Σχολή : Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Μάθημα :Αναγνώριση Προτύπων

Ονοματεπώνυμο : Κωνσταντίνος Τσιγγέλης ,03117149

Σιαφαρίκας Χρήστος ,03117097

Εξάμηνο : 9ο

1η Εργαστηριακή Άσκηση: Οπτική Αναγνώριση Ψηφίων

Στα πλαίσια της συγκεκριμένης εργασίας καλούμαστε να λύσουμε το πρόβλημα ταξινόμησης εικόνων που περιέχουν χειρόγραφα ψηφία από το 0 εώς το 9 σε 10 κατηγορίες ανάλογα με το ψηφίο που απεικονίζουν. Κάθε εικόνα αποτελείται από 256 πίξελ και αντιστοιχίζεται σε έναν 16x16 πίνακα. Για τον σκοπό αυτό αξιοποιούνται δεδομένα που προέρχονται από την US Postal Service και χωρίζονται σε train, για την εκμάθηση του συστήματος, και σε test για να αξιολογήσουμε την απόδοσή του. Θα εφαρμόσουμε λοιπόν supervised learning με διάφορα μοντέλα εκτίμησης αλλά και με συνδυασμούς αυτών προκειμένου να εξετάσουμε την αποδοτικότητά τους αλλά και την συμπεριφορά τους.

Βήμα 1

Στο συγκεκριμένο βήμα γίνεται η αναγνώριση των αρχείων τα οποία βρίσκονται σε μορφή .txt σε μορφή συμβατή με το scikit-learn σε 4 πίνακες X_train, X_test, Y_train και Y_test. Οι 2 πρώτοι πίνακες περιέχουν τα 256 features τις κάθε εικόνας ενω οι 2 τελευταιοι περιέχουν μόνο τα labels (ακέραιες τιμές απο το 0 εώς το 9). Αναλυτικότερα κάθε γραμμή των X_test και X_train πινάκων αντιστοιχεί στα 256 features/pixels μίας εικόνας ενώ το αντίστοιχο label της εικόνας βρίσκεται στην αντίστοιχη γραμμή των πινάκων Y_test και Y_train. Το train set περιέχει 7291 εικόνες και το test 2007. Η υλοποίηση φαίνεται ακολούθως ενδεικτικά για το train αρχείο:

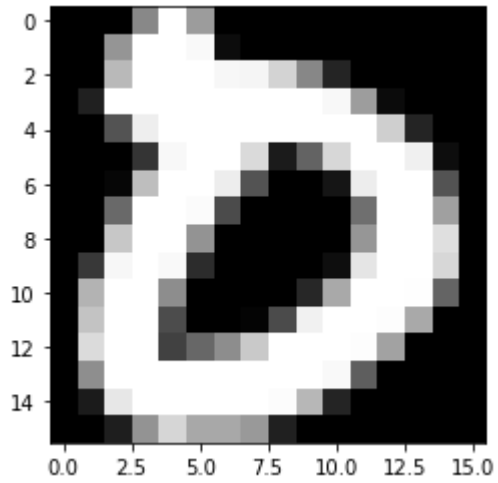
```
Y_train=[]
X_train=[]

#read train txt
with open(r'D:\Ktsin_Files\Downloads\pr_lab1_data\train1.txt') as f:

    pk = f.readlines()


    for i in pk:
        w=[]
        wi= i.split()
        for i in wi:
            w.append(float(i))
        Y_train.append(w[0])
        for i in range(1,len(w)):
            x1.append(w[i])
X_train=np.reshape(x1,(len(x1)//256,256))
Y_train =np.asarray(Y_train)
```

Τα αποτελέσματά μας είναι τα εξής:

```
X_train= [[-1.    -1.    -1.    ... -1.    -1.    -1.   ]
 [-1.    -1.    -1.    ... -0.671 -0.828 -1.   ]
 [-1.    -1.    -1.    ... -1.    -1.    -1.   ]
 ...
 [-1.    -1.    -1.    ... -1.    -1.    -1.   ]
 [-1.    -1.    -1.    ... -1.    -1.    -1.   ]
 [-1.    -1.    -1.    ... -1.    -1.    -1.   ]]

X_test= [[-1. -1. -1. ... -1. -1. -1.]
 [-1. -1. -1. ... -1. -1. -1.]
 [-1. -1. -1. ... -1. -1. -1.]
 ...
 [-1. -1. -1. ... -1. -1. -1.]
 [-1. -1. -1. ... -1. -1. -1.]
 [-1. -1. -1. ... -1. -1. -1.]]

Y_train= [6.0, 5.0, 4.0, 7.0,..... 3.0, 0.0, 1.0]

Y_test= [9.0, 6.0, 3.0,.....4.0, 0.0, 1.0]
```

Βήμα 2

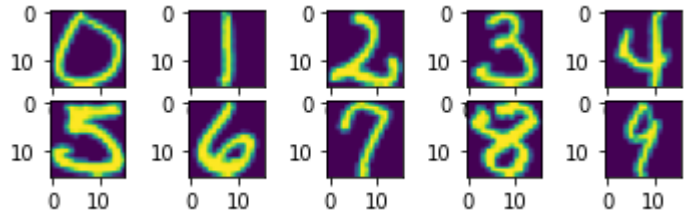
Στο βημα 2 της εργασίας μας ζητείται μια απλή σχεδίαση του ψηφίου με θέση 131 στο αρχείο train. Με βάση ότι οι πίνακες μας έχουν το πρώτο στοιχείο στη θέση 0, καταλαβαίνουμε ότι το στοιχείο 131 βρίσκεται στη θέση 130 του πίνακα και ειναι το κατωθι:



Για να πετύχουμε αυτό το σχεδιασμό χρησιμοποιήσαμε τη συνάρτηση np.reshape(X_train[130],(16,16)).

Βήμα 3

Για να διαλέξουμε ένα δείγμα από κάθε label χρησιμοποιούμε την εντολή l[i] = X_train2[Y_train.index(i)]. Τα νούμερα αν και σχεδιασμένα με το χέρι ειναι σχετικά ευδιάκριτα πράγμα που βοηθά σε μεγάλο βαθμό τη διαδικασία της μάθησης. Ετσι έχουμε:



Βήμα 4

Στο συγκεκριμένο βήμα υπολογίζουμε την μέση τιμή των features του πίξελ 10x10 για το ψηφίο 0 με βάση τα train δεδομένα. Αξίζει να σημειώσουμε ότι το συγκεκριμένο πίξελ αντιστοιχεί στην 9*16+9 στήλη δηλαδή στην 153η στήλη του πίνακα X_train εφόσον όταν εφαρμόσουμε την εντολή reshape στα χαρακτηριστικά μίας εικόνας για να την οδηγήσουμε στην επιθυμητή μορφή πίνακα 16x16 ένα πίξελ (i,j) με αρίθμηση απο το 0 αντιστοιχεί στην i*16 + j στήλη του X_train. Το αποτέλεσμα είναι ένας αριθμός της τάξης του -0.93 που είναι πάρα πολύ μικρος αν σκεφτούμε ότι τα πίξελς παίρνουν τιμές στο διάστημα [-1,1] .Αυτό σημαίνει πως στη συγκεκριμένη θέση σπάνια βρίσκεται μέρος του σχεδίου του μηδέν αποτέλεσμα που αναμέναμε εφόσον η θέση αυτή είναι κεντρική και το σχήμα του μηδέν έχει κενό στο κέντρο.

```
values=[]
for i in range(len(Y_train)):
    if Y_train[i]==0:
        values.append(X_train[i,153])
mean=sum(values)/len(values)
```

Result:
mean_(10x10pixel)=-0.927264656616415

Βήμα 5

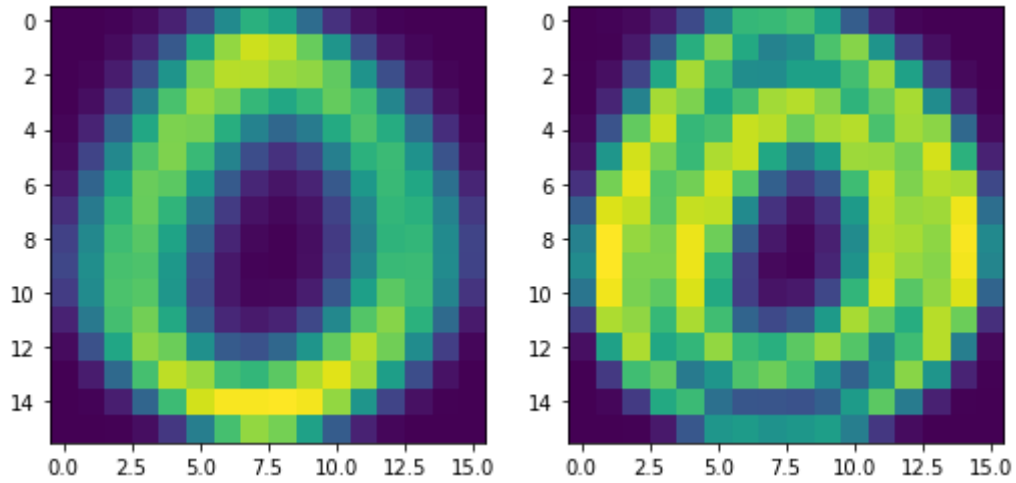
Με την ίδια λογική με το παραπάνω ερώτημα χρησιμοποιούμε την np.var() και υπολογίζουμε τη διασπορά των χαρακτηριστικών του pixel (10, 10) για το ψηφίο «0». Ετσι:
διασπορά = 0.08392455977262078

Βήμα 6

Για να υπολογίσουμε τα παραπανω των χαρακτηριστικών κάθε pixel για το ψηφίο «0» συγκεντρώνουμε τις θέσεις των 0 σε ένα πίνακα με την εντολή ap = np.where(Y_train==0.0) και στη συνέχεια τα μαζεύουμε ολα τα 0 σε ενα πινακα μεσω του οποίου βρίσκουμε τα ζητούμενα.

Βήμα 7,8

Με τις εντολες plt.imshow(mtola),plt.imshow(diaola) σχεδιάζουμε το ψηφίο 0 απο την μεση τιμή του και τη διασπορά του. Οπότε παραθέτουμε τα σχήματα:



Παρατηρούμε οτι το 0 της μέσης τιμής είναι πιο “καθαρό” καθώς και πιο “λεπτό” πράγμα που περιμέναμε σε σχέση με τη διασπορά αφού η τελευταία είναι μεγαλύτερη εκει που οι τιμές των εικόνων είναι μεγαλύτερες, δηλαδή στα σημεία που δεν είναι κενά.

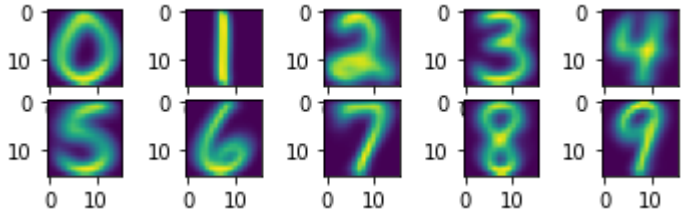
Βήμα 9

Για το ερώτημα αυτο δημιουργούμε μια συνάρτηση η οποία ουσιαστικά λειτουργεί όπως στα προηγούμενα ερωτήματα για την μέση τιμή και τη διασπορά του 0. Σαν είσοδο δέχεται το ψηφιο και ακολουθώντας την παραπάνω διαδικασία επιστρέφει για την μέση τιμή και τη διασπορά με βάση όλα τα δεδομένα για το ψηφίο της εισόδου. Η υλοποίηση αυτή είναι:

```
def diamesh(number):

    k3=[]
    k4=[]
    k5=[]
    k=[]
    diaola=[]
    ap = np.where(Y_train==number)

    for i in ap[0]:
      k3.append(X_train[i])
      k.append(X_train2[i])
    kpl=[0.0]*16
    k3 = np.array(k3)
    for i in k:
      kpl = i+kpl
    #mtola =kpl/len(k)
    mtola=[]
    for i in range(256):
       k4 = k3[:,i]
       k5.append(k4)
```

```
    for i in k5:
      mtola.append(np.mean(i))
      diaola.append(np.var(i))
    diaola = np.reshape(diaola,(16,16))
    mtola = np.reshape(mtola,(16,16))
    return mtola,diaola
```

Και παρουσιάζουμε όλα τα ψηφία χρησιμοποιώντας τις τιμές της μέσης τιμής που υπολογίσατε στο προηγούμενο βήμα(9α).



Βήμα 10

Βρίσκουμε την ευκλείδεια απόσταση μεταξύ της μεσης τιμης των χαρακτηριστικών των 10 labels και των χαρακτηριστικών του ζητούμενου ψηφίου και μετα με την εντολή np.argmin(k44)*1.0. Παρατηρούμε ότι τελικά με αυτόν τον τρόπο η ταξινόμηση για το 101 ψηφίο ειναι σωστη.

Βήμα 11

Καλούμε με μια επανάληψη την παραπάνω διαδικασία για ολα τα ψηφια και εξάγουμε το ποσοστό επιτυχίας με βαση τα labels και τις προβλέψεις μας. Το ποσοστό αυτό είναι 81%(0.8141504733432985).

Βήμα 12

Προσπαθούμε να υλοποιήσουμε τον ταξινομητή ευκλείδειας απόστασης σαν ένα scikit-learn estimator. Έχουμε τρεις μεθόδους και την init. Η fit περιέχει τη διαδικασία υπολογισμού μέσης τιμής με τη συνάρτηση που δημιουργήσαμε πριν. Στην predict επαναλαμβάνουμε τον τρόπο που κάναμε της προβλέψεις νωρίτερα και τέλος στο score βρίσκουμε το ποσοστό επιτυχίας των προβλέψεων μας.

Βήμα 13α

Ένα σημαντικό μέσο αξιολόγησης του ευκλείδειου ταξινομητή είναι κάνοντας χρήση του K Fold Validation σύμφωνα με τον οποίο υπολογίζουμε ποσοστά επιτυχίας για διαφορετικούς διαχωρισμούς των δεδομένων σε train και test. Η διαδικασία που ακολουθούμε είναι η εξής : συνενωνουμε τα train και test αρχεία σε ένα κοινό και στην συνέχεια διαχωρίζουμε το συνολικό αρχείο σε train και test με 5 διαφορετικούς τρόπους . Για κάθε ξεχωριστό συνδυασμό train και test δεδομένων υπολογίζω το ποσοστό επιτυχίας και τελικά παίρνουμε το μέσο όρο αυτών για να έχουμε κι ένα συνολικό ποσοστό. Με τη μέθοδο αυτή έχουμε καλύτερη αξιοποίηση των διαθέσιμων δεδομένων και μάλιστα τόσο καλύτερη όσο μεγαλύτερος είναι ο αριθμός των splits. Επιπλέον το ποσοστό επιτυχίας που εξάγουμε είναι πιο αντιπροσωπευτικό. Η υλοποίηση γίνεται κάνοντας χρήση της εντολής cross_val_score από την βιβλιοθήκη sklearn.model_selection όπως φαίνεται ακολούθως:

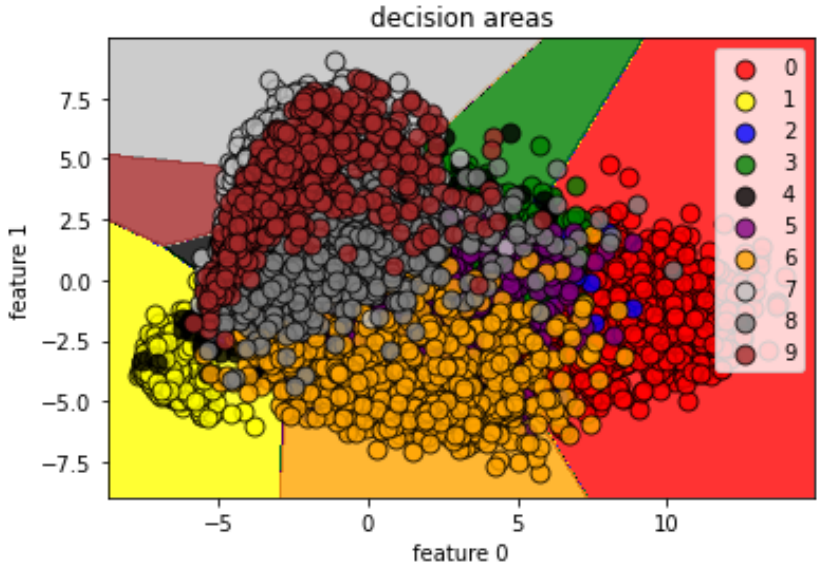
scores = cross_val_score(EuclideanClassifier(), X, Y,cv=KFold(n_splits=5, random_state=1,shuffle=True),scoring="accuracy")

Τα αποτελέσματά μας είναι τα εξής :
scores on each of the 5 splits: [0.85268817 0.83172043 0.84193548 0.84238838 0.83808499]

Average Score: 0.8413634917605141

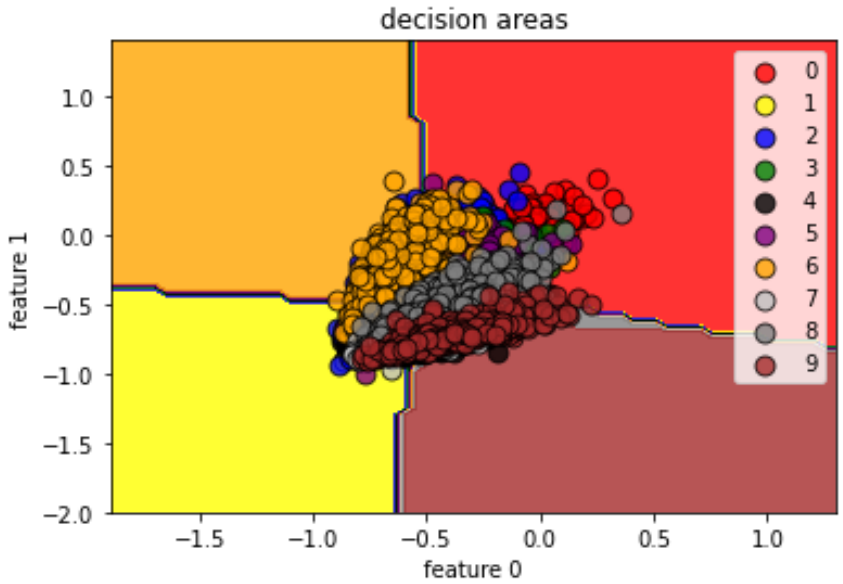
Βήμα 13b

Στο συγκεκριμένο βήμα θα οπτικοποιήσουμε την επίδοση του Ευκλείδειου ταξινομητή απεικονίζοντας τις περιοχές απόφασης. Για τον σκοπό αυτό θα πρέπει να απομονώσουμε δύο χαρακτηριστικά σύμφωνα με τα οποία θα γίνεται ο διαχωρισμός των δεδομένων στις κλάσεις και μάλιστα τα δύο αυτά χαρακτηριστικά θα παίζουν τον ρόλο των αξόνων (οριζόντιο:feature 0 , κάθετο :feature 1) στο τελικό διάγραμμα. Προφανώς με το να χρησιμοποιήσουμε 2 τυχαία features απο τα 256 γαι την ταξινόμηση των δεδομένων τα αναμενόμενα αποτελέσματα δεν θα είναι καλά. Για τον λόγο αυτό χρησιμοποιούμε διαφορετικές μετρικές οι οποίες ενσωματώνουν στα 2 features μεγαλύτερη και πιο αντιπροσωπευτική πληροφορία. Αρχικά χρησιμοποιούμε PCA για να μειώσουμε την διάσταση των δεδομένων μας στην επιθυμητή. Η ελλάτωση διάστασης γίνεται με SVD(singular value decomposition). Τα αποτελέσματά μας φαίνονται στην συνέχεια:



Επίσης μία άλλη μέθοδος που χρησιμοποιήσαμε για να ενσωματώσουμε όσο το δυνατόν περισσότερη πληροφορία στα 2 features είναι να ενσωματώσουμε σε κάθε feature την μέση τιμή του μισού πίνακα.Αναλυτικότερα:
feature0=np.mean(X[:,:153],axis=1)

feature1=np.mean(X[:,153:256],axis=1)

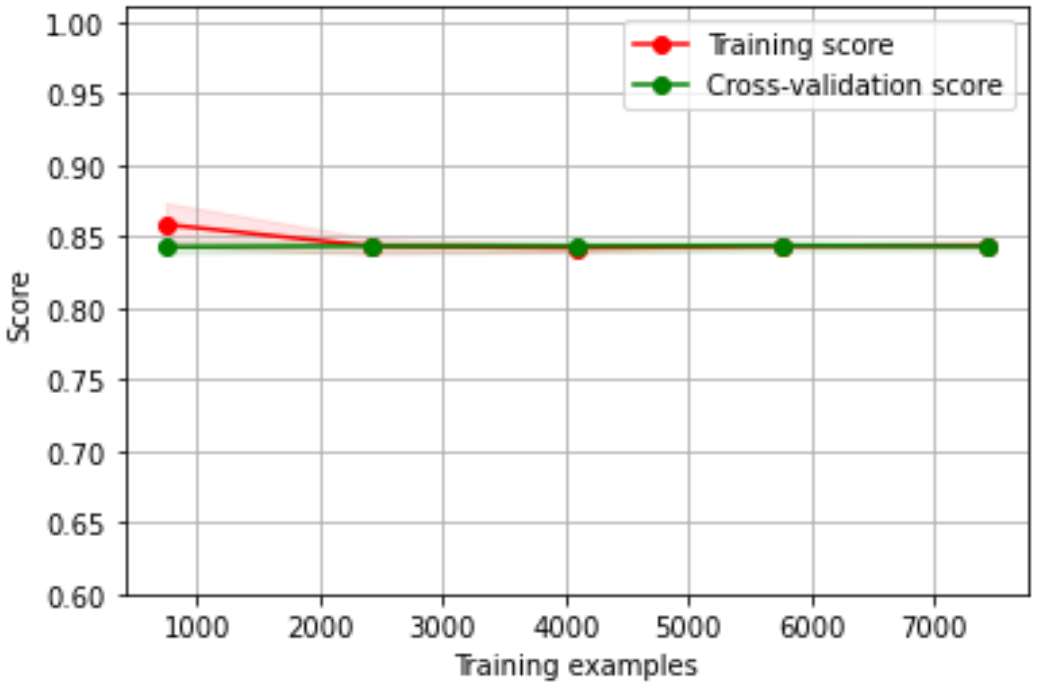


Στα διαγράμματα των περιοχών απόφασης βλέπουμε με συνεχές χρώμα τις περιοχές απόφασης. Η περιοχή απόφασης αντιστοιχεί στο γεωμετρικό τόπο του x-y επιπέδου όπου ο ταξινομητής δεχόμενος τις συντεταγμένες [feature0,feature1] δίνει ως έξοδο το label αυτό. Επίσης βλέπουμε την απεικόνιση των δειγμάτων του train πάνω στο x-y επίπεδο (scatter). Κάθε σημείο του train πηγαίνει στην κατάλληλη θέση x,y ανάλογα με τις τιμές των χαρακτηριστικών του κι έχει το χρώμα που αντιστοιχεί στο label του όπως φαίνεται στο legend. Η ποιότητα αλλά και η επιτυχία της ταξινόμησης που εφαρμόσαμε καθορίζεται από το πόσο ομαδοποιημένα αλλα και πόσο μη επικαλυπτόμενα είναι τα δεδομένα μεταξύ τους. Τέλος, το πρώτο διάγραμμα των περιοχών απόφασης είναι πολύ πιο ξεκάθαρο εφόσον ο PCA αποτελεί πολύ καλύτερη μετρική για τη διάσπαση των χαρακτηριστικών από ότι η μέση τιμή.

Βήμα 13c

Στο συγκεκριμένο βήμα σχεδιάζουμε την καμπύλη εκμάθησης. Το διάγραμμα αυτό μας δείχνει πως μεταβάλλεται το training score και το cross-validation score καθώς προσθέτουμε δείγματα στη μάθηση. Υψηλό training score σημαίνει ότι έχουμε καλό fitting στο μοντέλο μας αλλα αυτο πρέπει να συνοδεύεται και από αντιστοίχως υψηλό cross validation score. Σε αντίθετη περίπτωση επιβαρυνόμαστε απο σφάλματα variance αλλά και bias errors δηλαδή εσφαλμένες προβλέψεις .Σε κάθε περίπτωση πρέπει να υπάρχει μία ισορροπία μεταξύ των τιμών αυτών . Η καμπύλη αυτή είναι ένα εργαλείο για να μάθουμε πόσο ωφελείται ένα μοντέλο μηχανής από την προσθήκη περισσότερων δεδομένων εκπαίδευσης. Με άλλα λόγια εντοπίζουμε πότε μπορούμε να σταματήσουμε την εκπαίδευση λόγω κορεσμού εφόσον τα score συγκλίνουν σε συγκεκριμένες τιμές. Τέλος , μας δίνει την δυνατότητα να αναγνωρίσουμε πότε έχουμε bias error και τα score συγκλίνουν σε κακές τιμές και πότε έχουμε variance error όπου το training score είναι πολύ υψηλότερο του cross validation score.

Στο δικό μας παράδειγμα το training score πέφτει (ελαφρώς) και το validation score αυξάνει καθώς προσθέτουμε δείγματα ξεκινώντας από 1000, ενώ μετά τα 2500 δείγματα επέρχεται σύγκλιση στην τιμή 0.85.



Βήμα 14

Στο συγκεκριμένο βήμα υπολογίζουμε τις a_priori πιθανότητες για κάθε κλάση στην οποία μπορεί να ταξινομηθεί ένα δείγμα. Χρησιμοποιούμε την ακόλουθη μαθηματική σχέση P(Yc)=$\frac{Nc}{N}$ για κάθε κλάση c. Τα αποτελέσματά μας είναι τα ακόλουθα:
P(digit:0)=0.16376354
P(digit:1)=0.13784117
P(digit:2)=0.1002606
P(digit:3)=0.09024825
P(digit:4)=0.08942532
P(digit:5)=0.0762584
P(digit:6)=0.09107118
P(digit:7)=0.08846523
P(digit:8)=0.07433823
P(digit:9)=0.08832808
sum=1.0

Βήμα 15(a,b,c)

Σύμφωνα με τον BayessianClassifier ο μαθηματικός νόμος πρόβλεψης που υλοποιείται είναι ο ακόλουθος :

$$\hat{y} = \arg\max_y P(y)\prod_{i=1}^{n} P(x_i \mid y)$$

με P(y) τις a_priori πιθανότητες που υπολογίσαμε προηγουμένως και την δεσμευμένη πιθανότητα P(xi/y) που είναι η πιθανοφάνεια δηλαδή η πιθανότητα του δείγματος με δεδομένη την υπόθεσή μας και μπορεί επίσης να υπολογιστεί απλά από το training set ως η πυκνότητα πιθανότητας μίας κανονικής κατανομής

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

.

Ο scikit-learn εκτιμητής που υλοποιήσαμε κάνει predict τον ακόλουθο πίνακα:
[9. 6. 3. ... 9. 0. 1.]
και έχει score:
Current Score: 0.7070254110612855
CustomNBClassifier scores on each of the 5 splits: [0.72526882 0.74139785 0.69247312 0.74341044 0.70521786]
CustomNBClassifier CV Average Score: 0.7215536159456756

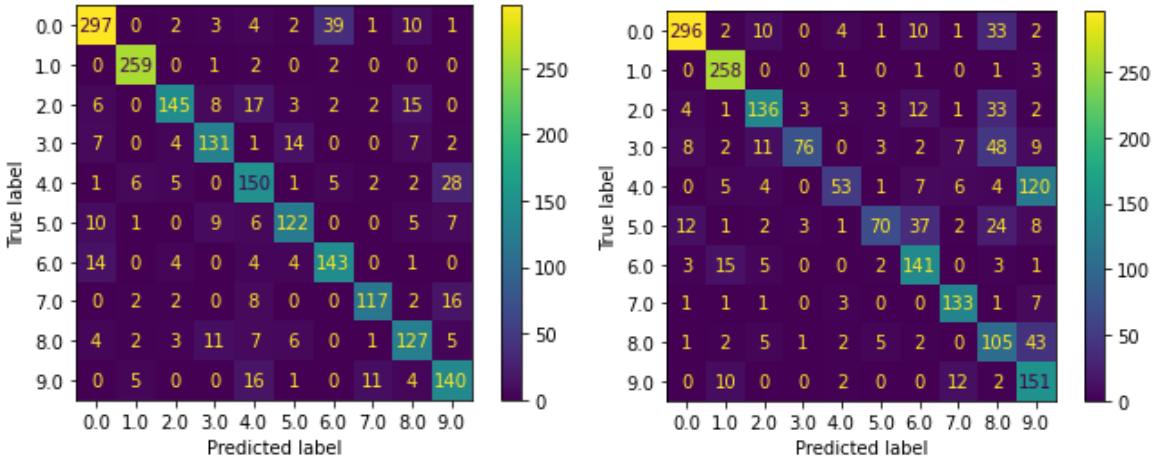
Όσον αφορά την σύγκριση της δικιάς μας υλοποίησης με την αντίστοιχη υλοποίηση Naive Bayes του scikit-learn(GausianNB)
sklearn NBGaussian scores on each of the 5 splits: [0.73978495 0.75107527 0.70806452 0.75739645 0.72027972]
sklearn NBGaussian CV Average Score: 0.7353201802333316

παρατηρούμε ότι η δικιά μας υπολείπεται κατά μία ποσοστιαία μονάδα.

Βήμα 16

Στο συγκεκριμένο βήμα επαναλαμβάνουμε τα βήματα 15α και 15β για var=1.
Οι προβλέψεις που παίρνουμε είναι οι: [9. 2. 3. ... 4. 0. 1.]
ενώ το score του ταξινομητή είναι 0.8126557050323866.
Τα αποτελέσματα βελτιώθηκαν σαφώς και είναι πλέον παρόμοια με τον mean euclidean.
Συγκρίνουμε αναλυτικότερα τις 2 υλοποιήσεις του BaseEstimator (αυτή που προέκυψε απο το ερώτημα 15 και για var=1) μέσω των confusion boxes:

var=1 var(X_train)



Βλέπουμε λοιπόν μια σαφή βελτίωση των προβλέψεών μας ιδιαιτερα στα ψηφία που κάνουμε predict ως 8 και 9 .

Βήμα 17

Αρχικά μελετάμε την χρήση του ταξινομητή kNN για την επίλυση του προβλήματος.Στην ταξινόμηση kNN ένα αντικείμενο ταξινομείται με πλήθος ψηφοφοριών των γειτόνων του, με το αντικείμενο να εκχωρείται στην κατηγορία που είναι πιο κοινή μεταξύ των *k* πλησιέστερων γειτόνων του. Αν *k* = 1, τότε το αντικείμενο απλώς εκχωρείται στην κλάση αυτού του απλού πλησιέστερου γείτονα.
Πειραματιζόμαστε με διάφορες τιμές του k και παίρνουμε τα ακόλουθα score με χρήση 5-Fold-Validation:

k=1 mean_scores= 0.96579951066 scores_std= 0.005151762140384
k=3 mean_scores 0.964508262622 scores_std 0.0033529475834133
k=5 mean_scores 0.961389636005 scores_std 0.0023647767535684
k=15 mean_scores 0.948698167010 scores_std 0.0026566546361207

Παρατηρούμε ότι το καλύτερο score έχουμε για k=1 και μάλιστα με ταυτόχρονη high variance. Επίσης παρατηρούμε ότι όσο πιο μικρό είναι το k τόσο μεγαλύτερο είναι τόσο το score όσο και το variance.

Η δεύτερη μέθοδος που μελετήσαμε είναι η SVM η βασική ιδέα της οποίας είναι η εύρεση του βέλτιστου hyperplane που διαχωρίζει βέλτιστα τα δεδομένα σε κλάσεις. Κάναμε tuning των παραμέτρων kernel, C και gamma με χρήση της εντολής GridSearch. Ιδιαίτερα δοκιμάσαμε διάφορους τύπους kernel:
γραμμικό με τύπο K(x, xi) = sum(x * xi)
πολυωνυμικό με τύπο K(x,xi) = 1 + sum(x * xi)^d με d τον βαθμό του πολυωνύμου
και rbf με τύπο K(x,xi) = exp(-gamma * sum((x – xi^2)) όπου gamma μία παράμετρο με τιμή στο διάστημα [0,1]
Οι τιμές που δοκιμάσαμε για τον πυρήνα C είναι οι (1,10,100) ενω για το gamma δοκιμάσαμε τις τιμές (0.01 , 0.001). Τα αποτελέσματα στα οποια καταλήξαμε φαίνονται ακολούθως:

Grid scores:

mean_score:0.9783 var:0.00598 params {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'})
mean_score:0.9550 var: 0.00670 params: {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'})
mean_score:0.9805 var:0.00430 params:{'C': 10, 'gamma': 0.01, 'kernel': 'rbf'})

mean_score:0.9681 var:0.00653 params:{'C': 10, 'gamma': 0.001, 'kernel': 'rbf'})
mean_score:0.9805 var:0.00430 params:{'C': 100, 'gamma': 0.01, 'kernel': 'rbf'})
mean_score:0.9685 var:0.01010 params:{'C': 100, 'gamma': 0.001, 'kernel': 'rbf'})

mean_score:0.9546 var:0.00964 params:{'C': 1, 'kernel': 'linear'}
mean_score:0.9547 var:0.00982 params:{'C': 10, 'kernel': 'linear'}
mean_score:0.9547 var:0.00982 params:{'C': 100, 'kernel': 'linear'}

mean_score:0.9797 var:0.00456 params:{'C': 1, 'kernel': 'poly'}
mean_score:0.9802 var:0.006165 params:{'C': 10, 'kernel': 'poly'}
mean_score:0.9801 var:0.00614 params:{'C': 100, 'kernel': 'poly'}

Best Score and best parameters:

best_score:0.980523612486144
best parameters; {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}

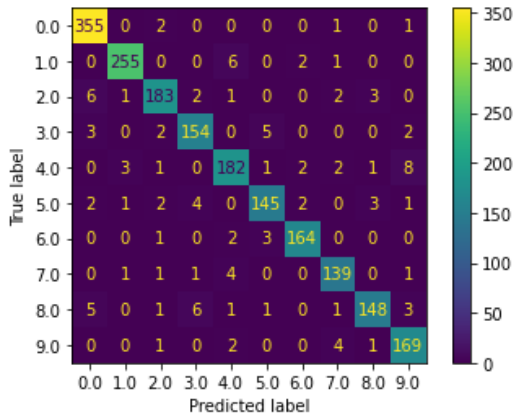
Συνεπώς παρατηρούμε πως ο SVM_Classifier μας δίνει πολυ υψηλή αποδοτικότητα της τάξης του 0,981 με σωστό tuning των παραμέτρων. Τώρα συνολικά οι ταξινομητές που έχουμε μελετήσει ακολουθούν την εξής ‘ιεράρχηση’ ανάλογα με την αποδοτικότητά τους:

SVM > KNN > Euclidean Classifier > Bayesian Classifier

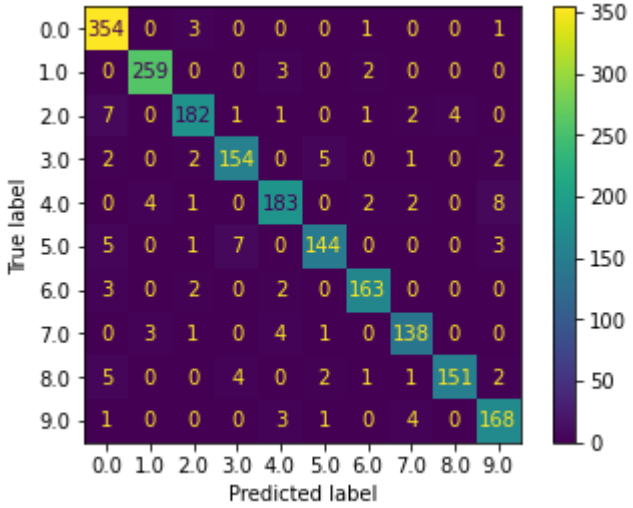
Βήμα 18α

Στο συγκεκριμένο βήμα εφαρμόζουμε την τεχνική ensemblin δηλαδή συνδυάζουμε ταξινομητές με αρκετά καλή επίδοση με στόχο να επιτευχθεί επίδοση υψηλότερη των επιμέρους επιδόσεων, με τον κάθε ταξινομητή να εκπαιδεύεται σε όλο το training set. Είναι σημαντικό οι ταξινομητές που θα συνδυαστούν να χαρακτηρίζονται από διαφορετικό τύπο λαθών ώστε να υπάρχει κάποια βελτίωση στο συνολικό score. Οι ταξινομητές που επιλέξαμε να συνδυάσουμε μαζί με τα confusion matrix τους είναι οι εξής:

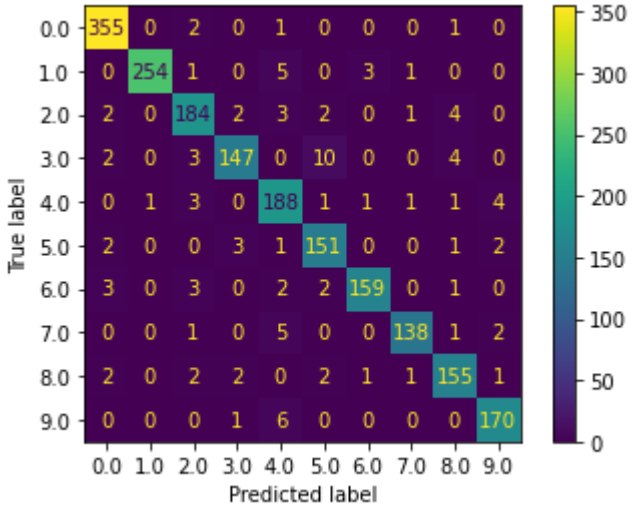
KNeighborsClassifier(n_neighbors=1)



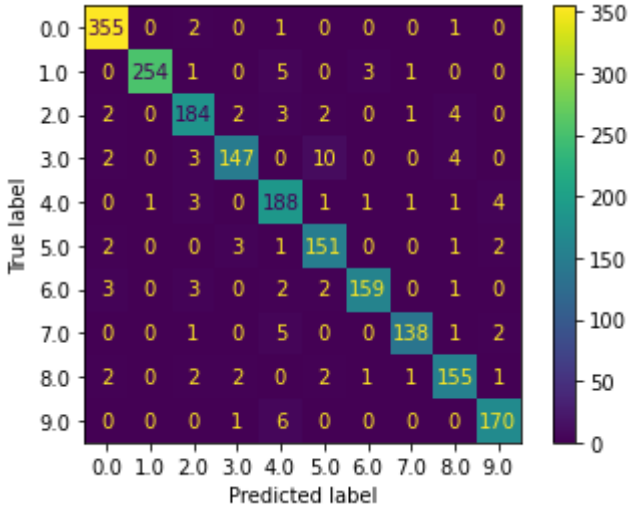
KNeighborsClassifier(n_neighbors=5)



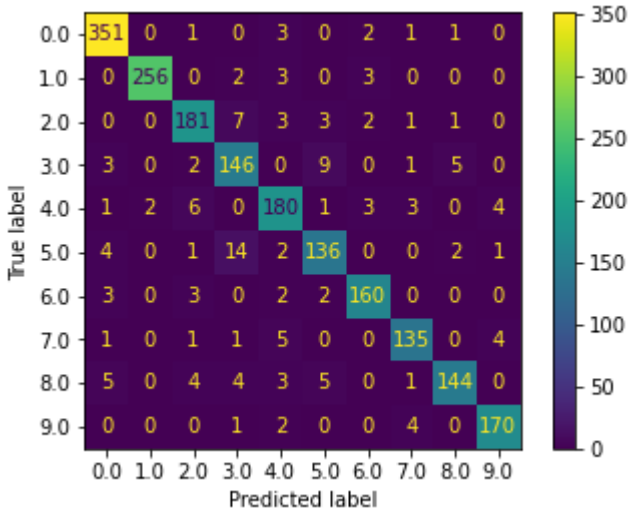
SVC(kernel='rbf',gamma='scale',probability=True)



SVC(kernel='rbf',gamma='scale',probability=True)



SVC(kernel='linear',probability=True)



Χρησιμοποούμε το VotingClassifier του scikit-learn για να τους συνδυάσουμε σε hard ή soft voting. Στον hard επιλέγεται με μορφή tournament η κλάση που πιστεύει η πλειοψηφία των classifiers και για αυτό το λόγο και επιλέγουμε μονό αριθμό classifiers προκειμένου να αποφευχθεί η πιθανότητα ισοπαλίας, καθώς οι ταξινομητές συνήθως δυσκολεύονται να αποφασίσουν μεταξύ δύο μόνο υποψήφιων κλάσεων από τις συνολικές. Από την άλλη, στο soft mode κάθε ταξινομητής παράγει τις πιθανότητες για κάθε sample αυτό να ανηκεί σε κάθε κλάση. Συγκεκριμένα στην άσκησή μας για κάθε sample παράγονται 10 πιθανότητες για κάθε digit (0-9). Στην συνέχεια, για κάθε κλάση προστίθενται οι επιμέρους πιθανότητες όλων των ταξινομητών και επιλέγεται η κλάση με το μεγαλύτερο άθροισμα.Τα αποτελέσματά μας φαίνονται ακολούθως:
1_NN accuracy score: 0.9436970602889886
5_NN accuracy score: 0.9446935724962631
SVM with poly kernel accuracy score: 0.94971848530144494
SVM with rbf kernel accuracy score: 0.9471848530144494
SVM with linear kernel accuracy score: 0.9262580966616841
Hard VotingClassifier score: 0.9501743896362731
Soft VotingClassifier score: 0.9511709018435476

Βλέπουμε λοιπόν πως και στις δύο περιπτώσεις(hard and soft voting) έχουμε μία μικρή βελτίωση του score.

Βήμα 18b

Στο συγκεκριμένο βήμα επιλέγουμε ταξινομητές από τα προηγούμενα βήματα και χρησιμοποιούμε τον BaggingClassifier για να δημιουργήσουμε ένα ensemble. Η τεχνική bagging, αφορά στο χωρισμό του ,training) dataset σε τυχαία υποσύνολα (με πιθανές επικαλύψεις) και την εφαρμογή ενός ταξινομητή σε κάθε ένα από αυτά. Η τελική απόφαση βγαίνει μέσω ψηφοφορίας ή μέσου όρου των προβλέψεων των επιμέρους ταξινομητών.

Αρχικά εφαρμόζω τον BaggingClassifier για τον ταξινομητή NearestNeighbor με k=5. Τα αποτελέσματά μας είναι τα εξής:
BaggingClassifier score : 0.9417040358744395
5_NN score: 0.94469

Στην συνέχεια εφαρμόζω τον BaggingClassifier για τον ταξινομητή SVC(kernel='poly',gamma='scale',C=1) Τα αποτελέσματά μας είναι τα εξής:
BaggingClassifier score : 0.9521674140508222
SVM with poly kernel score: 0.94971848530144494

Τέλος εφαρμόζω τον BaggingClassifier για τον ταξινομητή Decision Trees και μας δίνει τα ακόλουθα αποτελέσματα (ο παραπάνω συνδυασμός μας δίνει τον τον ταξινομητή Random Forest):
BaggingClassifier score:0.8849028400597907
Decision Trees score:0.8305929247633284

Βήμα 18c

Βλέπουμε ότι ο μόνο Decision Tree ταξινομητής βελτιώνεται περίπου κατά 5 ποσοστιαίες μονάδες με χρήση του bagging, γεγονός που οφείλεται στο ότι είναι αρκετά πιο ασταθής σε σχέση με τους υπόλοιπους (SVM, knn).