδειου Ταξινομητή *12*

γ) Σχεδιασμός learning curve Ευκλείδειου Ταξινομητή *13*

Βήμα 14: Υπολογισμός A-Priori πιθανοτήτων **15**

Βήμα 15: Naive Bayes Classifier **16**

Βήμα 16: Naive Bayes Classifier με μοναδιαία διασπορά για όλα τα χαρακτηριστικά **18**

Βήμα 17: Σύγκριση Naïve Bayes, Nearest Neighbors, SVM **19**

Βήμα 18: Voting και Bagging Classifier **21**

α) Voting Classifier *21*

Συμπεράσματα Voting Classifier 23

β) Bagging Classifier *23*

Συμπεράσματα Bagging Classifier 24

γ) Σύγκριση Votting Classifier και Bagging Classifier *25*

Βήμα 19: Κατασκευή Νευρωνικού Δικτύου **25**

Στόχος Εργασίας

Σκοπός αυτής της εργαστηριακής άσκησης είναι η υλοποίηση ενός συστήματος οπτικής αναγνώρισης χειρόγραφων ψηφίων. Τα δεδομένα προέρχονται από την US Postal Service, περιέχουν τα ψηφία από το 0 έως το 9 και διακρίνονται σε train και test. Στο κομμάτι της προπαρασκευής, πραγματοποιήθηκε μια αναλυτική μελέτη του dataset, από όπου και εξήχθησαν στατιστικά συμπεράσματα και στην συνέχεια ακολούθησε η υλοποίηση ενός ευκλείδειου ταξινομητή για την κατηγοριοποίηση των εικόνων ψηφίων.

Βήμα 1: Φόρτωση Δεδομένων

Αρχικά μας δόθηκαν δύο αρχεία που περιέχουν τα testing και training δεδομένα : test.txt και train.txt αντίστοιχα. Επεξεργαστήκαμε αυτά τα δεδομένα, διαβάζοντάς τα από τα αρχείο και δημιουργώντας τέσσερις numpy πίνακες X_train, X_test, y_train και y_test. Πιο συγκεκριμένα, στο καθένα από τα δύο δοσμένα αρχεία, κάθε γραμμή αφορά ένα ψηφίο (δείγμα), ενώ για κάθε ψηφίο δίνονται στην αντίστοιχη γραμμή 257 τιμές, εκ των οποίων η πρώτη αντιστοιχεί στο ίδιο το ψηφίο (0,…,9) και οι υπόλοιπες 256 είναι τα χαρακτηριστικά (features) που το περιγράφουν (grayscale values). Έτσι, αφού διαβάσαμε το αρχείο train.txt σε ένα δισδιάστατο πίνακα, χωρίσαμε αυτόν στους πίνακες X_train και y_train, με τον πρώτο (δισδιάστατο) να περιλαμβάνει στην γραμμή i τα 256 χαρακτηριστικά του δείγματος i και τον δεύτερο (μονοδιάστατο) να περιέχει στην θέση j την ετικέτα για το δείγμα j (δηλαδή ποιο ψηφίο αυτό είναι). Έτσι, ο πίνακας Xtrain είναι ένας 2D numpy πίνακας με διάσταση 7291 x 256 (έχουμε 7291 training samples) και ο πίνακας ytain είναι ένας 1D numpy πίνακας με διάσταση 1 x 7291. Ακριβώς την ίδια διαδικασία ακολουθήσαμε για το αρχείο test.txt, οπότε και προέκυψαν ο πίνακας X_test (2D numpy πίνακας με διάσταση 2007 x 256) και ο πίνακας y_test (1D numpy πίνακας με διάσταση 1 x 2007).

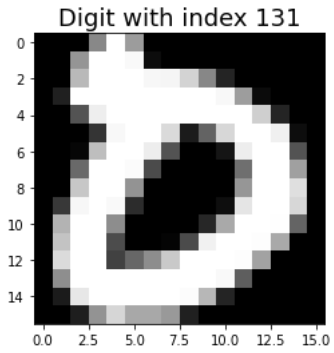
Τυπώνοντας τις διαστάσεις στον κώδικα python:

```
Dimensions of train labels array: (7291,)
Dimensions of train samples array: (7291, 256)
Dimensions of test labels array: (2007,)
Dimensions of test samples array: (2007, 256)
```

Σημειώνουμε διαβάζοντας τα δοσμένα αρχεία σε πίνακες λάβαμε τιμές με μορφή numpy.str_. Στην συνέχεια, τις τιμές αυτές τις μετατρέψαμε σε μορφή numpy.float64 στους πίνακες X_train, y_train, X_test, y_test, για λόγους υπολογιστικής ευκολίας στην συνέχεια.

Βήμα 2: Σχεδιασμός ενός ψηφίου από τα train δεδομένα

Στην συνέχεια σχεδιάσαμε το υπ' αριθμόν 131 ψηφίο από τα δεδομένα εκπαίδευσης. Για το σκοπό αυτό υλοποιήσαμε τη συνάρτηση show_sample() του αρχείου lib.py. Σε αυτήν, χρησιμοποιήσαμε τη συνάρτηση numpy.reshape για να οργανώσουμε τα 256 χαρακτηριστικά του ψηφίου αυτού σε ένα πίνακα 16x16. Το ζητούμενο ψηφίο απεικονίζεται παρακάτω με τη βοήθεια της συνάρτησης matplotlib.pyplot.imshow.



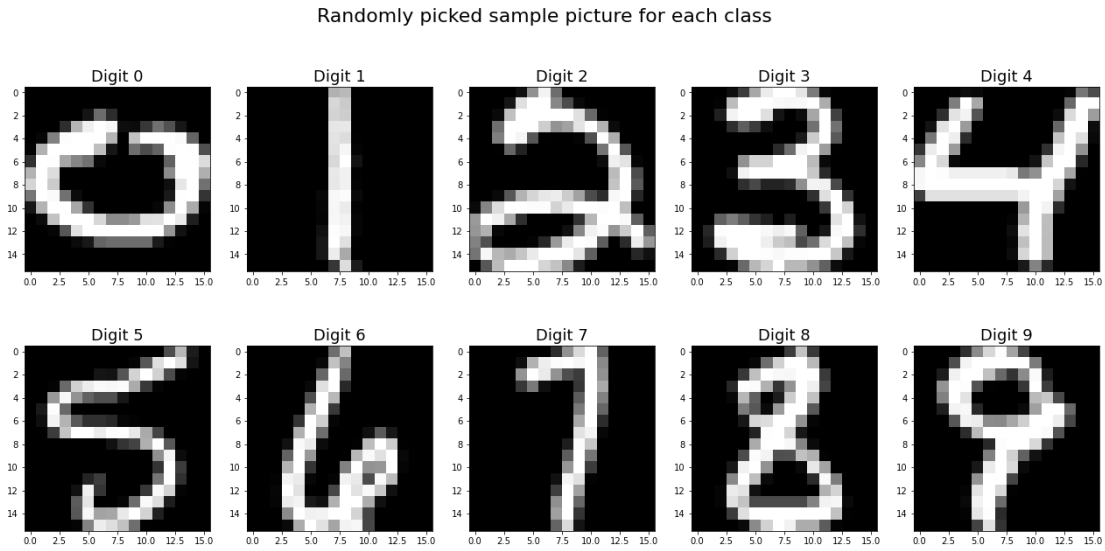
Το παραπάνω ψηφίο μοιάζει με το μηδέν, αλλά δεν είμαστε απόλυτα σίγουροι. Για το σκοπό αυτό τυπώνουμε και το αντίστοιχο label, προκειμένου να επιβεβαιώσουμε την υπόθεσή μας:

```
Digit is: 0
```

Βήμα 3: Σχεδιασμός ενός τυχαίου δείγματος από κάθε κλάση

Στην συνέχεια υλοποιήσαμε τη συνάρτηση plot_digits_samples() του αρχείου lib.py, η οποία επιστρέφει ένα τυχαίο δείγμα από κάθε ψηφίο και τα τυπώνει με τη σωστή σειρά. Για τον σκοπό αυτό η συνάρτηση δημιουργεί έναν δισδιάστατο πίνακα στον οποίο τοποθετεί στην σειρά i όλα τα indexes των σειρών του πίνακα δειγμάτων που έλαβε, οι οποίες σύμφωνα με τον πίνακα από ετικέτες αντιστοιχούν στο ψηφίο i. Ο υπολογισμός του πίνακα αυτού πραγματοποιείται στην βοηθητική συνάρτηση get_digit_indexes() που υλοποιήσαμε στο αρχείο lib.py. Αφού δημιουργήσει τον πίνακα αυτό, η plot_digits_samples() επιλέγει τυχαία ένα από τα δείγματα για κάθε ψηφίο και δημιουργεί την γραφική του παράσταση.

Παρακάτω φαίνεται ένα από τα αποτελέσματα που προκύπτουν από την εκτέλεση της συνάρτησης αυτής.



Μπορούμε να παρατηρήσουμε καλώντας ξανά την συνάρτηση plot_digits_samples(), πως κάθε φορά επιλέγονται διαφορετικά τυχαία δείγματα για τα ψηφία, και πως τα δείγματα κάθε ψηφίου διαφέρουν συχνά μεταξύ τους αρκετά, γεγονός που καθιστά και την χρήση ενός κατάλληλου ταξινομητή απαραίτητη για την κατηγοριοποίηση των διαφόρων δειγμάτων.

Βήμα 4: Υπολογισμός μέσης τιμής χαρακτηριστικών για συγκεκριμένο pixel ενός ψηφίου

Στόχος του βήματος αυτού ήταν η εύρεση της μέσης τιμής ενός συγκεκριμένου pixel ενός συγκεκριμένου ψηφίου. Συγκεκριμένα, ζητήθηκε ο υπολογισμός της μέσης τιμής του pixel (10, 10) για το ψηφίο «0» με βάση τα train δεδομένα. Πραγματοποιήσαμε τον υπολογισμό μέσω της συνάρτησης digit_mean_at_pixel() του αρχείου lib.py, οπότε και λάβαμε το αποτέλεσμα:

```
Mean value is: -0.9272646566164154
```

Το pixel 10 x 10 βρίσκεται στα μέσα του δακτυλίου που σχηματίζει το ψηφίο 0 αλλά όχι στο κέντρο του. Από τη μέση τιμή που βρήκαμε, καταλαβαίνουμε ότι συνήθως το pixel αυτό είτε είναι κενό (μαύρο) είτε πολύ ελαφρά βαμμένο, γεγονός που οφείλεται στο διαφορετικό πάχος της γραφίδας. Μπορούμε να διαπιστώσουμε το διαφοροποιείται ο χρωματισμός του από τη τιμή της διασποράς στο επόμενο βήμα.

Βήμα 5: Υπολογισμός διασποράς χαρακτηριστικών για συγκεκριμένο pixel ενός ψηφίου

Στόχο του βήματος αυτού αποτελούσε η εύρεση της διασποράς ενός συγκεκριμένου pixel ενός συγκεκριμένου ψηφίου και συγκεκριμένα ο υπολογισμός της διασποράς του pixel (10,10) για το ψηφίο «0» με βάση τα train δεδομένα. Αυτό πραγματοποιήθηκε

μέσω της συνάρτησης digit_variance_at_pixel() στο αρχείο lib.py και το που λάβαμε είναι το εξής:

```
Variance value is: 0.08399490726614352
```

Καθώς η τιμή της διασποράς είναι πολύ μικρή και αφού η μέση τιμή αυτής είναι -0.9272 (περίπου) συμπεραίνουμε ότι οι τιμές δεν αποκλίνουν σημαντικά από τη μέση τιμή και άρα το pixel αυτό είτε είναι κενό είτε πολύ ελαφρά βαμμένο στα δείγματα εκπαίδευσης που διαθέτουμε.

Βήμα 6: Υπολογισμός μέσης τιμής και διασποράς κάθε pixel για συγκεκριμένο ψηφίο

Στο βήμα αυτό υπολογίσαμε τη μέση τιμή και διασπορά των χαρακτηριστικών κάθε pixel για το ψηφίο «0» με βάση τα train δεδομένα, αφού πρώτα υλοποιήσαμε τις συναρτήσεις digit_mean() και digit_variance() του αρχείου lib.py αντίστοιχα. Συγκεκριμένα, η digit_mean() διατρέχει όλα τα pixel της εικόνας και καλεί σε κάθε ένα από αυτά την digit_mean_at_pixel() για τον υπολογισμό της μέσης τιμής για το χαρακτηριστικό αυτό. Τελικά επιστρέφει έναν 1D numpy array που περιέχει τις μέσες τιμές για όλα τα χαρακτηριστικά για το ψηφίο για το οποίο την καλέσαμε. Εντελώς ανάλογα λειτουργεί και η digit_variance().

Λαμβάνουμε τα εξής αποτελέσματα:

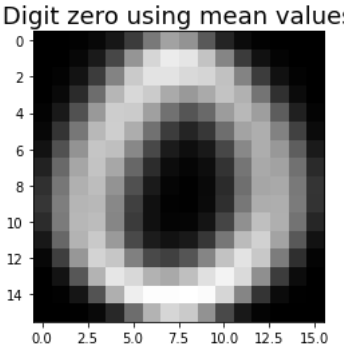
```
Mean values for digit 0 are:  [-0.99862814 -0.99539782 -0.98492295 -0.94125126 -0.83334255 -0.57142295
 -0.13158459  0.15260804  0.04628392 -0.35370101 -0.74124874 -0.92091625
 -0.98502513 -0.99718258 -0.99993467 -1.         -0.99823451 -0.99346566
 -0.94997069 -0.81149414 -0.48957538  0.06171608  0.5263928   0.6825
  0.63088945  0.39957621 -0.04891709 -0.57936348 -0.87249162 -0.96911139
 -0.99679983 -0.99999497 -0.99668174 -0.9728727  -0.86339531 -0.56290285
 -0.05700168  0.43772194  0.6278526   0.61645645  0.5500134   0.51951173
  0.37973953 -0.06139782 -0.56568928 -0.87724791 -0.97848995 -0.99811474
 -0.99374539 -0.93143049 -0.68924623 -0.21849832  0.29654858  0.5375134
  0.44771859  0.21163149  0.0910134   0.21372362  0.39326382  0.26515327
 -0.17200921 -0.64877387 -0.92078308 -0.99405025 -0.97799246 -0.82162228
 -0.41924121  0.09449414  0.46405611  0.44884757  0.13969012 -0.18907538
 -0.39019849 -0.23883166  0.11324456  0.3130067   0.12591374 -0.3365402
 -0.76123116 -0.97002764 -0.94678392 -0.62979229 -0.14879397  0.30968174
  0.46998325  0.23364992 -0.18668425 -0.57253601 -0.74139363 -0.60290536
 -0.21564405  0.17078894  0.24475042 -0.07310385 -0.53621943 -0.91532998
 -0.86794556 -0.41327471  0.05648911  0.4018727   0.36071106 -0.02580821
 -0.48111725 -0.80406198 -0.89878894 -0.81912982 -0.48869095 -0.02665159
  0.23628894  0.08193635 -0.32522781 -0.81318425 -0.75170101 -0.2495469
  0.18348911  0.39500921  0.18889615 -0.26513568 -0.68222529 -0.90914154
 -0.96112647 -0.90525544 -0.63567755 -0.16189615  0.2026608   0.15844891
 -0.1948258  -0.69670101 -0.64793216 -0.15201926  0.2530938   0.348067
  0.05766415 -0.42946566 -0.81036851 -0.95965997 -0.98827806 -0.93402094
 -0.68741206 -0.20671189  0.18481575  0.20578894 -0.12869347 -0.62338945
 -0.61312144 -0.10246901  0.29297404  0.31548911 -0.03203601 -0.53865913
 -0.8793593  -0.98113735 -0.99009045 -0.92726466 -0.66128894 -0.14991541
  0.23256784  0.24436935 -0.1204598  -0.6351675  -0.6791943  -0.1241608
  0.30576214  0.34223116 -0.04241374 -0.57421943 -0.88586013 -0.96591709
 -0.95777889 -0.85684506 -0.50418844  0.02544221  0.36323199  0.248866
 -0.18185092 -0.72877722 -0.82383082 -0.26320854  0.27515494  0.43617588
  0.09984255 -0.46859799 -0.79606533 -0.86990369 -0.81663233 -0.60312312
 -0.1465603   0.33558543  0.48892462  0.16248827 -0.36756198 -0.8678995
 -0.94752261 -0.54919347  0.07741206  0.50525544  0.41234925 -0.0868995
 -0.47579397 -0.53764489 -0.39347487 -0.05096901  0.38388358  0.61753518
  0.42662563 -0.11107705 -0.66541206 -0.96842211 -0.9939732  -0.85918174
 -0.37738861  0.2838526   0.64298827  0.53982747  0.2671206   0.20476047
  0.36249832  0.59526298  0.73646147  0.55646817  0.02170938 -0.57072194
 -0.91134171 -0.99459213 -0.99882747 -0.98417672 -0.8489196  -0.40532412
  0.25131575  0.69569598  0.80123869  0.80037521  0.82124288  0.78654941
  0.52453434 -0.05047822 -0.62455779 -0.91866918 -0.99191541 -0.99844054
 -0.99966248 -0.99798911 -0.99102764 -0.92893551 -0.70271022 -0.22043216
  0.28302848  0.52802513  0.43905193  0.04248827 -0.46701424 -0.82550586
 -0.96751089 -0.99652513 -0.99993635 -1.        ]


Variance values for digit 0 are:  [2.24710553e-03 6.61218108e-03 1.96070435e-02 6.29539251e-02
 1.80517711e-01 3.72425688e-01 5.32129066e-01 5.38652584e-01
 5.63057850e-01 4.58004391e-01 2.36125820e-01 8.27468429e-02
 1.21968225e-02 2.34697418e-03 2.25054864e-06 0.00000000e+00
 2.64590137e-03 6.10843679e-03 5.44937032e-02 2.28891036e-01
 4.99957965e-01 6.44135421e-01 4.85502183e-01 3.52813400e-01
```

```
3.92387709e-01 5.70234363e-01 6.52718455e-01 4.13231133e-01
1.49140835e-01 3.35148417e-02 2.23546619e-03 3.01507538e-08
5.20404783e-03 3.14306644e-02 1.64857981e-01 4.76595199e-01
6.88789851e-01 5.41577379e-01 3.90362785e-01 3.88269588e-01
4.51799131e-01 4.54445439e-01 5.59844163e-01 6.76945537e-01
4.56356382e-01 1.47290519e-01 2.15924446e-02 1.55355179e-03
4.14597368e-03 8.32034943e-02 3.53123478e-01 6.88346336e-01
6.41068459e-01 4.74419658e-01 5.60078092e-01 6.88158796e-01
7.10076394e-01 6.50761459e-01 5.17683527e-01 6.14311813e-01
6.92130527e-01 3.87860819e-01 9.27878817e-02 5.12100502e-03
1.91393956e-02 2.05134669e-01 6.06858572e-01 7.29689447e-01
5.29695079e-01 5.57585128e-01 7.30793599e-01 7.10336710e-01
6.18649738e-01 6.85906088e-01 7.09333105e-01 5.68199272e-01
6.86356168e-01 6.57376698e-01 2.70395809e-01 2.48985768e-02
4.40850480e-02 4.17875073e-01 7.42515186e-01 6.38162408e-01
5.37235505e-01 6.98720092e-01 7.29508904e-01 4.74217466e-01
3.25516987e-01 4.43293621e-01 6.79655574e-01 6.77065724e-01
6.26042325e-01 7.46769610e-01 5.11186331e-01 7.61127074e-02
1.12704196e-01 6.31760573e-01 7.68400941e-01 5.87753397e-01
6.28158843e-01 7.45823577e-01 5.83718058e-01 2.44491777e-01
1.38591431e-01 2.20492834e-01 5.50554522e-01 7.27714970e-01
6.42478838e-01 7.08914519e-01 6.94947665e-01 1.72906774e-01
2.33913432e-01 7.55480999e-01 7.25147358e-01 5.94271010e-01
7.25995567e-01 7.22193978e-01 3.80469844e-01 1.17147986e-01
4.31991265e-02 1.15479709e-01 4.34200884e-01 7.17030806e-01
6.73907492e-01 6.83998824e-01 7.77344538e-01 2.89288379e-01
3.50436769e-01 7.94669997e-01 6.71630923e-01 6.36640517e-01
7.73303781e-01 6.22014519e-01 2.35919167e-01 5.07989723e-02
1.20399193e-02 7.15966492e-02 3.84566269e-01 7.07973969e-01
6.92590109e-01 6.61402066e-01 7.89332749e-01 3.77955923e-01
3.85776254e-01 7.97320955e-01 6.39243646e-01 6.40600885e-01
7.73802377e-01 5.24883035e-01 1.50932276e-01 2.04257447e-02
9.22011419e-03 8.39949073e-02 4.10075030e-01 7.28030574e-01
6.73231711e-01 6.51889006e-01 7.84213907e-01 3.63540948e-01
3.08504371e-01 7.79277625e-01 6.37957603e-01 5.98517288e-01
7.51394657e-01 4.83112678e-01 1.46936648e-01 4.05883410e-02
5.44543082e-02 1.70038151e-01 5.24961809e-01 7.40375080e-01
5.89002146e-01 6.63904921e-01 7.54583681e-01 2.56664728e-01
1.47716979e-01 6.98442886e-01 6.64436284e-01 5.34499905e-01
6.96665715e-01 5.53315092e-01 2.56944480e-01 1.76332830e-01
2.25521646e-01 4.34592827e-01 6.92639770e-01 6.07276592e-01
4.97365974e-01 7.11134413e-01 6.18094593e-01 1.08415797e-01
3.41831432e-02 4.56040505e-01 6.98963381e-01 4.74563614e-01
5.27250088e-01 6.68950888e-01 5.36281114e-01 5.02052697e-01
5.97225542e-01 6.72443524e-01 5.84077077e-01 3.86144920e-01
5.48762343e-01 7.11081543e-01 3.41243475e-01 2.11326666e-02
2.95241839e-03 1.31238848e-01 5.52724120e-01 6.03826463e-01
3.25028972e-01 4.16482029e-01 5.75249085e-01 6.18132100e-01
5.54630131e-01 4.04193180e-01 2.50947600e-01 4.11167143e-01
6.55568540e-01 4.15272965e-01 7.81770198e-02 4.62918052e-03
7.84489907e-04 1.51501959e-02 1.35163611e-01 4.84368905e-01
5.44399884e-01 2.92923447e-01 2.12524624e-01 2.11261998e-01
2.00064900e-01 2.45288690e-01 4.36067513e-01 5.96362905e-01
3.32362057e-01 7.24819734e-02 5.11205068e-03 2.20130954e-03
9.47803680e-05 2.20209460e-03 6.10021466e-03 5.50333244e-02
2.15855312e-01 4.17985096e-01 4.39051166e-01 4.06222009e-01
4.20880554e-01 4.47276646e-01 3.40557090e-01 1.30286927e-01
2.49380171e-02 1.77901739e-03 4.24238263e-06 0.00000000e+00]
```

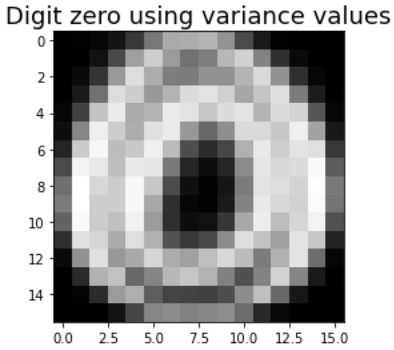
Βήμα 7: Σχεδιασμός ψηφίου χρησιμοποιώντας τις τιμές της μέσης τιμής

Στην συνέχεια σχεδιάζουμε το ψηφίο «0» χρησιμοποιώντας τις τιμές της μέσης τιμής που υπολογίσατε comστο Βήμα 6. Το αποτέλεσμα φαίνεται παρακάτω:



Βήμα 8: Σχεδιασμός ψηφίου χρησιμοποιώντας τις τιμές της διασποράς

Εδώ απεικονίζουμε το ψηφίο «0» χρησιμοποιώντας τις τιμές της διασποράς που υπολογίσαμε στο Βήμα 6:



Συμπεράσματα για τα βήματα 7,8:

- Είναι προφανές ότι και οι δύο εικόνες αναπαριστούν το ψηφίο μηδέν, αφού το σχήμα του ψηφίου διατηρείται. Αξίζει να αναφέρουμε όμως ότι η εικόνα της μέσης τιμής είναι πιο κοντά στο σχήμα του μηδενός από ότι αυτή της διασποράς.

- Παρατηρούμε ότι η απεικόνιση που προκύπτει από τη διασπορά είναι πιο διεσταλμένη, αλλά έχει παρόμοια μορφή με αυτή της μέσης τιμής.

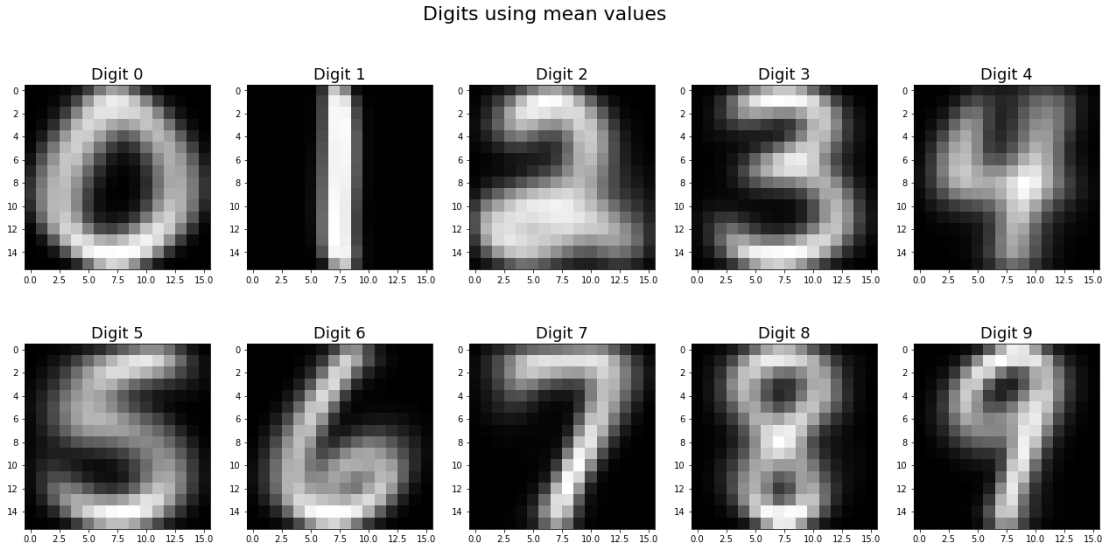
- Παρατηρούμε ότι τα pixel στο κέντρο του δακτυλίου που σχηματίζει το 0 καθώς και στις γωνίες της εικόνας έχουν μηδενική διασπορά, ενώ στα σημεία που αποτελούν το "μηδέν" η διασπορά είναι μικρότερη στο κέντρο του πάχους της γραμμής και μεγαλώνει όσο κινούμαστε προς τα όρια αυτής. Υψηλή μέση τιμή σημαίνει ότι είναι αρκετά συνηθισμένο από πολλά άτομα που σχεδιάζουν το ψηφίο μηδέν να συμπεριλάβουν το συγκεκριμένο pixel. Υψηλή διασπορά σημαίνει ότι κάποιοι ότι κάποια άτομα το επιλέγουν ενώ κάποια όχι. Έτσι, η κλιμάκωση της διασποράς από το κέντρο του πάχους προς τα όρια της γραμμής μπορεί να οφείλεται στο ότι κάθε άτομο μπορεί να χρησιμοποιεί διαφορετικό πάχος γραφίδας, και άρα ενώ το κεντρικό σημείο του πάχους της γραμμής σχεδιάζεται πάντα σχεδόν, τα σύνορα διαφέρουν. Οι διαφοροποιήσεις στην διασπορά μπορεί επίσης να οφείλονται στο ότι το κάθε άτομο δημιουργεί διαφορετική μορφολογία για τον αριθμό, κανένας δεν σχεδιάζει για παράδειγμα έναν τέλειο κύκλο. Αντίθετα, καθώς το μηδέν είναι σχετικά κυκλικό, οι γωνίες της εικόνας δεν χρωματίζονται από κανένα άτομο, γι' αυτό και πολύ χαμηλή μέση τιμή με μηδενική σχεδόν διασπορά.

- Υψηλότερη μέση τιμή έχουν η περιοχή προς τα πάνω και η περιοχή προς τα κάτω του μηδενικού, πράγμα λογικό αφού τα περισσότερα άτομα σχεδιάζουν το μηδέν ξεκινώντας τον κύκλο τους είτε από το πιο πάνω σημείο είτε από το πιο κάτω σημείο, και άρα τα σημεία αυτά σχεδιάζονται πάντα. Αντίθετα τα

σημεία στη μέση του ύψους μπορεί να μην σχεδιαστούν ανάλογα με το αν το μηδέν θα σχεδιαστεί κυκλικό ή οβάλ.

Βήμα 9: Υπολογισμός μέσης τιμής και διασποράς όλων των ψηφίων

Στο βήμα αυτό μελετήσαμε τη μέση τιμή και την διασπορά των χαρακτηριστικών για τα ψηφια (0-9) με βάση τα train δεδομένα, καλώντας τις συναρτήσεις digit_mean() και digit_variance() για όλα τα ψηφία.
Στην συνέχεια σχεδιάσαμε τα ψηφία χρησιμοποιώντας τις τιμές της μέσς τιμής που υπολογίσαμε. Το αποτέλεσμα είναι το εξής:



Ανάλογα συμπεράσματα που προέκυψαν από το ψηφίο μηδέν προκύπτουν και από τα άλλα ψηφία. Παρατηρούμε ότι το σύμβολο 1 έχει πολύ υψηλές τιμές μέσης τιμής. Αυτό είναι λογικό γιατί έχει τη πιο συνηθισμένη και απλή γραφή, άρα υπάρχουν μικρότερες διαφοροποιήσεις στα σημεία που χρωματίζουν οι χρήστες.

Βήμα 10: Ταξινόμηση ψηφίου των test δεδομένων με Euclidean Classifier

Στο βήμα αυτό θα ταξινομήσουμε το ψηφίο 101 από το test set δεδομένων σε μια από τις 10 κατηγορίες με βάση την Ευκλείδεια απόσταση μεταξύ κάθε pixel της εικόνας που μελετάμε με κάθε αντίστοιχο pixel από τις 10 εικόνες της μέσης τιμής των ψηφίων. Άρα για ένα δεδομένο x, όπου I είναι το i-οστο pixel ο ορισμός της Ευκλείδειας απόστασης είναι ο εξής:

$$(d_E)_j = \sqrt{\sum_{i=0}^{255} \left(x_i - y_j\right)^2} \ για\ κάθε\ j = \{0, \dots, 9\}$$

Η απόφαση του ευκλείδιου ταξινομητή είναι το

$$argmin\{((d_E)_j\}\ για\ κάθε\ j = \{0, \dots, 9\}$$

που ουσιαστικά είναι η ελαχιστοποίηση της L2 νόρμας.

Για το βήμα αυτό υλοποιήσαμε κατάλληλα τις συναρτήσεις euclidean_distance() και euclidean_distance_classifier() του αρχείου lib.py. Συγκεκριμένα, η πρώτη υπολογίζει απλά την Ευκλείδεια απόσταση δύο διανυσμάτων σύμφωνα με τον τύπο που παρουσιάστηκε νωρίτερα, ενώ η δεύτερη καλεί την πρώτη για να λάβει τελικά την απόφαση του ευκλείδιου ταξινομητή.

Τα αποτελέσματα είναι τα εξής:

```
Prediction class is: 0
Actual class is: 0
Prediction is correct.
```

Βήμα 11: Ταξινόμηση test set δεδομένων με βάση τον Euclidean Classifier

Στην συνέχεια ταξινομήσαμε όλα τα ψηφία των test δεδομένων σε μία από τις 10 κατηγορίες με βάση την Ευκλείδεια απόσταση. Προκειμένου να βρούμε το ποσοστό επιτυχίας χρησιμοποιήσαμε τη μετρική accuracy. Το accuracy ορίζεται ως εξής:

$$accuracy = \frac{Number\ of\ correct\ predictions}{Total\ size\ of\ X_{test}}$$

Το ποσοστό επιτυχίας είναι:

```
Success rate is: 0.8141504733432985
```

Παρατηρούμε ότι το ποσοστό επιτυχίας (81.42 %) είναι αρκετά υψηλό αν σκεφτούμε το πόσο απλός είναι ο ταξινομητής μας.

Βήμα 12: Υλοποίηση Euclidean Classifier σαν ένα scikit-learn estimator

Σε αυτό το στάδιο υλοποιήσαμε τον ταξινομητή Ευκλείδειας απόστασης απόστασης σαν ένα scikit-learn estimator, συμπληρώνοντας της συναρτήσεις fit(), predict(), score() της κλάσης EuclideanDistanceClassifier του αρχείου lib.py. Η λογική αυτών των συναρτήσεων είναι παρόμοια με τις συναρτήσεις που χρησιμοποιήσαμε στα βήματα 10,11. Για τον λόγο αυτό, προκειμένου να μην έχουμε επανάληψη κώδικα, οι συνάρτηση predict() καλεί την euclidean_distance_classifier(). Το ποσοστό επιτυχίας που προκύπτει είναι το ίδιο με πριν:

```
Success rate is: 0.8141504733432985
```

Βήμα 13: 5 -Fold Cross-Validation, Περιοχή Απόφασης, Learning Curve

α) 5 -Fold Cross-Validation

Στο ερώτημα αυτό ζητήθηκε ο υπολογισμός του score του Ευκλείδιου ταξινομητή με χρήση 5-fold cross-validation. Το k-fold cross-validation αποτελεί μία μέθοδο κατά την οποία τα δεδομένα χωρίζονται σε k ξένα set, και πραγματοποιείται εκπαίδευση k φορές. Κάθε φορά, ένα διαφορετικό από τα k set λαμβάνεται ως test set για τον έλεγχο της ακρίβειας της εκπαίδευσης και τα υπόλοιπα k-1 set χρησιμοποιούνται ως training set. Το τελικό score του ταξινομητή που προκύπτει λαμβάνεται ως το μέσο score των k επαναλήψεων. Στόχος της μεθόδου είναι η καλύτερη εκτίμηση του generalization error του μοντέλου μας, σε περιπτώσεις στις οποίες τα διαθέσιμα δεδομένα δεν είναι πάρα πολλά, και άρα ανακύπτει το πρόβλημα πως επιθυμούμε όσο μεγαλύτερο training set μπορούμε να έχουμε ώστε να εκπαιδεύσουμε καλύτερο μοντέλο, αλλά παράλληλα δεν μπορούμε να έχουμε πολύ μικρό test set καθώς τότε μπορεί να λάβουμε λανθασμένη εκτίμηση του generalization error. Έτσι, προκειμένου να μπορέσουμε να χρησιμοποιήσουμε όλα μας τα δεδομένα κάποια στιγμή στην εκπαίδευση, ώστε να λάβουμε το generalization error που προκύπτει όταν τα χρησιμοποιούμε, αλλά παράλληλα να μπορούμε να έχουμε πάντα ένα κατάλληλο μέρος αυτών για test set, χρησιμοποιούμε το k-fold cross-validation.

Στο παρόν ερώτημα, υλοποιήσαμε το k-fold cross-validation στην συνάρτηση evaluate_classifier() του αρχείου lib.py, η οποία καλεί την συνάρτηση cross_val_score() του πακέτου sklearn.model_selection. Εκτελέσαμε το 5-fold cross-validation στα δεδομένα X_train.

Για k=5 λαμβάνουμε τα επιμέρους score:

```
Scores with 5-fold cross-validation are: [0.84921179 0.85116598
0.84567901 0.84910837 0.84773663]
```

Άρα το μέσο score προκύπτει:

```
Mean score with 5-fold cross-validation is: 0.8485803550358166
```

β) Σχεδιασμός Περιοχής Απόφασης Ευκλείδειου Ταξινομητή

Στόχο του ερωτήματος αποτελεί ο σχεδιασμός της περιοχής απόφασης του Ευκλείδειου ταξινομητή. Αντιμετωπίζουμε όμως το εξής πρόβλημα: Καθώς για κάθε δείγμα έχουμε 256 χαρακτηριστικά, έχουμε στην πραγματικότητα ένα πρόβλημα 256 διαστάσεων και μια αντίστοιχη υπερ-επιφάνεια απόφασης. Κάτι τέτοιο όμως δεν είναι δυνατόν να οπτικοποιηθεί. Επομένως, πρέπει να μειώσουμε τις διαστάσεις σε 2 ή το πολύ τρεις, κρατώντας μόνο 2 ή 3 αντίστοιχα χαρακτηριστικά για κάθε ψηφίο. Ανακύπτει λοιπόν το ερώτημα, ποια είναι τα βέλτιστα εκείνα χαρακτηριστικά που πρέπει να επιλέξουμε, τα οποία θα περιέχουν όσο το δυνατόν περισσότερη πληροφορία που θα επιτρέπει την ταξινόμηση από τον Ευκλείδειο ταξινομητή. Για την επιλογή των κατάλληλων χαρακτηριστικών χρησιμοποιούμε μία μέθοδο γνωστή ως Principal Component Analysis (PCA).

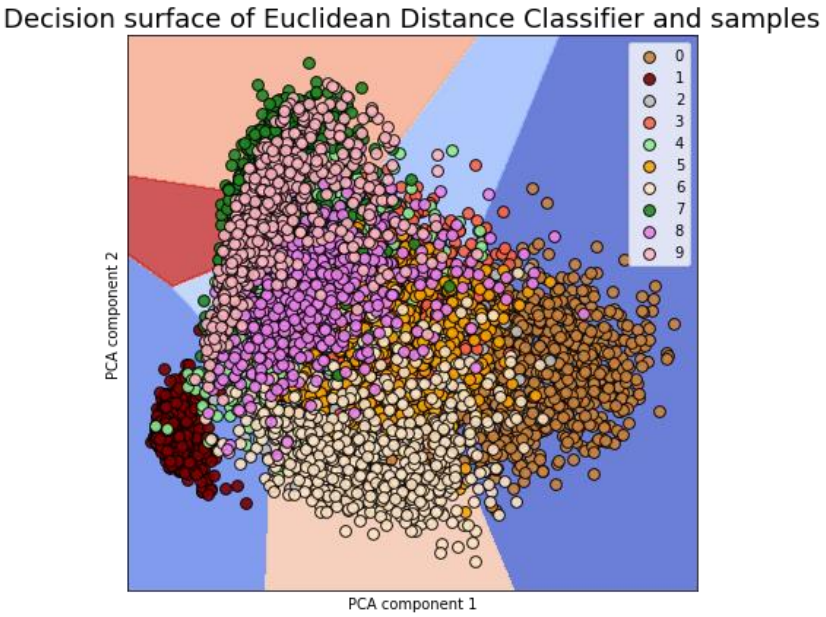
Η μέθοδος PCA αποτελεί μια συχνά χρησιμοποιούμενη μέθοδο για την μείωση της διαστατικότητας. Η μέθοδος αυτή υπολογίζει μία νέα ορθοκανονική βάση για τα δεδομένα, ώστε οι διαστάσεις των δεδομένων σε αυτή να είναι γραμμικά ασυσχέτιστες. Αυτό επιτυγχάνεται με τον υπολογισμό του πρώτου διανύσματος ως αυτό που μπορεί να προσεγγίσει καλύτερα τα δεδομένα υπό την έννοια της ελαχιστοποίησης του μέσου τετραγωνικού λάθους, στην συνέχεια του δεύτερου καλύτερου για τον σκοπό αυτό, το οποίο όμως είναι ορθογώνιο με το πρώτο, του τρίτου το οποίο είναι ορθογώνιο με τα άλλα δύο κοκ. Έτσι υπολογίζονται τελικά τα διανύσματα τα οποία ονομάζονται principal components και τα οποία είναι γραμμικοί συνδυασμοί των αρχικών μεταβλητών. Το χαρακτηριστικό των διανυσμάτων αυτών είναι πως η μέγιστη ποσότητα πληροφορίας βρίσκεται στα λίγα πρώτα διανύσματα. Έτσι, μπορούμε επιτυχώς να μειώσουμε την διαστατικότητα κρατώντας τις δύο ή τρεις πρώτες συνιστώσες με όσο το δυνατόν λιγότερη απώλεια πληροφορίας.

Στο παρόν πρόβλημα επιλέξαμε να κρατήσουμε τις δύο πρώτες συνιστώσες ώστε να έχουμε δισδιάστατη αναπαράσταση. Πραγματοποιήσαμε PCA με χρήση της υλοποίησης του πακέτου sklearn.decomposition. Όσον αφορά το ποσοστό πληροφορίας που περιέχουν οι δύο πρώτες συνιστώσες λαμβάνουμε:

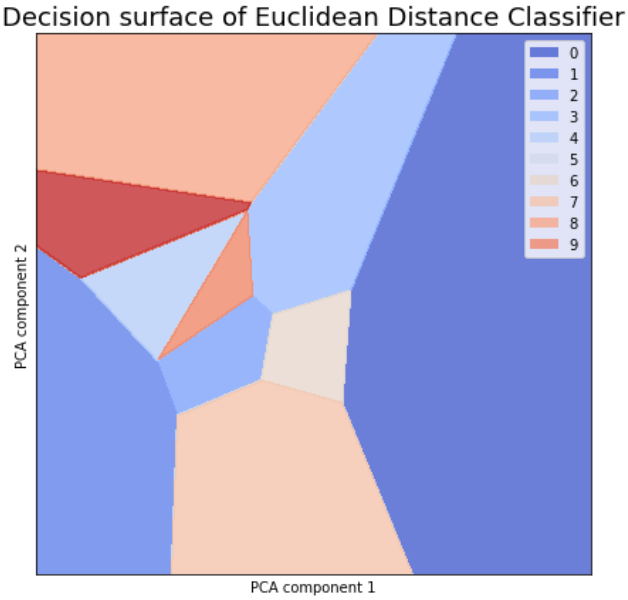
```
First component contains 17.88% of the total information.
Second component contains 8.97% of the total information.
The 2 components combined contain 26.85% of the total information.
```

Παρατηρούμε πως έχουμε καταφέρει να συμπεριλάβουμε το 26,85% της πληροφορίας. Αυτό είναι ένα αρκετά μεγάλο ποσοστό, αν αναλογιστεί κανείς πως έχουν 2 αντί για 256 διαστάσεις. (Κρατώντας δύο τυχαίες διαστάσεις θα είχαμε κατά μέσο όρο το 0,78% της πληροφορίας).

Ύστερα από τη εφαρμογή της μεθόδου PCA, έχουμε την παρακάτω γραφική παράσταση στην οποία αρχικά απεικονίζουμε σε ένα κοινό διάγραμμα το περιοχές απόφασης και την προβολή των δεδομένων στα δυο components . Παρατηρούμε ότι τα ψηφία 0, 1 και 6 διακρίνονται ευκολότερα από τα υπόλοιπα τα οποία βρίσκονται περίπου στην ίδια περιοχή και επομένως διαχωρίζονται πιο δύσκολα. Η γραφική παράσταση είναι η εξής:



Προκειμένου να είναι πιο ξεκάθαρος ο διαχωρισμός του χώρου σε περιοχές απόφασης απεικονίζουμε μόνο το διάγραμμα με τις περιοχές απόφασης. Στο παρακάτω διάγραμμα βλέπουμε ότι μερικές περιοχές είναι μεγαλύτερες από κάποιες άλλες επειδή στις μεγαλύτερες περιοχές πέφτουν πιο αραιά δείγματα και άρα εκεί η απόφαση είναι πιο σίγουρη. Επιπλέον, φαίνεται ότι κάποιες περιοχές είναι πολύ μικρές (π.χ οι περιοχές που αφορούν τα ψηφία 4,5, ή 9) πράγμα που υποδηλώνει ότι λόγω της μεθόδου PCA έχουμε σημαντική απώλεια πληροφορίας.



γ) Σχεδιασμός learning curve Ευκλείδειου Ταξινομητή

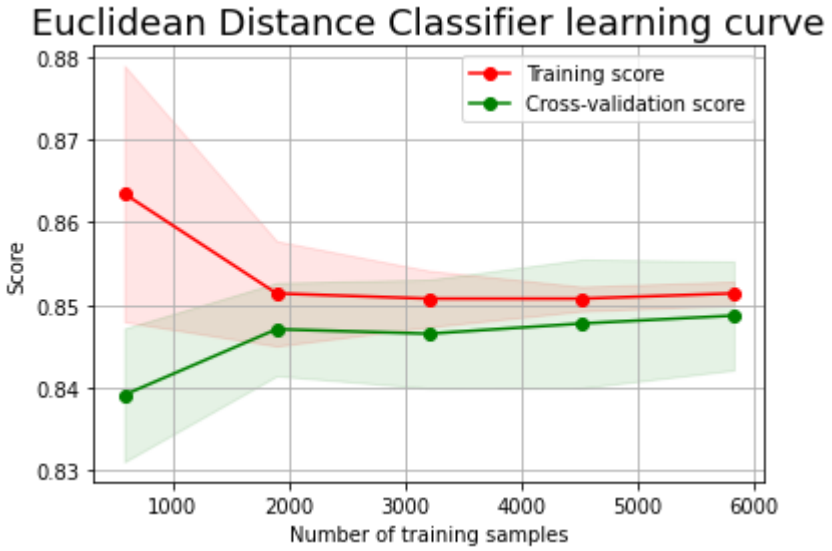
Στο παρόν ερώτημα ζητήθηκε ο σχεδιασμός της καμπύλης εκμάθησης του ευκλείδειου ταξινομητή. Για τον σκοπό αυτό χρησιμοποιήθηκε η συνάρτηση learning_curve() του πακέτου sklearn.model_selection. Η συνάρτηση αυτή πραγματοποιεί εκπαίδευση για διαφορετικά μεγέθη training set, ενώ για κάθε μέγεθος πραγματοποιεί k-fold cross validation, πραγματοποιεί δηλαδή k splits και εκπαιδεύει για κάθε split το μοντέλο μας, έχοντας training set με συγκεκριμένο μέγεθος. Για κάθε φορά που εκπαιδεύει το

μοντέλο υπολογίζει το training-score (στα train data) και το validation-score (στα test-data). Τελικά η συνάρτηση επιστρέφει τα μεγέθη training set για τα οποία πραγματοποίησε εκπαίδευση, και για κάθε μέγεθος τα training και validation score για κάθε fold. Σημειώνουμε ότι καλέσαμε την συνάρτηση θέτοντας την παράμετρο shuffle σε True. Αυτό εξασφαλίζει πως σε κάθε split τα training data θα λαμβάνονται με την απαραίτητη τυχαιότητα.

Τυπώνοντας τα μεγέθη που επιστρέφει η συνάρτηση λαμβάνουμε:

```
Train sizes are:
[ 583 1895 3207 4519 5832]
Train scores are:
[[0.87821612 0.84734134 0.86620926 0.84391081 0.88164666]
 [0.8474934  0.84116095 0.85382586 0.85540897 0.8591029 ]
 [0.84752105 0.84658559 0.85063923 0.85375741 0.8553165 ]
 [0.84797522 0.85107325 0.85107325 0.85240097 0.85129453]
 [0.84910837 0.85202332 0.85322359 0.85048011 0.85219479]]
Test scores are:
[[0.85332419 0.83744856 0.83470508 0.84087791 0.82921811]
 [0.85263879 0.84087791 0.84567901 0.85459534 0.84156379]
 [0.8553804  0.8388203  0.8436214  0.85322359 0.84156379]
 [0.854695   0.83950617 0.84773663 0.85802469 0.8388203 ]
 [0.8574366  0.84224966 0.8436214  0.85596708 0.84430727]]
```

Στην συνέχεια αναπαριστούμε τα training και cross-validation scores σε γραφική παράσταση, δημιουργούμε δηλαδή το ζητούμενο learning-curve. Για τον σκοπό αυτό υπολογίσαμε τα μέσα scores για κάθε train-size καθώς και την τυπική απόκλιση αυτών. (Η τελευταία θα αναπαριστά πόσο μεγάλες αποκλίσεις υπήρχαν στα scores μεταξύ των διαφορετικών folds του 5-fold cross validation). Προέκεψε τελικά η εξής γραφική παράσταση:



Από την παραπάνω γραφική παράσταση παρατηρούμε τα εξής:

- Το training score έχει την υψηλότερη τιμή για το μικρότερο μέγεθος training data ενώ το cross-validation score έχει την χαμηλότερη τιμή για αυτό το μέγεθος. Αυτό είναι λογικό, καθώς το training score υπολογίζεται πάνω στα

training data, και άρα όσο λιγότερα είναι αυτά, το μοντέλο μας μπορεί να "ταιριάξει καλύτερα" σε αυτά. Καθώς αυτά αυξάνονται, η δυνατότητά του να προβλέπει όλα τα train data μειώνεται. Αντίθετα, το training score αυξάνεται για αύξηση του πλήθους train data, το οποίο είναι λογικό, αφού έχοντας εκπαιδευτεί σε μεγαλύτερη ποικιλία δειγμάτων, το μοντέλο μας γενικεύει καλύτερα.

- Το validation score είναι αρκετά υψηλό, ενώ για μεγάλες τιμές training set size συγκλίνει με το training score. Αυτό αποτελεί μία ένδειξη πως δεν έχουμε υψηλό bias στο μοντέλο μας: Αν αντίθετα είχαμε πολύ καλό score στα train data και χαμηλό score στα test data, τότε αυτό θα αποτελούσε ένδειξη πως το μοντέλο μας είναι προσαρμοσμένο πολύ καλά στα δεδομένα εκπαίδευσης αλλά δεν γενικεύει σε διαφορετικά δείγματα εξίσου καλά, είναι δηλαδή biased.
- Η τυπική απόκλιση παραμένει σχεδόν σταθερή για τα διαφορετικά μεγέθη training set για το cross-validation score, το ίδιο και για το training score με εξαίρεση στο τελευταίο την πρώτη περίπτωση μεγέθους training score όπου είναι μεγαλύτερη (ακόμα όμως και σε αυτή την περίπτωση οι διαφοροποιήσεις δεν είναι πολύ μεγάλες: περίπου 3%). Αυτό σημαίνει πως τα διαφορετικά folds του 5-fold cross validation έχουν αντίστοιχες διαφοροποιήσεις του score ως αποτέλεσμα, ανεξάρτητα του μεγέθους του train size.
- Τόσο το training όσο και το cross-validation score, δεν φτάνουν σε καμία περίπτωση τιμές μεγαλύτερες του 86%, ενώ για μεγαλύτερα training set size συγκλίνουν περίπου στο 84.5%. Το score αυτό θα μπορούσε ενδεχομένως να βελτιωθεί ελαφρά μεγαλώνοντας λίγο περισσότερο το μέγεθος δεδομένων που διαθέτουμε, φαίνεται όμως πως δεν θα φτάναμε σε ακρίβεια πολύ μεγαλύτερη. Κάτι τέτοιο αποτελεί ένδειξη πως ενδεχομένως ο classifier που χρησιμοποιούμε δεν είναι βέλτιστος, δηλαδή είναι πολύ απλός για να πετύχει τα βέλτιστα αποτελέσματα στο παρόν task. Το πλήθος των χαρακτηριστικών που διαθέτουμε θα μπορούσε επίσης να αποτελεί έναν καθοριστικό παράγοντα: μεγαλύτερο πλήθος χαρακτηριστών θα μπορούσε και αυτό να διαφοροποιήσει το τελικό score.

Βήμα 14: Υπολογισμός A-Priori πιθανοτήτων

Προκειμένου να υπολογίσουμε τις a-priori πιθανότητες μετρήσαμε τη συχνότητα που εμφανίζεται κάθε ψηφίο στο training set. Ουσιαστικά αυτή ορίζεται ώς εξής:

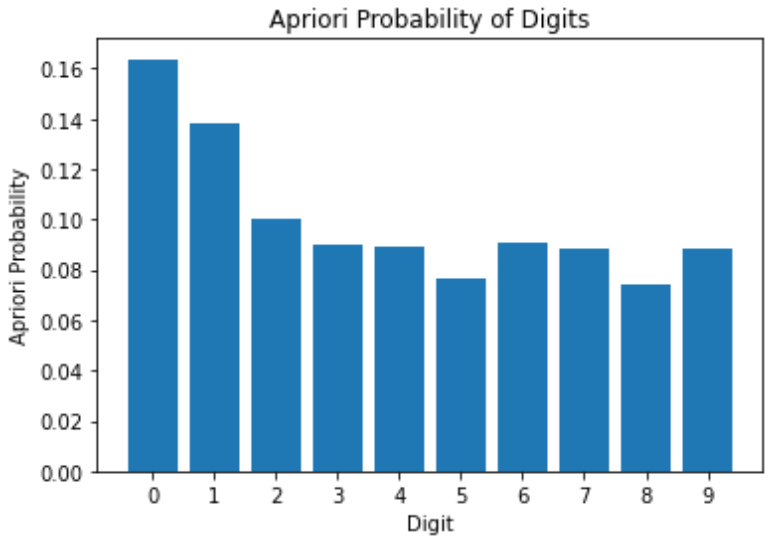
$$P(class = i) = \frac{\#examples\ of\ class\ i}{\#examples\ of\ all\ classes}$$

Έτσι συμπληρώνοντας κατάλληλα και τρέχοντας την συνάρτηση calculate_priors του αρχείου lib.py προέκυψαν τα εξής αποτελέσματα:

Class	A-priori Probability (%)
0	16.376
1	13.784
2	10.026
3	9.025
4	8.943

5	7.626
6	9.107
7	8.847
8	7.434
9	8.833

Τα παραπάνω αποτελέσματα απεικονίζονται στο παρακάτω διάγραμμα:



Παρατηρούμε ότι τα μηδενικά και οι άσσοι είναι πιο συχνά σε σχέση με τα υπόλοιπα ψηφία. Επομένως, φαίνεται πως στο σύνολο X_train το πλήθος δειγμάτων ανά ψηφίο διαφέρει αρκετά (π.χ. διπλάσια μηδενικά από πέντε). Κάτι τέτοιο μπορεί να έχει συνέπειες στην εκπαίδευση των ταξινομητών, οι οποίοι ενδέχεται να διαθέτουν καλύτερες γενικεύσεις για κάποια ψηφία και χειρότερες για άλλα.

Βήμα 15: Naive Bayes Classifier

α) Στόχος του ερωτήματος αυτού ήταν η υλοποίηση ενός Naive Bayes Classifier. Ο ταξινομητής αυτός κάνει μια απλοϊκή υπόθεση και θεωρεί ότι όλα τα pixels της εικόνας είναι πιθανοτικά ανεξάρτητα και ασυσχέτιστα μεταξύ τους, πράγμα που προφανώς δεν ισχύει. Ο Naive Bayes Classifier βασίζεται στον κανόνα του Bayes:

$$P(sample) = \frac{P(class) * P(class)}{P(sample)}$$

Από τη στιγμή που δεν μας ενδιαφέρει η πραγματική τιμή της παραπάνω πιθανότητας αλλά μας νοιάζει μόνο το να την διατάξουμε σε όλες τις κλάσεις, μπορούμε να αγνοήσουμε το παρονομαστή και έτσι καταλήγουμε στην εξής σχέσης:

$$P(sample) \propto P(class) * P(class)$$

Λόγω της υπόθεσης του Naive Bayes Ταξινομητή είναι:

$$P(sample) = P(class = 0) * \prod_{i=0}^{255} P(x_i|class = 0)$$

.
.
.

$$P(sample) = P(class = 9) * \prod_{i=0}^{255} P(x_i|class = 9)$$

Έτσι τελικά, όταν ταξινομούμε τα 256 χαρακτηριστικά (sample) μια νέα εικόνας που περιέχει ψηφίο, υπολογίζουμε τις 10 αυτές πιθανότητες που στηρίζονται στην υπόθεση αυτή. Η τελική πρόβλεψη θα είναι η κλάση εκείνη που έχει την μεγαλύτερη πιθανότητα:

$$result = argmax_{class}\{P(sample), \dots, P(sample)\}$$

Πιο συγκεκριμένα:

- Η πιθανότητα P(class) είναι η **apriori πιθανότητα** κάθε κλάσης που υπολογίσαμε στο βήμα 14.

- Όσον αφορά την P(sample|class), θα λάβουμε υπόψη τις τιμές της μέσης τιμής και διασποράς που υπολογίσατε στο Βήμα 9(α). Συγκεκριμένα στο Βήμα 9(α) υπολογίσαμε για κάθε κλάση την μέση τιμή και την διασπορά κάθε feature. Υποθέτουμε ότι για κάθε κλάση κάθε feature (pixel) ακολουθεί μέση τιμή και διασπορά αυτή που υπολογίσαμε μόλις, ή ισοδύναμα για κάθε κλάση όλα τα features ακολουθούν μια κοινή πολυδιάστατη κανονική κατανομή με πίνακα μέσων τιμών τις μέσες τιμές του κάθε feature και πίνακα συνδιακύμανσης έναν διαγώνιο πίνακα με τιμές στην διαγώνιο τις διασπορές που υπολογίσαμε μόλις.

Αξίζει να αναφέρουμε ότι στην υλοποίηση μας για λόγους μαθηματικής ευστάθειας προσθέσαμε λίγο θόρυβο στην διαγώνιο του πίνακα συνδιακύμανσης (σε ορισμένα σημεία η διασπορά προκύπτει 0 και άρα δεν μπορεί να υπολογιστεί ο αντίστροφος του πίνακα) και επιπλέον υπολογίσαμε τον λογάριθμο των πιθανοτήτων (τα γινόμενα πιθανοτήτων μπορούν να γίνουν πολύ μικρά και άρα να έχουμε θέματα υπερχείλισης). Προκειμένου να βρούμε τη καλύτερη τιμή της παραμέτρου για το smoothing πραγματοποιήσαμε tuning της υπερπαραμέτρου smoothing για να μεγιστοποιήσουμε την απόδοση, με χρήση της συνάρτησης GridSearchCV του πακέτου sklearn.model_selection. Τελικά καταλήξαμε στην τιμή 0.0008 για την παράμετρο smoothing του ταξινομητή που υλοποιήσαμε.

Με βάση τα παραπάνω έγινε αρχικά η ταξινόμηση όλων των ψηφίων των test δεδομένων ως προς τις 10 κατηγορίες. Λάβαμε ενδεικτικά τις προβλεψεις:

```
[9 2 3 ... 4 0 1]
```

β) Υπολογίζοντας το score (accuracy) του ταξινομητή μας προέκυψε:

```
Custom Naive Bayes Classifier has an accuracy of:  0.8126557050323866
```

και σε 5-fold cross -validation:

```
Scores with 5-fold cross-validation are: [0.82179575 0.81618656
0.80589849 0.7654321  0.79835391]
Custom Naive Bayes Classifier with 5 cross-validation has a mean
accuracy of:  0.8015333613510954
```

γ) Στο ερώτημα αυτό συγκρίναμε την υλοποιήσή μας με έναν έτοιμο ταξινομητή της βιβλιοθήκης scikit-learn. Σημειώνουμε ότι πραγματοποιήσαμε ξανά tuning για την εύρεση της βέλτιστης τιμής της υπερπαραμέτρου smoothing και αυτή προέκυψε ίση με 0.0008. Τα αποτελέσματα του έτοιμου ταξινομητή είναι:

```
Scikit-learn Naive Bayes Classifier has an accuracy of:
0.7713004484304933
```

και σε 5-fold cross-validation:

```
Scores with 5-fold cross-validation are: [0.80397533 0.79355281
0.77572016 0.74005487 0.77846365]
Scikit-learn Naive Bayes Classifier with 5 cross validation has
an accuracy of:  0.7783533641528717
```

Παρατηρούμε ότι ο δικός μας ταξινομητής πετυχαίνει ελαφρώς καλύτερο accuracy σε σχέση με αυτόν της βιβλιοθήκης scikit-learn. Η διαφοροποίηση που παρατηρείται στην απόδοση των δύο ταξινομητών θα μπορούσε να είναι ενδεικτική ελαφρά καλύτερης υλοποίησης, είναι όμως αρκετά πιθανό να είναι τυχαία για το συγκεκριμένο test set και να μην παρατηρούταν με χρήση διαφορετικού train ή test set.

Βήμα 16: Naive Bayes Classifier με μοναδιαία διασπορά για όλα τα χαρακτηριστικά

Στο βήμα αυτό επαναλάβαμε το Βήμα 15 (α), (β) υποθέτοντας ότι η διασπορά για όλα τα χαρακτηριστικά, για όλες τις κατηγορίες ισούται με 1. Κάτι τέτοιο επιτυγχάνει καλύτερα αποτελέσματα σε σχέση με πριν. Αυτό μας οδηγεί στο συμπέρασμα πως η πληροφορία των διασπορών των χαρακτηριστικών δεν αποτελεί σημαντική πληροφορία για το δοθέν dataset. Τα 256 χαρακτηριστικά κάθε ψηφίου μπορούν να προκαλέσουν μια μικρή διακύμανση και έτσι η Γκαουσιανή κατανομή δεν έχει μεγάλο σ και άρα είναι σαν ένα spike. Αυτό οδηγεί σε πολλές δυσκολίες για το μοντέλο κατά την πρόβλεψη πολλών χαρακτηριστικών χαμηλών διακυμάνσεων. Άρα για αυτό με μια τυπική διασπορά 1 πετυχαίνουμε υψηλότερη ακρίβεια. Επίσης τονίζουμε ότι για να βρούμε τη καλύτερη τιμή της παραμέτρου για το smoothing πραγματοποιήσαμε ξανά tuning της υπερπαραμέτρου και η καλύτερη τιμή που προέκυψε είναι η 0.00001.

Τα αποτελέσματα που λάβαμε υπολογίζοντας το score (accuracy) του ταξινομητή είναι τα εξής:

```
Naive Bayes Classifier with unit variances has an accuracy of:
0.8126557050323866
```

και με χρήση 5-fold cross-validation:

```
Scores with 5-fold cross-validation are: [0.84989719 0.85048011
0.84705075 0.84979424 0.84430727]
Naive Bayes Classifier with unit variances has an  5-fold accuracy
of:  0.848305912593984
```

Βήμα 17: Σύγκριση Naïve Bayes, Nearest Neighbors, SVM

Στο παρόν ερώτημα ζητήθηκε η σύγκριση της επίδοσης διαφορετικών ταξινομητών. Συγκεκριμένα, δημιουργήσαμε έναν ταξινομητή Nearest Neighbors (KNN), τρεις ταξινομητές SVM με διαφορετικούς πυρήνες ο καθένας (linear, rbf, polynomial), καθώς και τρεις Naive Bayes ταξινομητές, δύο με βάση την δική μας υλοποίηση (ο ένας με μοναδιαία variances) και έναν με βάση την υλοποίηση του scikit-learn.

Για καθένα από τους παραπάνω ταξινομητές χρησιμοποιήσαμε υπερπαραμέτρους τις οποίες προσδιορίσαμε με tuning, με χρήση της μεθόδου GridSearchCV του πακέτου sklearn.model_selection. (Για τους Naive Bayes ταξινομητές το tuning είχε ήδη πραγματοποιηθεί σε προηγούμενο ερώτημα). Οι ακριβείς υπερπαράμετροι ανάμεσα στις οποίες επιλέξαμε τις βέλτιστες φαίνονται στον κώδικά μας. Στην συνέχεια παρατίθενται οι υπερπαράμετροι των συναρτήσεων του scikit-learn που χρησιμοποιήθηκαν μετά το tuning για κάθε ταξινομητή:

- Linear SVM Classifier:
  C: 0.01, gamma: scale, kernel: linear
- RBF SVM Classifier:
  C: 10, gamma: scale, kernel: rbf
- Polynomial SVM Classifier:
  C:10, gamma: auto, kernel: poly
- Nearest Neighbors Classifier:
  algorithm: auto, n_neighbors:2, weights: distance
- Custom Naive Bayes Classifier:
  smoothing: 0.0008
- Scikit-learn Naive Bayes Classifier:
  var_smoothing: 0.0008
- Custom Naive Bayes Classifier with unit variances:
  smoothing: 0.00001

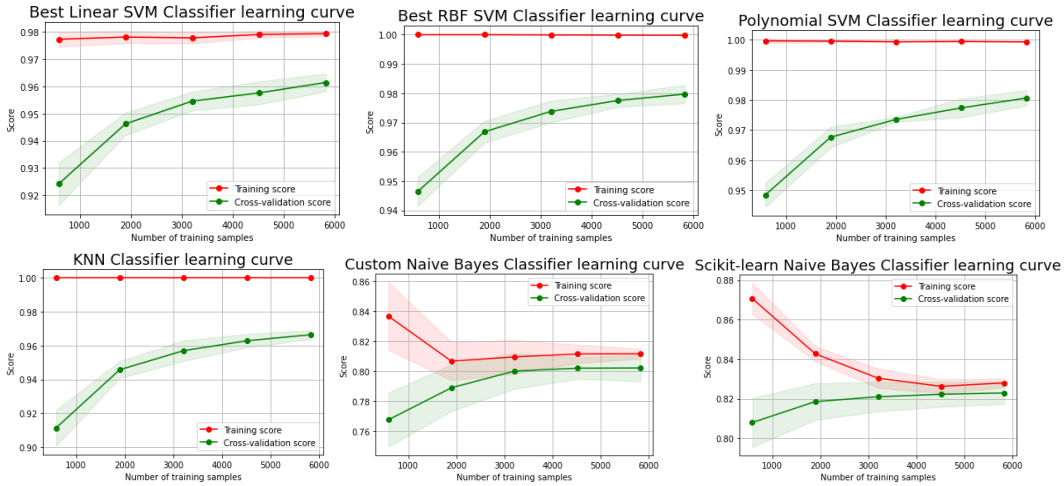
Αφού λοιπόν για κάθε ταξινομητή πραγματοποιήθηκε ο προσδιορισμός των υπερπαραμέτρων, ακολούθησε αξιολόγηση του ταξινομητή με χρήση 5-fold cross-validation πάνω στα δεδομένα X_train. Η αξιολόγηση αυτή πραγματοποιήθηκε με την κλήση της κατάλληλης συνάρτησης evaluate_*name*_classifier όπου *name* ο τύπος του

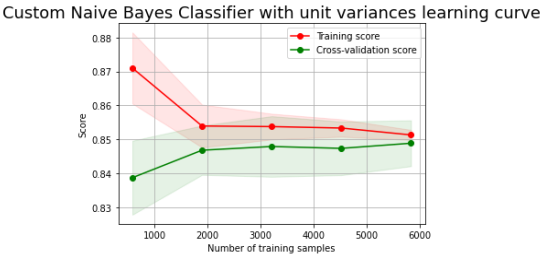
ταξινομητή, του αρχείου lib.py. Οι συναρτήσεις αυτές καλούν όλες την συνάρτηση βάσης evaluate_classifier() του ίδιου αρχείου, η οποία και πραγματοποιεί το 5-fold cross-validation με χρήση της βιβλιοθήκης scikit-learn. Τα αποτελέσματα που προέκυψαν παρουσιάζονται στον πίνακα που ακολουθεί:

Classifier type	Mean score with 5-fold cross-validation (%)
Linear SVM	96.15
RBF SVM	98.08
Polynomial SVM	98.08
Nearest Neighbors	96.74
Custom Naive Bayes	80.15
Scikit-learn Naive Bayes	77.84
Custom Naive Bayes with unit variances	84.83

Παρατηρούμε ότι όλοι οι SVM ταξινομητές καθώς και ο KNN ταξινομητής έχουν υψηλότερες επιδόσεις σε σχέση με τους Naive Bayes ταξινομητές, με την βέλτιστη επίδοση να επιτυγχάνεται από τους RBF και Polynomial SVM ταξινομητές. Ένας λόγος που ο linear πυρήνας δεν επιτυγχάνει εξίσου καλές επιδόσεις με τους άλλους δύο πυρήνες για SVM ταξινομητή είναι πως ο linear πυρήνας είναι κυρίως κατάλληλος για γραμμικά διαχωρίσιμα dataset. Έτσι, αν τα δεδομένα μας δεν έχουν αυτή την ιδιότητα, είναι λογικό να μην έχουμε το ίδιο καλό αποτέλεσμα. Όσον αφορά δε την αισθητά χαμηλότερη επίδοση των Naive Bayes ταξινομητών, μπορούμε να εξηγήσουμε αυτή με βάση το γεγονός πως πρόκειται για πιο απλούς ταξινομητές, που κάνουν την υπόθεση γκαουσιανής κατανομής για τα χαρακτηριστικά και μη ύπαρξης συσχέτισης μεταξύ τους. Έτσι, προκύπτουν σφάλματα σε σχέση με την πραγματική κατανομή των χαρακτηριστικών και άρα δεν μπορεί να κωδικοποιηθεί σωστά όλη η πληροφορία για τα ψηφία, ώστε να κατηγοριοποιούνται κατάλληλα σε όλες τις περιπτώσεις.

Στην συνέχεια παρουσιάζουμε τα learning curves των ταξινομητών, προς σύγκριση:





Παρατηρούμε ότι το cross-validation score (score στα test data) έχει περίπου ίδια μορφή σε όλες τις περιπτώσεις. Αντίθετα, στους ταξινομητές SVM και KNN, η καμπύλη του training score είναι σχεδόν σταθερή, ενώ στους Naive Bayes είναι πολύ πιο έντονα φθίνουσα. Η διαφοροποίηση αυτή μπορεί να εξηγηθεί από το γεγονός πως οι SVM και KNN έχουν καλύτερες επιδόσεις και μπορούν να μοντελοποιήσουν καλύτερα τα δείγματα, οπότε και να κατηγοριοποιούν σωστά δείγματα που έχουν ήδη δει. Αντίθετα, στους Naive Bayes, καθώς αυξάνεται το πλήθος των δειγμάτων εκπαίδευσης, αρχίζουν να παρουσιάζονται σφάλματα στην μοντελοποίηση των κατανομών τους και δεν είναι πια εφικτό να κωδικοποιηθεί όλη η απαραίτητη πληροφορία για να κατηγοριοποιούνται όλα σωστά.

Βήμα 18: Voting και Bagging Classifier

α) Voting Classifier

Στόχο του παρόντος ερωτήματος αποτελούσε η εφαρμογή της τεχνικής ensembling, ο συνδυασμός δηλαδή επιμέρους ταξινομητών οι οποίοι έχουν ήδη υψηλές επιδόσεις, με στόχο την επίτευξη ακόμα υψηλότερης επίδοσης. Για τον σκοπό αυτό χρησιμοποιήσαμε η τεχνική του voting.
Συγκεκριμένα, χρησιμοποιήσαμε έναν VotingClassifier() του πακέτου sklearn.ensemble. Ο ταξινομητής λαμβάνει μία λίστα από ταξινομητές και πραγματοποιεί είτε hard είτε soft voting. Στην πρώτη περίπτωση, υπολογίζεται για κάθε δείγμα που πρέπει να ταξινομηθεί η πρόβλεψη καθενός από τους επιμέρους ταξινομητές και επιλέγεται τελικά η κλάση “με τους περισσότερους ψήφους” αυτή δηλαδή στην οποία υπήρξαν οι περισσότερες προβλέψεις ότι ανήκει το δείγμα. Στην δεύτερη περίπτωση, κάθε ταξινομητής δεν δίνει μία απλή πρόβλεψη κλάσης, αλλά απαντά με βάση το πόσο πιθανό είναι ένα δείγμα να ανήκει σε μία κλάση. Έτσι, ο Voting ταξινομητής υπολογίζει στην συνέχεια το άθροισμα των πιθανοτήτων για κάθε κλάση και επιλέγει την κλάση με την μέγιστη πιθανότητα.

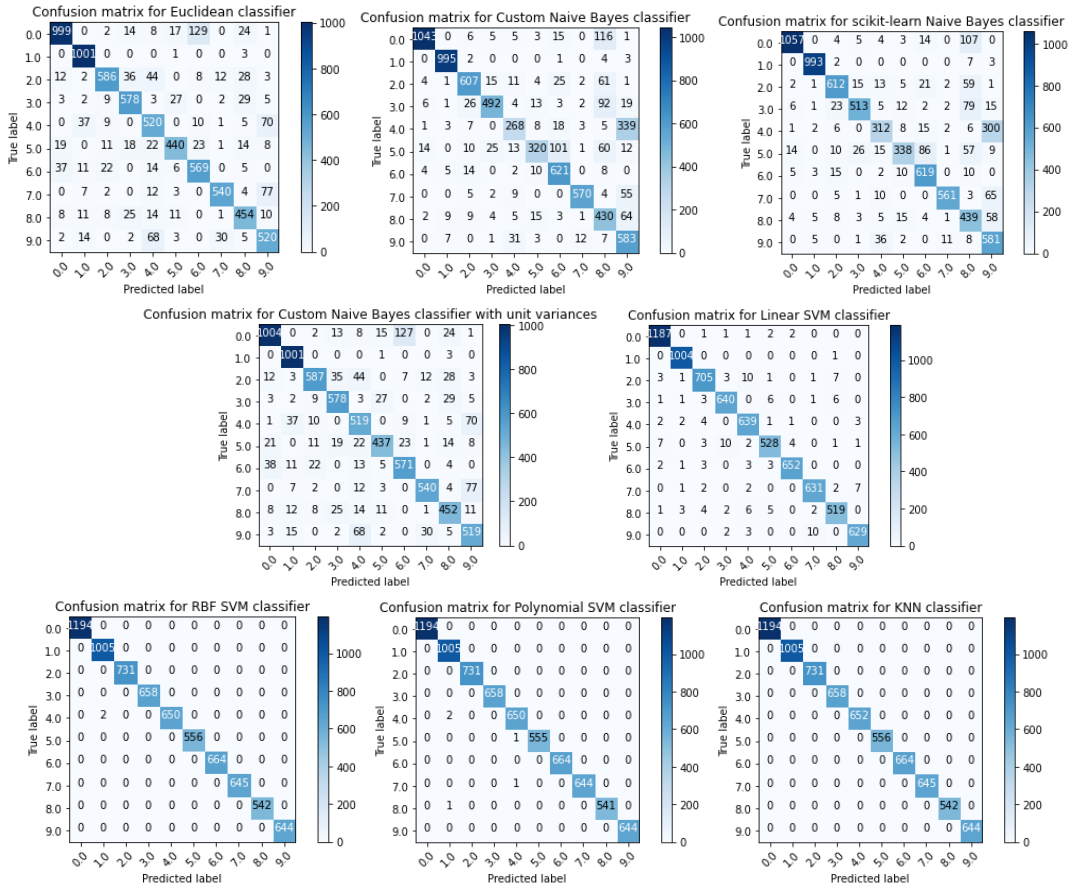
Σχετικά με την επιλογή των επιμέρους ταξινομητών που θα χρησιμοποιούνται από τον Voting Classfier αναφέρουμε τα εξής:

- Αρχικά, είναι σημαντικό να επιλεγεί περιττός αριθμός ταξινομητών. Κάτι τέτοιο εξασφαλίζει πως δεν θα υπάρχουν ισοπαλίες στην περίπτωση του hard voting, και άρα ο Voting ταξινομητής θα μπορεί να επιλέξει μία κλάση.

- Επιπλέον, προκειμένου να βελτιωθεί η επίδοση, είναι πολλές φορές χρήσιμο οι ταξινομητές που χρησιμοποιούνται να μην έχουν την τάση να κάνουν σφάλμα στις ίδιες περιπτώσεις. Έτσι, με χρήση της πλειοψηφίας σε περιπτώσεις που

ένας ταξινομητής κάνει σφάλμα, θα επιλέγεται η κλάση που επέλεξαν οι περισσότεροι ταξινομητές, οι οποίοι αν δεν κάνουν τα ίδια σφάλματα θα επιλέγουν πιο συχνά την σωστή κλάση.

Προκειμένου να λάβουμε μία εικόνα των περιπτώσεων που κάθε ταξινομητής κάνει σφάλμα υλοποιήσαμε confusion matrices για τους ταξινομητές που συγκρίναμε στο προηγούμενο ερώτημα, καθώς και για τον Ευκλείδειο ταξινομητή. Σημειώνουμε ότι χρησιμοποιούμε τα δεδομένα train για την δημιουργία των πινάκων αυτών, καθώς θεωρούμε πως τα δεδομένα test είναι διαθέσιμα μόνο για τελική αξιολόγηση των μοντέλων μας.



Με βάση τα παραπάνω, επιλέξαμε τελικά την υλοποίηση ενός Voting Classifier με χρήση των τριών ταξινομητών:

- Scikit-learn Naive Bayes,
- KNN,
- Polynomial SVM.

Τον ταξινομητή αυτό υλοποιήσαμε με χρήση hard και με χρήση soft voting. Επιπλέον, χάριν σύγκρισης υλοποιήσαμε επιπλέον έναν Voting Classifier με χρήση τριών ταξινομητών:

- Linear SVM,
- RBF SVM,
- Polynomial SVM.

Τα αποτελέσματα που λάβαμε βρίσκονται στον παρακάτω πίνακα:

Voting Classifier type	Score on test data	Mean score with 5-fold cross-validation (%)
Hard voting (Bayes,SVM,KNN)	94.67	92.13
Soft voting (Bayes,SVM,KNN)	94.47	91.23
Hard Voting (SVM)	95.17	93.27
Soft Voting (SVM)	95.17	93.32

Συμπεράσματα Voting Classifier

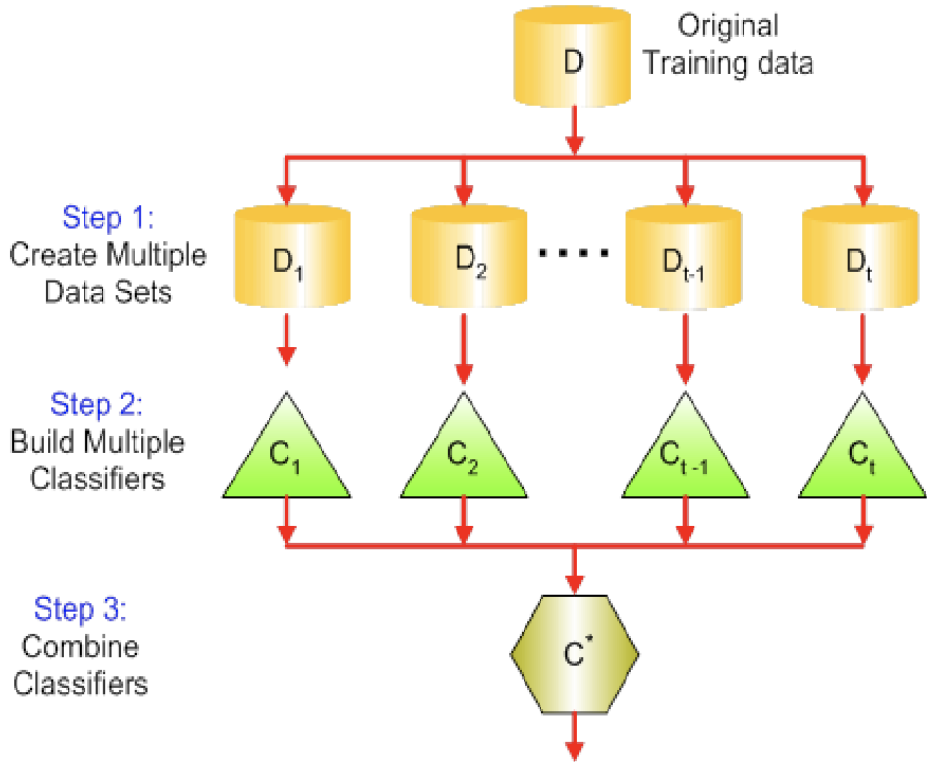
Παρατηρούμε τα εξής:

- Αν και θα αναμέναμε ο Voting ταξινομητής που συνδυάζει διαφορετικούς τύπους ταξινομητών να έχει ελαφρώς καλύτερα αποτελέσματα, παρατηρούμε πως ο συνδυασμός των τριών SVM πετυχαίνει τελικά καλύτερο score. Αυτό θα μπορούσε να οφείλεται στο γεγονός πως οι τρεις SVM ταξινομητές έχουν πολύ υψηλά ποσοστά ακρίβειας (όπως παρουσιάσαμε στο προηγούμενο ερώτημα) και άρα, ακόμα και αν πρόκειται για ίδιο τύπο ταξινομητή με παρόμοια σφάλματα το τελικό αποτέλεσμα εξακολουθεί να είναι πολύ καλό, ενώ αντίθετα, χρησιμοποιώντας τον Naive Bayes ως ταξινομητή ο οποίος κάνει πολύ περισσότερα σφάλματα αυτός επηρεάζει τελικά αρνητικά την τελική επίδοση ψηφίζοντας λάθος σε περιπτώσεις που άλλος ένας ταξινομητής μπορεί να ψηφίσει λάθος στην περίπτωση του hard voting ή ενισχύοντας τη πιθανότητα για μια λάθος ταξινόμηση στην περίπτωση του soft voting. (Τα σφάλματα στην κατηγοριοποίηση μπορεί πολλές φορές να μην οφείλονται μόνο στην αδυναμία του ταξινομητή να κωδικοποιήσει όλη την απαραίτητη πληροφορία αλλά στο ότι ένα δείγμα αποκλίνει περισσότερο από τα δείγματα που έχει “δει” ο ταξινομητής κατά την εκπαίδευση. Σε τέτοιες περιπτώσεις μπορεί να συμπίπτουν λάθη ταξινομητών διαφορετικών κατηγοριών). Και στις δύο περιπτώσεις επιμέρους ταξινομητών η επίδοση είναι πάντως αρκετά υψηλή.

- Οι τεχνικές hard και soft voting έχουν παρόμοια αποτελέσματα, και δεν διακρίνεται μία ως αισθητά καλύτερη της άλλης.

β) Bagging Classifier

Στόχος του ερωτήματος αυτού ήταν η επιλογή ενός ταξινομητή από τα προηγούμενα βήματα και η χρήση του Bagging Classifier για τη δημιουργία ενός ensemble. Ο Bagging Classifier βασιζόμενος σε ένα είδος ταξινομητή δημιουργεί πολλά υποσύνολα των training δεδομένων, τα οποία δεν είναι απαραίτητα ανεξάρτητα μεταξύ τους και για καθένα από αυτά τα υποσύνολα εκπαιδεύει έναν ταξινομητή. Η τελική απόφαση προκύπτει είτε ως ψηφοφορία των διάφορων ταξινομητών που εκπαιδεύτηκαν είτε ως μέσο όρος των προβλέψεων. Γενικά, η χρήση ενός Bagging classifier μειώνει το overfiting και το bias, καθώς κανένας από τους επιμέρους ταξινομητές δεν βλέπει το σύνολο των training δεδομένων και άρα ο καθένας εκπαιδεύεται διαφορετικά. Σχηματικά η λειτουργία του Bagging Classifier είναι η εξής:



Αν σε αυτή την μέθοδο χρησιμοποιήσουμε για ταξινομητή Decision trees ο ταξινομητής που προκύπτει ονομάζεται Random Forest. Στην υλοποίησή μας επιλέξαμε αρχικά τον ταξινομητή SVM σαν βάση του Bagging Classifier, αφού αυτός είχε πετύχει μέχρι στιγμής την καλύτερη απόδοση, ενώ στην συνέχεια χρησιμοποιήσαμε Decision trees. Τα αποτελέσματα που προέκυψαν είναι τα εξής:

Bagging Classifier type	Score on test data	Mean score with 5-fold cross-validation (%)
SVM	94.97	93.57
Decision Tree	88.59	83.56

Συμπεράσματα Bagging Classifier

Από τα παραπάνω μπορούμε να βγάλουμε τα εξής συμπεράσματα:

- Η χρήση SVM ταξινομητών για τον Bagging classifier οδηγεί σε καλύτερα αποτελέσματα σε σχέση με την χρήση Decision Trees για το συγκεκριμένο train και test set.
- Η ακρίβεια που επιτυγχάνει ο Bagging Classifier είναι μεν αρκετά υψηλή, ωστόσο παρατηρούμε πως δεν είναι όσο υψηλή ήταν η ακρίβεια ενός SVM ταξινομητή, σε αντίθεση ίσως με το αποτέλεσμα που θα αναμέναμε, δεδομένου ότι ο Bagging classifier μειώνει το bias. Κάτι τέτοιο μπορεί να οφείλεται στο γεγονός πως στην περίπτωση του Bagging Classifier κανένας από τους επιμέρους ταξινομητές δεν βλέπει ολόκληρο το train set και εκπαιδεύεται σε

λιγότερα δεδομένα. Κάτι τέτοιο φαίνεται πως έχει αρνητικές επιπτώσεις στην συγκεκριμένη περίπτωση, καθώς το σύνολο των ταξινομητών τελικά δεν αποφασίζει σωστά όσο συχνά όσο ένας μόνο ταξινομητής ο οποίος εκπαιδεύτηκε από όλα τα δεδομένα.

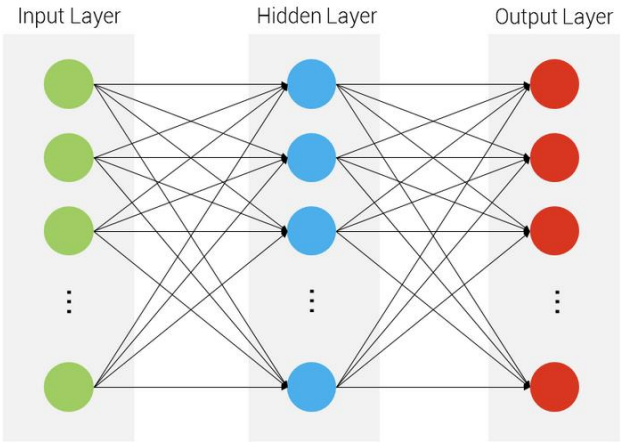
γ) Σύγκριση Votting Classifier και Bagging Classifier

Σε συνδυασμό με τα συμπεράσματα που εξήγαμε στα ερωτήματα (α) και (β), συγκρίνοντας τους ταξινομητές Bagging και Voting που υλοποιήσαμε μπορούμε να παρατηρήσουμε πως έχουν αντίστοιχες επιδόσεις (στην περίπτωση χρήσης SVM ταξινομητών στον Voting και στον Bagging ταξινομητή).

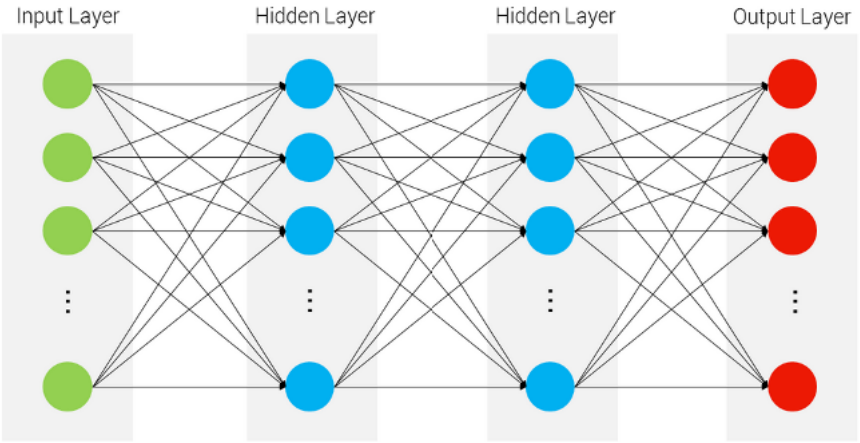
Βήμα 19: Κατασκευή Νευρωνικού Δικτύου

α) Σε αυτό το βήμα κάναμε μια εισαγωγή στα νευρωνικά δίκτυα και στη βιβλιοθήκη PyTorch. Αρχικά υλοποιήσαμε τις διαδικασίες φόρτωσης των training και testing δεδομένων στο νευρωνικά. Αυτό έγινε μετά από μετατροπή των δεδομένων σε tensors, μια δομή του pyTorch που μοιάζει πολύ με του numpy πίνακες. Έπειτα καλέσαμε τον DataLoader για κάθε είδος δεδομένων.
Επίσης σε αυτό το σημείο ορίζουμε το batch size που είναι μια υπερπαράμετρος που δηλώνει το πλήθος των δεδομένων που θα χρησιμοποιηθούν από το δίκτυο σε κάθε βήμα. Ο λόγος που χρησιμοποιούμε batches και δεν χρησιμοποιούμε όλο το dataset κατευθείαν είναι επειδή τα batches χρησιμοποιούν λιγότερη μνήμη και επιταχύνεται η διαδικασία της εκπαίδευσης. Η παράμετρος batch size συνήθως παίρνει δυνάμεις του 2, αφού άλλωστε και όταν αναφερόμαστε σε μέγεθος μνήμης έχουμε την τάση να χρησιμοποιούμε δυνάμεις του δύο. Από την άλλοη πλευρά όσο πιο μικρό είναι το batch τόσο λιγότερο accurate είναι η gradient. Γενικότερα, αξίζει να σημειωθεί ότι όσο αυξάνεται το batch size ο αλγόριθμος βρίσκει ελάχιστο που όμως είναι αρκετά δεν είναι "ομαλά". Αυτό οδηγεί σε πολύ μεγάλη υποβάθμιση του μοντέλου μας έχοντας ως μονάδα μέτρησης το πόσο καλά κάνει generalize σε νεα δεδομένα . Ακόμα, μεγαλύτερο batch size οδηγεί σε μεγαλύτερες ανάγκες σε πόρους, χωρίς όμως να υπάρχει σημαντική βελτίωση στην αβεβαιότητα του gradient. Συνεπώς αφήσαμε το batch size ως έχει σε 32 που είναι μια καλή default τιμή.

β) Στο ερώτημα αυτό υλοποιήσαμε διάφορα fully connected νευρωνικό δίκτυο σε PyTorch σαν μια υποκλάση της nn.Module. Τονίζεται ότι όλα τα νευρωνικά που υλοποιήσαμε έχουν σαν είσοδο 256, όσα και τα pixels των εικόνων στο dataset, και έξοδο 10, όσο και τα ψηφία που καλούμαστε να προβλέψουμε. Αρχικά ορίσαμε ένα 2-layer fully connected νευρωνικό δίκτυο το οποίο έχει τη παρακάτω δομή:



και ένα 3-layer fully connected νευρωνικό δίκτυο το οποίο έχει την παρακάτω δομή;



- Το 2-layer fully connected νευρωνικό δίκτυο έχει ένα κρυφό επίπεδο. Το πλήθος των κόμβων ελέγχεται από το χρήστη. Εμείς χρησιμοποιήσαμε τις τιμές 32, 128. Το 3-layer fully connected νευρωνικό δίκτυο έχει ένα κρυφό επίπεδο έχει 2 κρυφά επίπεδα. Το πλήθος των κόμβων σε κάθε επίπεδο επιλέχθηκε να είναι 128,64 αντίστοιχα και 64,32 αντίστοιχα.
- Όσον αφορά την συνάρτηση ενεργοποίησης δοκιμάσαμε τόσο την ReLU όσο και την Sigmoid. Παρατηρήσαμε ότι η ReLU οδήγησε σε καλύτερο αποτέλεσμα στο δεδομένο testing set από ότι η Sigmoid. Συγκεκριμένα τα αποτελέσματα που προέκυψαν ήταν:

Neural Network	Hidden Layer nodes	Activation Function	Accuracy (%)
2-layer	128	ReLU	93.47
2-layer	32	ReLU	93.33
2-layer	128	Sigmoid	93.17
2-layer	32	Sigmoid	93.32
3-layer	128,64	ReLU	93.87
3-layer	64,32	ReLU	93.32
3-layer	128,64	Sigmoid	93.27
3-layer	64,32	Sigmoid	92.76

- Άλλες υπερπαράμετροι που χρησιμοποιήθηκαν είναι:
    - *το πλήθος των εποχών* το οποίο ορίστηκε ίσο με 40,
    - *τα learning rates* που ορίστηκαν ίσα με 0.01,
    - *το batch size* που ορίστηκε ίσο με 32,
    - *ο optimizer* και τα χαρακτηριστικά του (επιλέξαμε τον SGD),
    - *το criterion* (επιλέχθηκε το cross entropy loss)
- Παρατηρούμε ότι κατά τη διάρκεια της εκπαίδευσης το νευρωνικό έχει ολοένα και μικρότερο loss. Παρόλα αυτά η τελική επίδοση θα κριθεί στο test dataset.

```
Epoch: 32       Batch: 100      Loss 0.007974412971139324
Epoch: 32       Batch: 200      Loss 0.007024518022746941
Epoch: 33       Batch: 0        Loss 0.01650902070105076
Epoch: 33       Batch: 100      Loss 0.006818362036235815
Epoch: 33       Batch: 200      Loss 0.006097755536630481
Epoch: 34       Batch: 0        Loss 0.003719078144058585
Epoch: 34       Batch: 100      Loss 0.008230881452546593
Epoch: 34       Batch: 200      Loss 0.00721864192196358
Epoch: 35       Batch: 0        Loss 0.0039731222204864025
Epoch: 35       Batch: 100      Loss 0.005068315217262226
Epoch: 35       Batch: 200      Loss 0.005576654803442226
Epoch: 36       Batch: 0        Loss 0.004333892837166786
Epoch: 36       Batch: 100      Loss 0.007154773380192421
Epoch: 36       Batch: 200      Loss 0.006126613235993042
Epoch: 37       Batch: 0        Loss 0.00792566780000925
Epoch: 37       Batch: 100      Loss 0.007076076586329664
Epoch: 37       Batch: 200      Loss 0.00600290477761781
Epoch: 38       Batch: 0        Loss 0.0041721840389072895
Epoch: 38       Batch: 100      Loss 0.004410386988737546
Epoch: 38       Batch: 200      Loss 0.005792397278924565
Epoch: 39       Batch: 0        Loss 0.0016922674840316176
Epoch: 39       Batch: 100      Loss 0.0034902724953216157
Epoch: 39       Batch: 200      Loss 0.0049468746871585536
```

γ) Στο ερώτημα αυτό γράψαμε τον κώδικα για την εκπαίδευση και το evaluation του νευρωνικού ,συμβατή με το scikit-learn. Σημειώνουμε ότι τμήματα του κώδικα που χρησιμοποιήθηκαν (σχετικά με την εκπαίδευση και την αξιολόγηση) γράφτηκαν σε αναλογία με τον δοσμένο κώδικα του εργαστηρίου του μαθήματος.

Σημειώνουμε ότι για την υλοποιήσαμε επίσης χωρισμό τον δεδομένων του αρχείου train.txt σε train και validation, και χρησιμοποιήσαμε σε κάθε περίπτωση τα πρώτα για την επαίδευση του νευρωνικού και τα δεύτερα για την αξιολόγησή του κατά την διαδικασία της ανάπτυξης.

Τελικά λάβαμε τα εξής αποτελέσματα:

Neural Network	Hidden Layer nodes	Activation Function	Score on train data (%)	Score on validation data(%)
2-layer	128	ReLU	99.09	97.03
2-layer	32	ReLU	98.64	95.66
2-layer	128	Sigmoid	97.73	95.34
2-layer	32	Sigmoid	98.12	95.36
3-layer	128,64	ReLU	98.96	96.67
3-layer	64,32	ReLU	98.83	96.16
3-layer	128,64	Sigmoid	97.17	94.92
3-layer	64,32	Sigmoid	97.27	94.70

Παρατηρούμε πως η ακρίβεια στα train δεδομένα είναι σε κάθε περίπτωση υψηλότερη από αυτήν στα validation δεδομένα. Αυτό είναι λογικό, καθώς το νευρωνικό έχει ήδη δει τα δεδομένα εκπαίδευσης κατά την εκπαίδευσή του, ενώ στα δεδομένα εκπαίδευσης, τα οποία δεν έχει "δει" ξανά μπορεί να υπάρχουν δείγματα που απέχουν περισσότερο από τα δεδομένα στα οποία το νευρωνικό εκπαιδεύτηκε.

δ) Τέλος, έγινε η αξιολόγηση των νευρωνικών στα **testing δεδομένα**. Τα αποτελέσματα που προέκυψαν είναι τα εξής:

Neural Network	Hidden Layer nodes	Activation Function	Accuracy (%)
2-layer	128	ReLU	93.67
2-layer	32	ReLU	92.97
2-layer	128	Sigmoid	92.18
2-layer	32	Sigmoid	92.07
3-layer	128,64	ReLU	93.17
3-layer	64,32	ReLU	93.07
3-layer	128,64	Sigmoid	91.62
3-layer	64,32	Sigmoid	91.88

Παρατηρούμε πως η καλύτερη απόδοση επιτεύχθηκε για το 3-layer νευρωνικό δίκτυο με συνάρτηση ενεργοποίησης ReLU και πλήθος κόμβων 128,64 στα 2 κρυφά επίπεδα αντίστοιχα. Το αποτέλεσμα που προέκυψε είναι αρκετά ικανοποιητικό ωστόσο αρκετές από τις προηγούμενες μεθοδοι, πχ SVM KNN πέτυχαν τελικά καλύτερο score.

Συγκρίνοντας αναλυτικότερα τις επιμέρους ακρίβειες των διαφόρων νευρωνικών μεταξύ τους μπορούμε να παρατηρήσουμε πως γενικά για ίδιο αριθμό νευρώνων και επιπέδων η χρήση της συνάρτησης ενεργοποίησης ReLU οδηγεί σε μεγαλύτερη ακρίβεια σε σχέση με την σιγμοειδή. Επίσης, για ίδιο πλήθος επιπέδων και συνάρτηση ενεργοποίησης παρατηρούμε ότι με μεγαλύτερο αριθμό νευρώνων στο hidden layer επιτεύχθηκαν καλύτερα αποτελέσματα. Τέλος, συγκρίνοντας την ακρίβεια που προκύπτει με προσθήκη ενός επιπλέον hidden layer, παρατηρούμε πως αυτή δεν

άλλαξε σημαντικά, δηλαδή δεν έχουμε μεγάλες διαφορές στις επιδόσεις των νευρωνικών δύο ή τριών layers.