ΜΥΕ046 – Υπολογιστική Όραση: Άνοιξη 2023

3η Σειρά Ασκήσεων: 50% του συνολικού βαθμού

Διδάσκων: Άγγελος Γιώτης

- ΠΑΡΑΔΟΣΗ: **Σάββατο, 10 Ιουνίου, 2023 23:59**

Γενικές Οδηγίες

Απαντήστε στα παρακάτω ζητήματα χρησιμοποιώντας Python στο συνημμένο σημειωματάριο Jupyter και ακολουθήστε τις παρακάτω οδηγίες:

- Οι ασκήσεις είναι **ατομικές** - δεν επιτρέπεται η μεταξύ σας συνεργασία για την υλοποίηση/παράδοσή τους.
- **Δεν** επιτρέπεται να χρησιμοποιήσετε κώδικα που τυχόν θα βρείτε στο διαδίκτυο (είτε αυτούσιο, είτε **παραγόμενο από AI**). Η χρήση κώδικα τρίτων θα έχει σαν αποτέλεσμα τον αυτόματο μηδενισμό σας.
- Όλες οι λύσεις πρέπει να είναι γραμμένες σε αυτό το σημειωματάριο `Jupyter notebook`.
- **Εάν** ένα ζήτημα περιλαμβάνει θεωρητική ερώτηση, η απάντηση θα **πρέπει** να συμπεριληφθεί στο τέλος του ζητήματος, σε ξεχωριστό "Markdown" κελί.
- Ο κώδικάς σας πρέπει να σχολιαστεί εκτενώς!
- Αφού ολοκληρώσετε (υλοποιήσετε και εκτελέσετε) τις απαντήσεις σας στο σημειωματάριο (notebook), εξαγάγετε το notebook ως PDF και υποβάλετε, τόσο το σημειωματάριο όσο και το PDF (δηλαδή τα αρχεία `.ipynb` και `.pdf`) στο `turnin` του μαθήματος, μαζί με ένα συνοδευτικό αρχείο `onoma.txt` που θα περιέχει το ον/μο σας και τον Α.Μ. σας.
- Οι απαντήσεις θα παραδοθούν με την εντολή: **turnin assignment_3@mye046 onoma.txt assignment3.ipynb assignment3.pdf**
- Μπορείτε να χρησιμοποιήσετε βασικά πακέτα γραμμικής άλγεβρας (π.χ. `NumPy`, `SciPy`), αλλά δεν επιτρέπεται να χρησιμοποιείτε τα πακέτα/βιβλιοθήκες που επιλύουν άμεσα τα προβλήματα, εκτός και αν αναφέρεται διαφορετικά η χρήση συγκεκριμένου πακέτου σε κάποιο ζήτημα. Αν δεν είστε βέβαιοι για κάποιο συγκεκριμένο πακέτο/βιβλιοθήκη ή συνάρτηση που θα χρησιμοποιήσετε, μη διστάσετε να ρωτήσετε τον διδάσκοντα.

- Συνιστάται ιδιαίτερα να αρχίσετε να εργάζεστε στις ασκήσεις σας το συντομότερο δυνατό!

Late Policy: Εργασίες που υποβάλλονται καθυστερημένα θα λαμβάνουν μείωση βαθμού 10% για κάθε 24 ώρες καθυστέρησης. Οι εργασίες δεν θα γίνονται δεκτές 96 ώρες (4 ημέρες) μετά την προθεσμία παράδοσης. Για παράδειγμα, παράδοση της εργασίας 2 ημέρες μετά την προθεσμία βαθμολογείται με άριστα το 40 (από 50).

Άσκηση 1: Μηχανική Μάθηση [25 μονάδες]

Στην άσκηση αυτή θα υλοποιήσετε μια σειρά από τεχνικές μηχανικής μάθησης με εφαρμογή στην επίλυση προβλημάτων υπολογιστικής όρασης.

Ζήτημα 1.1: Αρχική Εγκατάσταση

Θα χρησιμοποιήσουμε την ενότητα Scikit-learn (Sklearn) για αυτή την άσκηση. Είναι μια από τις πιο χρήσιμες και ισχυρές βιβλιοθήκες για μηχανική μάθηση στην Python. Παρέχει μια επιλογή αποτελεσματικών εργαλείων για μηχανική μάθηση και στατιστική μοντελοποίηση, συμπεριλαμβανομένης της ταξινόμησης (classification), της παλινδρόμησης (regression), της ομαδοποίησης (clustering) και της μείωσης διάστασης (dimensionality reduction). Αυτό το πακέτο, το οποίο είναι σε μεγάλο βαθμό γραμμένο σε Python, βασίζεται στις βιβλιοθήκες NumPy, SciPy και Matplotlib.

Αρχικά καλούμε/εγκαθιστούμε τη βασική μονάδα της βιβλιοθήκης sklearn.

```
import sklearn
sklearn.__version__
```

```
'1.2.1'
```

Ζήτημα 1.2: Λήψη συνόλου δεδομένων χειρόγραφων ψηφίων "MNIST" και απεικόνιση παραδειγμάτων [2 μονάδες]

Η βάση δεδομένων MNIST (Modified National Institute of Standards and Technology database) είναι ένα αρκετά διαδεδομένο σύνολο δεδομένων που αποτελείται από εικόνες χειρόγραφων ψηφίων, διαστάσεων 28x28 σε κλίμακα του γκρι. Για αυτό το ζήτημα, θα χρησιμοποιήσουμε το πακέτο Sklearn για να κάνουμε ταξινόμηση μηχανικής μάθησης στο σύνολο δεδομένων MNIST.

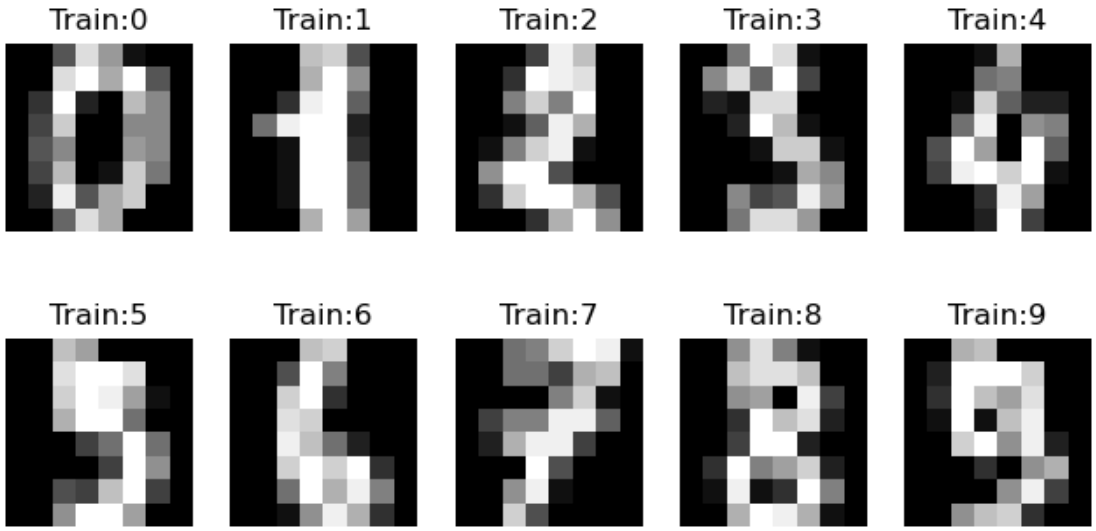
Το Sklearn παρέχει μια βάση δεδομένων MNIST χαμηλότερης ανάλυσης με εικόνες ψηφίων 8x8 pixel. Το πεδίο (attribute) `images` του συνόλου δεδομένων, αποθηκεύει πίνακες 8x8 τιμών κλίμακας του γκρι για κάθε εικόνα. Το πεδίο (attribute) `target` του συνόλου δεδομένων αποθηκεύει το ψηφίο που αντιπροσωπεύει κάθε εικόνα. Ολοκληρώστε τη συνάρτηση 'plot_mnist_sample()' για να απεικονίσετε σε ένα σχήμα 2x5 ένα δείγμα εικόνας από κάθε μια κατηγορία (κάθε πλαίσιο του 2x5 σχήματος αντιστοιχεί σε ένα ψηφίο/εικόνα μιας κατηγορίας). Η παρακάτω εικόνα δίνει ένα παράδειγμα: mnist

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets

# Download MNIST Dataset from Sklearn
digits = datasets.load_digits()

# Print to show there are 1797 images (8 by 8)
print("Images Shape" , digits.images.shape)

# Print to show there are 1797 image data (8 by 8 images for a
dimensionality of 64)
print("Image Data Shape" , digits.data.shape)

# Print to show there are 1797 labels (integers from 0-9)
print("Label Data Shape", digits.target.shape)
```

```
Images Shape (1797, 8, 8)
Image Data Shape (1797, 64)
Label Data Shape (1797,)
```

```
def plot_mnist_sample(digits):
    """
    This function plots a sample image for each category,
    The result is a figure with 2x5 grid of images.

    """
    #Set the size of the overall figure in inches, so that its size is
similar to the one provided..
    #..by the assignment. Specifically figsize=(8,4) sets the figure
size to be 8 by 4 inches.
    fig = plt.figure(figsize=(8,4))
    #Since we want to create a 2x5 grid, we need to assign the
appropriate parameters.
    columns = 5 #Number of columns
    rows = 2 #Number of rows

    #Since the total amount of number samples we need is 10 [0-9]..
    #..we need 10 loops. The code contained in the for loop should
be..
    #.. used x10 times, for each subplot/ number sample we want to
display.
    for i in range(1,11):
        #digits.images is an array that contains 1797 images of
numbers from 0 to 9.
        #Each one is 8 by 8.
        #digits.target is an array that contains 1797 values ranging
from 0 to 9.
        #Each n'th digits.target value is equal to the number the
corresponding..
```

```
        #..digits.images image represents.
        #So since digits.target[0] is equal to 0, digits.images[0]
should contain..
        #.. an image with the number of 0.
        #So with each i'th iteration we need to search for the correct
digits.target value..
        #.. in order to retrieve the correct digits.images image
        #For example, in the first loop we want the number 0, in the
next 1 and so on
        #In order to achieve that we search for the indexes that point
to digits.target cells..
        #..that contain the value we need in each loop. But indexes
array contains many indexes..
        #..since we just need 1, we'll always keep the indexes[0]
        #So [index for index, value in enumerate(digits.target) if
value == (i - 1)] does..
        #.. exactly that. It returns all the indexes to the value we
want in each loop.
        indexes = [index for index, value in enumerate(digits.target)
if value == (i - 1)]
        #Now that we've got our index we get the corresponding digit
image as well using..
        #..digits.images[indexes[0]]
        img = digits.images[indexes[0]]
        fig.add_subplot(rows, columns, i) #Add a new sublot to the
figure with the new image
        plt.imshow(img, cmap = 'gray') #Show image in grayscale
        #Since each subplot should have a title we use..
        #.."Train:" + str(i-1)
        title = "Train:" + str(i-1)
        plt.title(title) #Add title to the current subplot
        plt.axis('off') #Don't show the axes labels
    plt.show()

# PLOT CODE: DO NOT CHANGE
# This code is for you to plot the results.

plot_mnist_sample(digits)
```



Ζήτημα 1.3: Αναγνώριση χειρόγραφων ψηφίων με Sklearn [4 μονάδες]

Ένα από τα πιο ενδιαφέροντα πράγματα σχετικά με τη βιβλιοθήκη Sklearn είναι ότι παρέχει έναν εύκολο τρόπο δημιουργίας και κλήσης/χρήσης διαφορετικών μοντέλων. Σε αυτό το μέρος της άσκησης, θα αποκτήσετε εμπειρία με τα μοντέλα ταξινόμησης `LogisticRegressionClassifier` (ταξινόμηση με λογιστική παλινδρόμηση) και `kNNCalssifier` (ταξινόμηση με τη μέθοδο κ-κοντινότερων γειτόνων).

Ακολουθούν αρχικά 2 βοηθητικές ρουτίνες: 1) μια *ρουτίνα δημιουργίας* mini-batches (παρτίδων) δεδομένων εκπαίδευσης και ελέγχου, αντίστοιχα, 2) μια *ρουτίνα ελέγχου* του εκάστοτε ταξινομητή στις παρτίδες δεδομένων (train/test): α) RandomClassifier(), β) LogisticRegressionClassifier(), γ) kNNClassifier καθώς και των ταξινομητών των ζητημάτων 1.4, 1.5, 1.6 και 2.2, 2.4, 2.5. Στη συνέχεια η συνάρτηση train_test_split() διαχωρίζει το σύνολο δεδομένων σε δεδομένα μάθησης (training set: <X_train, y_train>) και ελέγχου (test set: <X_test, y_test>).

Ο κώδικας που ακολουθεί στη συνέχεια ορίζει κάποιες συναρτήσεις/μεθόδους για 3 ταξινομητές: 2 για τον RandomClassifier() και 3 μεθόδους για τους ταξινομητές LogisticRegressionClassifier() και kNNClassifier(). Οι 2 τελευταίες κλάσσεις έχουν μια μέθοδο **init** για αρχικοποίηση, μια μέθοδο **train** για την εκπαίδευση του μοντέλου και μια μέθοδο **call** για την πραγματοποίηση προβλέψεων. Πρέπει να συμπληρώσετε τα μέρη κώδικα που λείπουν από τις κλάσεις LogisticRegressionClassifier και kNNClassifier, χρησιμοποιώντας τις υλοποιήσεις `LogisticRegression` και `KNeighborsClassifier` από το Sklearn.

```
# DO NOT CHANGE
#### Some helper functions are given below####
def DataBatch(data, label, batchsize, shuffle=True):
    """
```

```
    This function provides a generator for batches of data that
    yields data (batchsize, 3, 32, 32) and labels (batchsize)
    if shuffle, it will load batches in a random order
    """
    n = data.shape[0]
    if shuffle:
        index = np.random.permutation(n)
    else:
        index = np.arange(n)
    for i in range(int(np.ceil(n/batchsize))):
        inds = index[i*batchsize : min(n,(i+1)*batchsize)]
        yield data[inds], label[inds]

def test(testData, testLabels, classifier):
    """
    Call this function to test the accuracy of a classifier
    """
    batchsize=50
    correct=0.
    for data,label in
DataBatch(testData,testLabels,batchsize,shuffle=False):
        prediction = classifier(data)
        correct += np.sum(prediction==label)
    return correct/testData.shape[0]*100

# DO NOT CHANGE
# Split data into 90% train and 10% test subsets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    digits.images.reshape((len(digits.images), -1)), digits.target,
test_size=0.1, shuffle=False)

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

class RandomClassifier():
    """
    This is a sample classifier.
    given an input it outputs a random class
    """
    def __init__(self, classes=10):
        self.classes=classes
    def __call__(self, x):
        return np.random.randint(self.classes, size=x.shape[0])

class LogisticRegressionClassifier():
    def __init__(self, sol='liblinear'):
        """
        Initialize Logistic Regression model.
```

```
        Inputs:
        sol: Solver method that the Logistic Regression model would
use for optimization
        """
        self.model = LogisticRegression(solver = sol)


    def train(self, trainData, trainLabels):
        """
        Train your model with image data and corresponding labels.

        Inputs:
        trainData: Training images (N,64)
        trainLabels: Labels (N,)
        """
        self.model.fit(trainData, trainLabels)


    def __call__(self, x):
        """
        Predict the trained model on test data.

        Inputs:
        x: Test images (N,64)

        Returns:
        predicted labels (N,)
        """
        return self.model.predict(x)


class kNNClassifier():
    def __init__(self, k=3, algorithm='brute'):
        """
        Initialize KNN model.

        Inputs:
        k: number of neighbors involved in voting
        algorithm: Algorithm used to compute nearest neighbors
        """
        self.model = KNeighborsClassifier(n_neighbors = k, algorithm =
algorithm)

    def train(self, trainData, trainLabels):
        """
        Train your model with image data and corresponding labels.

        Inputs:
        trainData: Training images (N,64)
```

```
        trainLabels: Labels (N,)
        """
        self.model.fit(trainData, trainLabels)


    def __call__(self, x):
        """
        Predict the trained model on test data.

        Inputs:
        x: Test images (N,64)

        Returns:
        predicted labels (N,)
        """
        return self.model.predict(x)

# TEST CODE: DO NOT CHANGE
randomClassifierX = RandomClassifier()
print ('Random classifier accuracy: %f'%test(X_test, y_test,
randomClassifierX))
```

```
Random classifier accuracy: 8.888889
```

```
# TEST CODE: DO NOT CHANGE
# TEST LogisticRegressionClassifier

lrClassifierX = LogisticRegressionClassifier()
lrClassifierX.train(X_train, y_train)
print ('Logistic Regression Classifier classifier accuracy:
%f'%test(X_test, y_test, lrClassifierX))
```

```
Logistic Regression Classifier classifier accuracy: 93.888889
```

```
# TEST kNNClassifier
knnClassifierX = kNNClassifier(k = 3, algorithm = "brute")
knnClassifierX.train(X_train, y_train)
print('k-NN Classifier accuracy: %f' % test(X_test, y_test,
knnClassifierX))
```

```
k-NN Classifier accuracy: 96.666667
```

Ζήτημα 1.4: Πίνακας Σύγχυσης [4 μονάδες]

Ένας πίνακας σύγχυσης είναι ένας 2Δ πίνακας που χρησιμοποιείται συχνά για να περιγράψει την απόδοση ενός μοντέλου ταξινόμησης σε ένα σύνολο δεδομένων ελέγχου/δοκιμής (test data) για τα οποία είναι γνωστές οι πραγματικές τιμές (known labels). Εδώ θα υλοποιήσετε τη συνάρτηση που υπολογίζει τον πίνακα σύγχυσης για έναν ταξινομητή. Ο πίνακας (M) πρέπει να είναι $n \times n$ όπου n είναι ο αριθμός των κλάσεων/κατηγοριών. Η καταχώριση `M[i,j]` πρέπει να περιέχει το ποσοστό/λόγο των εικόνων της κατηγορίας `i`

που ταξινομήθηκε ως κατηγορία `j`. Αν οι καταχωρήσεις `M[i,j]` έχουν υπολογιστεί σωστά, τότε τα στοιχεία `M[k,j]` κατά μήκος μιας γραμμής k για $j \neq k$ (εκτός της κύριας διαγωνίου) αναμένεται να αντιστοιχούν σε "ψευδώς αρνητικές" ταξινομήσεις (false negatives), ενώ τα στοιχεία `M[i,k]` κατά μήκος μιας στήλης k για $i \neq k$ (εκτός της κύριας διαγωνίου) αναμένεται να αντιστοιχούν σε "ψευδώς θετικές" ταξινομήσεις (false positives). Το ακόλουθο παράδειγμα δείχνει τον πίνακα σύγχυσης για τον `RandomClassifier` ταξινομητή. Ο στόχος σας είναι να σχεδιάσετε τα αποτελέσματα για τον `LogisticRegressionClassifier` και τον `kNNClassifier` ταξινομητή.

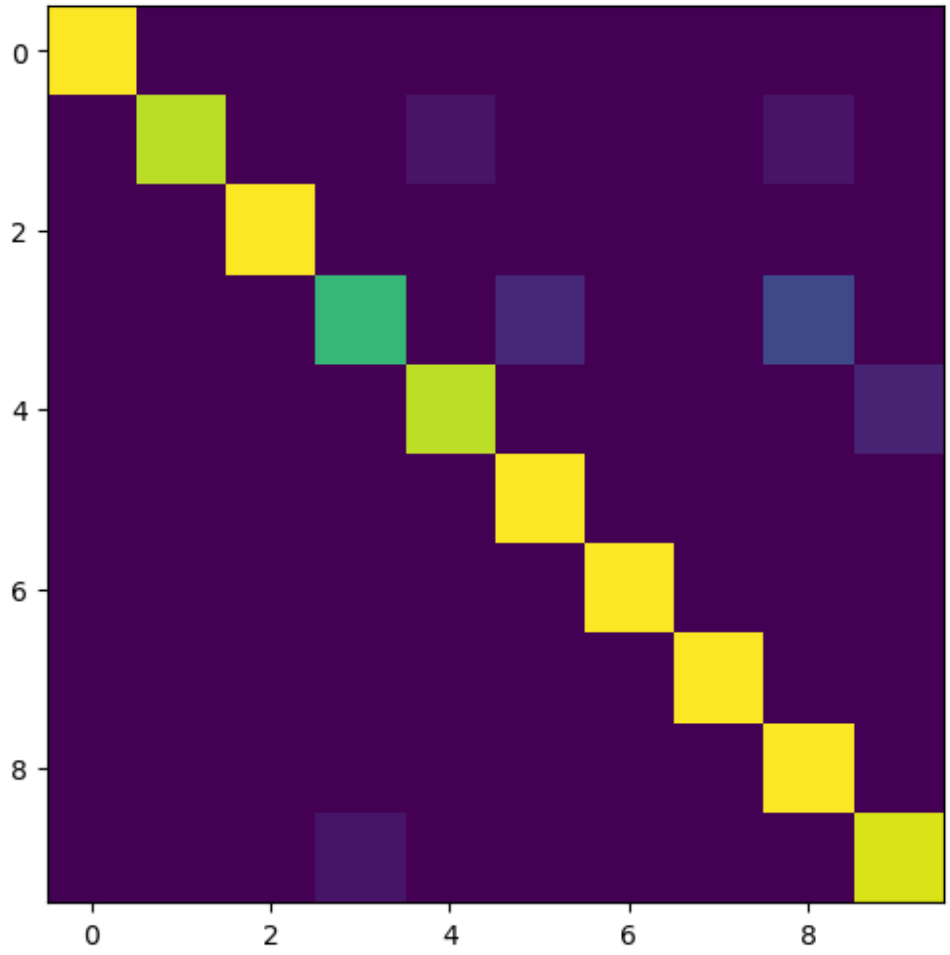
confusion

```
from tqdm import tqdm

def Confusion(testData, testLabels, classifier):
    batchsize=50
    correct=0
    M=np.zeros((10,10))
    num=testData.shape[0]/batchsize
    count=0
    acc=0

    for data,label in
tqdm(DataBatch(testData,testLabels,batchsize,shuffle=False),total=len(
testData)//batchsize):
        #Initialize an empty prediction matrix. It will hold the
predicted labels.
        prediction = []
        #Call the input classifier and get the predicted labels for..
        #..the currect batch of data
        prediction = classifier(data)
        #We want to access both matrices (label,prediction) pair by
pair..
        #..simultaneously.In the 1st loop we gain access to both 1st
elements,..
        #..in the 2nd loop to both 2nd elements and so on. In order to
do that..
        #..we initialize two indeces label_index, prediction_index to
0.
        #Each index keeps track of the element we are traversing in
the current loop.
        #Trey are incremented at the end of each loop so that we can
move on to..
        #..the next pair.
        #By doing so, we ultimately iterate through the M matrix
cells, whose coordinates..
        #..are the (label,prediction) pairs.
        #In order to create the confusion matrix we increase the value
of each M cell..
        #..we access by 1. Later on we will divide said values..
```

```
        #..in order to get the percentages.
        label_index = prediction_index = 0 #Set indeces to 0
        while(label_index < len(label) and prediction_index <
len(prediction)):
            current_label = label[label_index] #Get current label
            current_prediction = prediction[prediction_index] #Get
current prediction
            M[current_label][current_prediction] += 1 #Increase M cell
by 1.
            #If the classifier made the correct prediction and it
matches the label
            if current_label == current_prediction:
                correct += 1 #We increase the number of correct
prediction by 1.
            label_index += 1 #Get next label index
            prediction_index += 1 #Get next prediciton index
        #Count should hold the amount of all batch items.
        #So that we can divide the correct predictions by the total
amount of..
        #.. the batch items.
        count += len(label)

    #Initialize a matrix of size M.shape[0] to 0.
    row_totals = np.zeros(M.shape[0])
    for i in range(M.shape[0]):
        row_totals[i] = np.sum(M[i]) #Keep the sum of each M row

    #Divide each matrix element by the total of the row it belongs to.
    for i in range(M.shape[0]):
        for j in range(M.shape[1]):
            if row_totals[i] != 0: #Safety measure for the denominator
                M[i][j] = M[i][j] / row_totals[i] * 100

    # Calculate accuracy
    #If count is different than 0 we calculate the accuracy.
    #Since we need to divide by the count, we use the if condition..
    #..as a safety measure.
    if count != 0:
        acc = correct / count * 100.0

    return M, acc

def VisualizeConfussion(M):
    plt.figure(figsize=(14, 6))
    plt.imshow(M)
    plt.show()
    print(np.round(M,2))

# TEST/PLOT CODE: DO NOT CHANGE
# TEST LogisticRegressionClassifier
```

```
M,acc = Confusion(X_test, y_test, lrClassifierX)
VisualizeConfussion(M)
```

```
4it [00:00, 2012.14it/s]
```

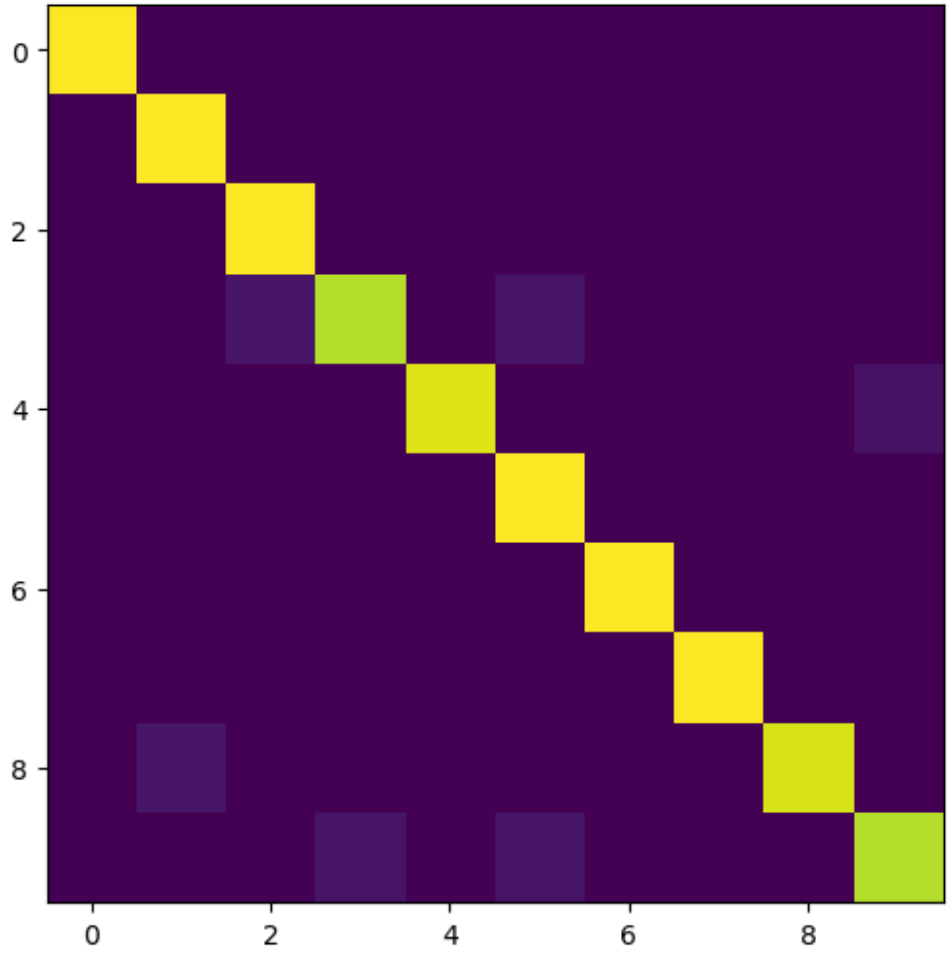



```
[[100.     0.     0.     0.     0.     0.     0.     0.     0.     0.
]
 [  0.    89.47   0.     0.     5.26   0.     0.     0.     5.26   0.
]
 [  0.     0.   100.     0.     0.     0.     0.     0.     0.     0.
]
 [  0.     0.     0.    66.67   0.    11.11   0.     0.    22.22   0.
]
 [  0.     0.     0.     0.    90.     0.     0.     0.     0.    10.
]
 [  0.     0.     0.     0.     0.   100.     0.     0.     0.     0.
]
 [  0.     0.     0.     0.     0.     0.   100.     0.     0.     0.
]
 [  0.     0.     0.     0.     0.     0.     0.   100.     0.     0.
```

```
]
 [  0.     0.     0.     0.     0.     0.     0.     0.   100.     0.
]
 [  0.     0.     0.     5.56   0.     0.     0.     0.     0.
94.44]]
```

```
# TEST/PLOT CODE: DO NOT CHANGE
# TEST kNNClassifier

M,acc = Confusion(X_test, y_test, knnClassifierX)
VisualizeConfussion(M)
```

```
4it [00:00,  7.19it/s]
```




```
[[100.     0.     0.     0.     0.     0.     0.     0.     0.     0.
]
 [  0.   100.     0.     0.     0.     0.     0.     0.     0.     0.
]
 [  0.     0.   100.     0.     0.     0.     0.     0.     0.     0.
]
 [  0.     0.     5.56  88.89   0.     5.56   0.     0.     0.     0.
```

```
]
 [  0.     0.     0.     0.    95.     0.     0.     0.     0.     5.
]
 [  0.     0.     0.     0.     0.   100.     0.     0.     0.     0.
]
 [  0.     0.     0.     0.     0.     0.   100.     0.     0.     0.
]
 [  0.     0.     0.     0.     0.     0.     0.   100.     0.     0.
]
 [  0.     5.88   0.     0.     0.     0.     0.     0.    94.12   0.
]
 [  0.     0.     0.     5.56   0.     5.56   0.     0.     0.
88.89]]
```

Ζήτημα 1.5: κ-Κοντινότεροι Γείτονες (k-Nearest Neighbors/kNN) [7 μονάδες]

Για αυτό το πρόβλημα, θα ολοκληρώσετε έναν απλό ταξινομητή kNN χωρίς χρήση του πακέτου Sklearn. Η μέτρηση της απόστασης είναι η Ευκλείδεια απόσταση (L2 norm) στον χώρο των pixel. Μπορείτε να χρησιμοποιήσετε τη συνάρτηση **np.linalg.norm** για να υπολογίσετε την απόσταση. Το k αναφέρεται στον αριθμό των γειτόνων που συμμετέχουν στην ψηφοφορία για την ομάδα/κλάση.

```
class kNNClassifier_v1_5():
    def __init__(self, k=3):
        self.k=k

    def train(self, trainData, trainLabels):
        self.X_train = trainData
        self.y_train = trainLabels

    def __call__(self, X):
        """
        Predict the labels for the input data using KNN method.

        Inputs:
        X: Test images (N,64)

        Returns:
        predicted labels (N,)
        """
        #Initialize an empty prediction matrix. It will hold the
predicted labels.
        prediction = []
        for x in X: #for each test image
            #Calculate the euclidean distance between the training
images and the..
            #..x test image. The parameter axis = 1 is used so that
the computation is,
            #..done along the whole row. So the end result will be an
```

```
array of euclidean..
            #..distances.
            eucd = np.linalg.norm(self.X_train - x, axis = 1)
            #Get the sorted indices of the nearest neighbors using
np.argsort(eucd).
            #Out of all of those, we need to keep only the first k,
since they have..
            #..the shortest euclidean distance.
            nearest_indices = np.argsort(eucd)
            kneigh_indices = [] #Initialize empty array.It will hold
the first k indices
            for i in range(self.k): #traverse the first k elements
                kneigh_indices.append(nearest_indices[i]) #Add each
one

            #Get the labels of those k closest neighbors, using their
indices
            neighlabels = self.y_train[kneigh_indices]
            uniques = [] #Holds unique labels
            counts = [] #Holds the amount of the corresponding unique
label
            index = 0
            #Now we are gonna keep track of all the unique labels and
the number of time..
            #..each one appears.
            for neighlabel in neighlabels: #For each label
                #If we haven't seen the current label yet, it's unique
                #Add it to the uniques matrix and append 1 to the
corresponding..
                #..counts matrix cell, since it is the first time we
encounter it.
                if neighlabel not in uniques:
                    uniques.append(neighlabel)
                    counts.append(1)
                else: #We've already encountered this label. Increase
count by 1.
                    lindex = uniques.index(neighlabel)
                    counts[lindex] += 1
            #Finally, we choose the label with the highest number of
encounters..
            #..and we append it to the list of predictions.
            max_encounters = max(counts) #Find the max number of
encounters
            #Get index of the most encountered label
            cindex = np.argmax(max_encounters)
            #Get the most encountered label
            predicted_label = uniques[cindex]
            prediction.append(predicted_label) #Add it to the list

        return prediction
```

```
# TEST/PLOT CODE: DO NOT CHANGE
# TEST kNNClassifierManual

knnClassifierManualX = kNNClassifier_v1_5()
knnClassifierManualX.train(X_train, y_train)
print ('kNN classifier accuracy: %f'%test(X_test, y_test,
knnClassifierManualX))
```

```
kNN classifier accuracy: 96.666667
```

Ζήτημα 1.6: PCA + κ-κοντινότεροι γείτονες (PCA/k-NN) [8 μονάδες]

Σε αυτό το ζήτημα θα εφαρμόσετε έναν απλό ταξινομητή kNN, αλλά στον χώρο PCA, δηλαδή όχι τον χώρο των πίξελ, αλλά αυτόν που προκύπτει μετά από ανάλυση σε πρωτεύουσες συνιστώσες των εικόνων του συνόλου εκπαίδευσης (για k=3 και 25 πρωτεύουσες συνιστώσες).

Θα πρέπει να υλοποιήσετε μόνοι σας την PCA χρησιμοποιώντας "Singular Value Decomposition (SVD)". Η χρήση του `sklearn.decomposition.PCA` ή οποιουδήποτε άλλου πακέτου που υλοποιεί άμεσα μετασχηματισμούς PCA θα οδηγήσει σε μείωση μονάδων.

Μπορείτε να χρησιμοποιήσετε την προηγούμενη υλοποίηση του ταξινομητή kNN σε αυτό το ζήτημα.

Είναι ο χρόνος ελέγχου για τον ταξινομητή PCA-kNN μεγαλύτερος ή μικρότερος από αυτόν για τον ταξινομητή kNN; Εφόσον διαφέρει, σχολιάστε γιατί στο τέλος της άσκησης.

```
def svd(A):
    #SVD stands for Singular Value Decomposition
    #Given an array, it will return its decomposition
    #The decomposition of the array is A = U * singular_values * V
    #Essentially, the SVD "breaks" A into 3 pieces
    #U:Left singular vectors of A
    #singular_values: Singular values of A
    #V:Right singular vectors of A

    #Firstly, initialize the arrays that will be returned to zero
    #Since they stem from the decomposition of A, their dimensions
will be..
    #..the same
    U = np.zeros((A.shape[0],A.shape[1]))
    singular_values = np.zeros(A.shape[0])
    V = np.zeros((A.shape[0],A.shape[1]))

    #Call np.linalg.svd(A) to decompose A and assign the 3 components
to..
    #..U, singular_values, V
    U, singular_values, V = np.linalg.svd(A)
```

```
    #V.T is the transpose matrix of V.
    return U, singular_values, V.T



class PCAKNNClassifer():
    def __init__(self, components=25, k=3):
        """
        Initialize PCA kNN classifier

        Inputs:
        components: number of principal components
        k: number of neighbors involved in voting
        """
        self.components = components
        self.k = k

    def train(self, trainData, trainLabels):
        """
        Train your model with image data and corresponding labels.

        Inputs:
        trainData: Training images (N,64)
        trainLabels: Labels (N,)
        """
        self.trainData = trainData
        self.trainLabels = trainLabels

        #Since we need to perform the svd on centered data, we need
to..
        #..subtract their mean from them.
        #Subtracting the mean from the data can remove any unwanted
bias..
        #..improve the accuracy of our classifier

        #Firstly, we calculate the mean
        #The axis=0 parameter is essential. Without it the accuracy
decreases..
        #..drastically. Without it  np.nean would return a single
value for..
        #..the entirety of the matrix. Whereas, now mean will be
performed..
        #..along the columns resulting in a matrix of means
        mean = np.mean(trainData, axis=0)

        #Now in order to center the data we subtract the mean from it
        X_hat = self.trainData - mean

        # Perform SVD on centered data (mean-deviation form of data
matrix)
```

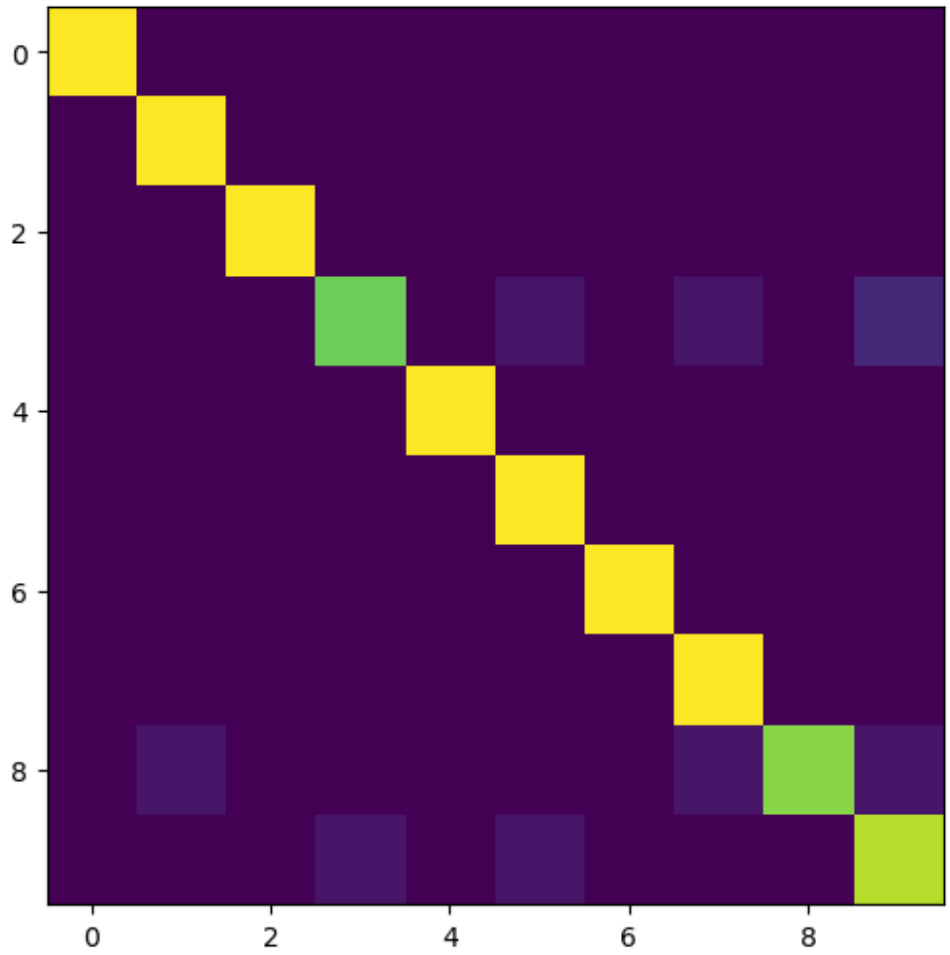
```
        U, D, V = svd(X_hat)

        #In order to be able to access the U,D,V that we've just got..
        #..from the svd we need to assign them as class variable.
        #So that the other class methods can "see" them too.

        #Firstly, we initialize them to 0
        self.U = np.zeros((U.shape[0],self.components))
        self.D = np.zeros(self.components)
        self.V = np.zeros((V.shape[0],self.components))

        #Then we assign each one to its corresponding class variable
        #Since the 3 components have different sizes they should be..
        #..handled separately
        for i in range(self.U.shape[0]):
            for j in range(self.components):
                self.U[i][j] = U[i][j]

        for i in range(self.components):
            self.D[i] = D[i]

        for i in range(self.V.shape[0]):
            for j in range(self.components):
                self.V[i][j] = V[i][j]


        #Lastly, since we've got the 3 principal components of our
data..
        #..we need to get the train data for the pca
        #In order to do that we need to multiply the 2 matrices..
        #..X_hat and self.V
        #Note: in order to be able to multiply two matrices..
        #..the column number of the 1st need to be equal to the..
        #..row number of the 2nd
        #That's why we first initialize it to None
        self.pca_trainData = None
        #np.matmul() function takes two matrices as arguments and..
        #..returns the result of their multiplication
        self.pca_trainData = np.matmul(X_hat, self.V)

    def __call__(self, x):
        """
        Predict the trained model on test data.

        Inputs:
        x: Test images (N,64)

        Returns:
        predicted labels (N,)
        """
```

```
        self.x = x #Assign x as class variable
        #In order to improve the accuracy we need to center the data..
        #..by subtracting the mean from it
        #Calculate mean using np.mean()
        mean = np.mean(self.x, axis=0)
        #Center the data by subtracting the data
        X_hat = self.x - mean
        #Initialize x_pca. it will hold the result of the matrix
multiplication..
        #..later on
        x_pca = None
        x_pca = np.matmul(X_hat, self.V) #Calculate matrix
multiplication
        self.x_pca = x_pca #Assign the result as class variable

        #Lastly, we use the knn classifier that was implemented in
1.5..
        #..with the new data from the pca
        knnClassifierManualX = kNNClassifier_v1_5(self.k)
        knnClassifierManualX.train(self.pca_trainData,
self.trainLabels)
        prediction = knnClassifierManualX(self.x_pca)
        return prediction


# test your classifier with only the first 100 training examples (use
this
# while debugging)
pcaknnClassiferX = PCAKNNClassifer()
pcaknnClassiferX.train(X_train[:100], y_train[:100])
print ('PCA-kNN classifier accuracy: %f'%test(X_test, y_test,
pcaknnClassiferX))
```

```
PCA-kNN classifier accuracy: 87.777778
```

```
# test your classifier with all the training examples
pcaknnClassifer = PCAKNNClassifer()
pcaknnClassifer.train(X_train, y_train)
# display confusion matrix for your PCA KNN classifier with all the
training examples
M_pca, acc = Confusion(X_test, y_test, pcaknnClassifer)

# Display the accuracy and visualize the confusion matrix
print ('PCA-kNN classifier accuracy: %f'%test(X_test, y_test,
pcaknnClassifer))
VisualizeConfussion(M_pca)
```

```
4it [00:00, 46.54it/s]

PCA-kNN classifier accuracy: 95.000000
```




```
[[100.     0.     0.     0.     0.     0.     0.     0.     0.     0.
]
 [  0.   100.     0.     0.     0.     0.     0.     0.     0.     0.
]
 [  0.     0.   100.     0.     0.     0.     0.     0.     0.     0.
]
 [  0.     0.     0.    77.78   0.     5.56   0.     5.56   0.
11.11]
 [  0.     0.     0.     0.   100.     0.     0.     0.     0.     0.
]
 [  0.     0.     0.     0.     0.   100.     0.     0.     0.     0.
]
 [  0.     0.     0.     0.     0.     0.   100.     0.     0.     0.
]
 [  0.     0.     0.     0.     0.     0.     0.   100.     0.     0.
]
 [  0.     5.88   0.     0.     0.     0.     0.     5.88  82.35
5.88]
 [  0.     0.     0.     5.56   0.     5.56   0.     0.     0.
88.89]]
```

- Σχολιασμός του χρόνου εκτέλεσης PCA-kNN σε σχέση με τον kNN.

""" The PCA-kNN has a significantly better time than the simple kNN. This becomes apparent by the fact that the it/s of PCA-kNN are way higher than those of the simple kNN. It/s means the number of tasks or operations that an algorithm can perform per second. So higher it/s rate means better processing capability by the PCA-kNN algorithm. The improvement in performance has to do with the reduction in dimensionality of the data, provided by the PCA-kNN. Whereas simple kNN has to process the data pixel by pixel in order to find the nearest neighbors. So, since the simple kNN does not reduce the dimensionality it can have very high processing complexity, especially when we set the number of k neighbors to higher values. """

Άσκηση 2: Βαθιά Μάθηση [25 μονάδες]

Ζήτημα 2.1 Αρχική Εγκατάσταση (απεικόνιση παραδειγμάτων) [1 μονάδα]

- **Τοπικά** (jupyter): Ακολουθήστε τις οδηγίες στη διεύθυνση https://pytorch.org/get-started/locally/ για να εγκαταστήσετε την PyTorch τοπικά στον υπολογιστή σας. Για παράδειγμα, αφού δημιουργήσετε και ενεργοποιήσετε κάποιο εικονικό περιβάλλον anaconda με τις εντολές: π.χ. `(base)$ conda create -n askisi3`, `(base)$ conda activate askisi3`, η εντολή `(askisi3)$ conda install pytorch torchvision torchaudio cpuonly -c pytorch` εγκαθιστά την βιβλιοθήκη "PyTorch" σε περιβάλλον Linux/Windows χωρίς GPU υποστήριξη.

Προσοχή σε αυτό το σημείο, αν τρέχετε την άσκηση τοπικά σε jupyter, εκτός της εγκατάστασης του PyTorch, θα χρειαστούν ξανά και κάποιες βιβλιοθήκες `matplotlib`, `scipy`, `tqdm` και `sklearn` (όπως και στην 1η άσκηση), μέσα στο περιβάλλον 'askisi3', πριν ανοίξετε το jupyter: `(askisi3)$ conda install matplotlib tqdm scipy` και `(askisi3)$ conda install -c anaconda scikit-learn`. Αυτό χρειάζεται διότι σε ορισμένες περιπτώσεις, αφού εγκαταστήσετε τις βιβλιοθήκες που απαιτούνται, πρέπει να εξασφαλίσετε ότι ο *Python Kernel* αναγνωρίζει την προϋπάρχουσα εγκατάσταση (PyTorch, matplotlib, tqdm, κτλ.). Τέλος, χρειάζεται να εγκαταστήσετε το jupyter ή jupyterlab μέσω του περιβάλλοντος conda: `(askisi3)$ conda install jupyter` και μετά να εκτελέσετε `(askisi3)$ jupyter notebook` για να ανοίξετε το jupyter με τη σωστή εγκατάσταση. Αν όλα έχουν γίνει σωστά, θα πρέπει ο *Python Kernel* να βλέπει όλα τα 'modules' που χρειάζεστε στη 2η άσκηση. Διαφορετικά, μπορείτε να εγκαταστήσετε εξ' αρχής όλες τις βιβλιοθήκες, από την αρχή υλοποίησης της 3ης σειράς ασκήσεων, μέσα στο εικονικό περιβάλλον *askisi3* ώστε να μην είναι απαραίτητη εκ νέου η εγκατάσταση των βιβλιοθηκών που θα χρειαστούν στη 2η άσκηση.

- **Colab**: **Αν** χρησιμοποιείτε google colab, τότε δεν θα χρειαστεί λογικά κάποιο βήμα εγκατάστασης. Αν ωστόσο σας παρουσιαστεί κάποιο πρόβλημα με απουσία πακέτου, π.χ. "ModuleNotFoundError - torchvision", τότε μπορείτε απλώς να το εγκαταστήσετε με χρήση του εργαλείου `pip`

εκτελώντας την αντίστοιχη εντολή (π.χ. "!pip install torchvision") σε ένα νέο κελί του notebook.

Σημείωση: Δεν θα είναι απαραίτητη η χρήση GPU για αυτήν την άσκηση, γι' αυτό μην ανησυχείτε αν δεν έχετε ρυθμίσει την εγκατάσταση με υποστήριξη GPU. Επιπλέον, η εγκατάσταση με υποστήριξη GPU είναι συχνά πιο δύσκολη στη διαμόρφωση, γι' αυτό και προτείνεται να εγκαταστήσετε μόνο την έκδοση CPU. Ο Διδάσκων δεν θα παρέχει καμία υποστήριξη που σχετίζεται με GPU ή την CUDA.

Εκτελέστε τις παρακάτω εντολές για να επαληθεύσετε την εγκατάστασή σας (PyTorch).

```
import scipy
import torch.nn as nn
import torch.nn.functional as F
import torch
from torch.autograd import Variable

# create a tensor of 4x3 random-valued array
x = torch.rand(4, 3)
print(x)
```

Σε αυτή την άσκηση, θα χρησιμοποιήσουμε το πλήρες σύνολο δεδομένων της βάσης δεδομένων MNIST με τις εικόνες ψηφίων 28x28 pixel (60.000 εικόνες εκπαίδευσης, 10.000 εικόνες ελέγχου).

Ο κώδικας που ακολουθεί "κατεβάζει" το σύνολο δεδομένων MNIST της κλάσης torchvision.datasets, στο φάκελο `mnist` (του root καταλόγου). Μπορείτε να αλλάξετε τον κατάλογο που δείχνει η μεταβλητή `path` στη διαδρομή που επιθυμείτε. Ενδεικτικό `path` σε περιβάλλον Windows: **path = 'C:/Users/user/Υπολογιστική Όραση/assignments/assignment 3/'**. Στην περίπτωση που εργάζεστε μέσω **colab** μπορεί να χρειαστεί η φόρτωση του καταλόγου στο drive, εκτελώντας `from google.colab import drive` και `drive.mount('/content/gdrive')` και μετά θέτοντας π.χ το **path = '/content/gdrive/assignment3/'**.

- Θα πρέπει να απεικονίσετε σε ένα σχήμα 2x5 ένα τυχαίο παράδειγμα εικόνας που αντιστοιχεί σε κάθε ετικέτα (κατηγορία) από τα δεδομένα εκπαίδευσης (αντίστοιχα του ζητήματος 1.2).

```
import torch
import torchvision.datasets as datasets

# import additional libs in case not already done in 'askisi 1'
import matplotlib.pyplot as plt
import numpy as np

# Define the dataset directory
path = './mnist/'
```

```
# Load the MNIST training dataset
train_dataset = datasets.MNIST(root=path, train=True, download=True)

# Extract the images and labels from the training dataset
X_train = train_dataset.data.numpy()
y_train = train_dataset.targets.numpy()

# Load the MNIST testing dataset
test_dataset = datasets.MNIST(root=path, train=False, download=True)

# Extract the images and labels from the testing dataset
X_test = test_dataset.data.numpy()
y_test = test_dataset.targets.numpy()

def plot_mnist_sample_high_res(X_train, y_train):
    """
    This function plots a sample image for each category,
    The result is a figure with 2x5 grid of images.

    """
    plt.figure()

    """ ==========
    YOUR CODE HERE
    ========== """

# PLOT CODE: DO NOT CHANGE
# This code is for you to plot the results.

plot_mnist_sample_high_res(X_train, y_train)
```

Ζήτημα 2.2: Εκπαίδευση Νευρωνικού Δικτύου με PyTorch [6 μονάδες]

Ακολουθεί ένα τμήμα βοηθητικού κώδικα για την εκπαίδευση των βαθιών νευρωνικών δικτύων (Deep Neural Networks - DNN).

- Ολοκληρώστε τη συνάρτηση train_net() για το παρακάτω DNN.

Θα πρέπει να συμπεριλάβετε τις λειτουργίες της διαδικασίας της εκπαίδευσης σε αυτή τη συνάρτηση. Αυτό σημαίνει ότι για μια παρτίδα/υποσύνολο δεδομένων (batch μεγέθους 50) πρέπει να αρχικοποιήσετε τις παραγώγους, να υλοποιήσετε τη διάδοση προς τα εμπρός της πληροφορίας (forward propagation), να υπολογίσετε το σφάλμα εκτίμησης, να κάνετε οπισθοδιάδοση της πληροφορίας (μετάδοση προς τα πίσω των παραγώγων σφάλματος ως προς τα βάρη - backward propagation), και τέλος, να ενημερώσετε τις παραμέτρους (weight update). Θα πρέπει να επιλέξετε μια κατάλληλη συνάρτηση απώλειας και βελτιστοποιητή (optimizer) από την βιβλιοθήκη PyTorch για αυτό το πρόβλημα.

Αυτή η συνάρτηση θα χρησιμοποιηθεί στα επόμενα ζητήματα με διαφορετικά δίκτυα. Θα μπορείτε δηλαδή να χρησιμοποιήσετε τη μέθοδο `train_net` για να εκπαιδεύσετε το βαθύ νευρωνικό σας δίκτυο, εφόσον προσδιορίσετε τη συγκεκριμένη αρχιτεκτονική σας και εφαρμόσετε το `forward pass` σε μια υπο/κλάσση της DNN (βλ. παράδειγμα "LinearClassifier(DNN)"). Μπορείτε να ανατρέξετε στη διεύθυνση https://pytorch.org/tutorials/beginner/pytorch_with_examples.html για περισσότερες πληροφορίες. Επίσης, ένα αρκετά χρήσιμο "tutorial" περιλαμβάνεται στο σημειωματάριο jupyter (`tutorial1_pytorch_introduction.ipynb`) στη σελίδα ecourse του μαθήματος.

```
# base class for your deep neural networks. It implements the training
loop (train_net).


import torch.nn.init
import torch.optim as optim
from torch.autograd import Variable
from torch.nn.parameter import Parameter
from tqdm import tqdm
from scipy.stats import truncnorm

class DNN(nn.Module):
    def __init__(self):
        super(DNN, self).__init__()
        pass

    def forward(self, x):
        raise NotImplementedError

    def train_net(self, X_train, y_train, epochs=1, batchSize=50):
        """ ==========
        YOUR CODE HERE
        ========== """


    def __call__(self, x):
        inputs = Variable(torch.FloatTensor(x))
        prediction = self.forward(inputs)
        return np.argmax(prediction.data.cpu().numpy(), 1)

# helper function to get weight variable
def weight_variable(shape):
    initial = torch.Tensor(truncnorm.rvs(-1/0.01, 1/0.01, scale=0.01,
size=shape))
    return Parameter(initial, requires_grad=True)

# helper function to get bias variable
def bias_variable(shape):
```

```
    initial = torch.Tensor(np.ones(shape)*0.1)
    return Parameter(initial, requires_grad=True)

# example linear classifier - input connected to output
# you can take this as an example to learn how to extend DNN class
class LinearClassifier(DNN):
    def __init__(self, in_features=28*28, classes=10):
        super(LinearClassifier, self).__init__()
        # in_features=28*28
        self.weight1 = weight_variable((classes, in_features))
        self.bias1 = bias_variable((classes))

    def forward(self, x):
        # linear operation
        y_pred = torch.addmm(self.bias1, x.view(list(x.size())[0], -
1), self.weight1.t())
        return y_pred

X_train=np.float32(np.expand_dims(X_train,-1))/255
X_train=X_train.transpose((0,3,1,2))

X_test=np.float32(np.expand_dims(X_test,-1))/255
X_test=X_test.transpose((0,3,1,2))

## In case abovementioned 4 lines return error: Modify the lines for
transposing X_train
## and X_test by uncommenting the following 4 lines and place the 4
lines above in comments

#X_train = np.float32(X_train) / 255.0
#X_train = X_train.reshape(-1, 1, 28, 28)

#X_test = np.float32(X_test) / 255.0
#X_test = X_test.reshape(-1, 1, 28, 28)

# test the example linear classifier (note you should get around 90%
accuracy
# for 10 epochs and batchsize 50)
linearClassifier = LinearClassifier()
linearClassifier.train_net(X_train, y_train, epochs=10)

print ('Linear classifier accuracy: %f'%test(X_test, y_test,
linearClassifier))

# display confusion matrix
""" ==========
YOUR CODE HERE
========== """
```

Ζήτημα 2.3: Οπτικοποίηση Βαρών (Visualizing Weights of Single Layer Perceptron) [3 μονάδες]

Αυτός ο απλός γραμμικός ταξινομητής που υλοποιείται στο παραπάνω κελί (το μοντέλο απλά επιστρέφει ένα γραμμικό συνδυασμό της εισόδου) παρουσιάζει ήδη αρκετά καλά αποτελέσματα.

- Σχεδιάστε τα βάρη του φίλτρου που αντιστοιχούν σε κάθε κατηγορία εξόδου (τα **βάρη**/weights, όχι τους όρους *bias*) ως εικόνες. Κανονικοποιήστε τα βάρη ώστε να βρίσκονται μεταξύ 0 και 1 ($z_i=(w_i-min(w))/(max(w)-min(w))$). Χρησιμοποιήστε έγχρωμους χάρτες όπως "inferno" ή "plasma" για καλά αποτελέσματα (π.χ. cmap='inferno', ως όρισμα της imshow()).
- Σχολιάστε με τι μοιάζουν τα βάρη και γιατί μπορεί να συμβαίνει αυτό.

```
# Plot filter weights corresponding to each class, you may have to
reshape them to make sense out of them
# linearClassifier.weight1.data will give you the first layer weights
""" ==========
YOUR CODE HERE
========== """
```

Σχολιασμός των βαρών

Ζήτημα 2.4: Νευρωνικό δίκτυο πολλαπλών επιπέδων - Multi Layer Perceptron (MLP) [7 μονάδες]

Θα υλοποιήσετε ένα MLP νευρωνικό δίκτυο. Το MLP θα πρέπει να αποτελείται από 2 επίπεδα (πολλαπλασιασμός βάρους και μετατόπιση μεροληψίας/bias - γραμμικός συνδυασμός εισόδου) που απεικονίζονται (map) στις ακόλουθες διαστάσεις χαρακτηριστικών:

- 28x28 -> hidden (50)
- hidden (50) -> classes
- Το κρυμμένο επίπεδο πρέπει να ακολουθείται από μια μη γραμμική συνάρτηση ενεργοποίησης ReLU. Το τελευταίο επίπεδο δεν θα πρέπει να έχει εφαρμογή μη γραμμικής απεικόνισης καθώς επιθυμούμε την έξοδο ακατέργαστων 'logits' (στη μηχανική μάθηση, τα logits είναι οι τιμές που παράγονται από το τελικό επίπεδο ενός μοντέλου πριν περάσουν από μια συνάρτηση ενεργοποίησης softmax. Αντιπροσωπεύουν τις προβλέψεις του μοντέλου για κάθε κατηγορία χωρίς να μετατρέπονται σε πιθανότητες).
- Η τελική έξοδος του υπολογιστικού γράφου (μοντέλου) θα πρέπει να αποθηκευτεί στο self.y καθώς θα χρησιμοποιηθεί στην εκπαίδευση.

Εμφανίστε τον πίνακα σύγχυσης (confusion matrix - υλοποίηση 1ης άσκησης) και την ακρίβεια (accuracy) μετά την εκπαίδευση. Σημείωση: Θα πρέπει να έχετε ~95% ακρίβεια για 10 εποχές (epochs) και μέγεθος παρτίδας (batch size) 50.

Σχεδιάστε τα βάρη του φίλτρου που αντιστοιχούν στην αντιστοίχιση από τις εισόδους στις πρώτες 10 εξόδους του κρυμμένου επιπέδου (από τις 50 συνολικά). Μοιάζουν τα βάρη αυτά καθόλου με τα βάρη που απεικονίστηκαν στο προηγούμενο ζήτημα; Γιατί ή γιατί όχι?

Αναμένεται ότι το μοντέλο εκπαίδευσης θα διαρκέσει από 1 έως μερικά λεπτά για να τρέξει, ανάλογα με τις δυνατότητες της CPU.

```
class MLPClassifer(DNN):
    def __init__(self, in_features=28*28, classes=10, hidden=50):
        """
        Initialize weight and bias variables
        """
        super(MLPClassifer, self).__init__()
        """ ==========
        YOUR CODE HERE
        ========== """


    def forward(self, x):
        """ ==========
        YOUR CODE HERE
        ========== """



mlpClassifer = MLPClassifer()
mlpClassifer.train_net(X_train, y_train, epochs=10, batchSize=50)

# Plot confusion matrix
M_mlp,acc_mlp = Confusion(X_test, y_test, mlpClassifer)

print ('Confusion matrix - MLP classifier accuracy: %f'%acc_mlp)

# Check also standard accucary of test() for consistency
print ('MLP classifier accuracy: %f'%test(X_test, y_test,
mlpClassifer))

VisualizeConfussion(M_mlp)

# Plot filter weights
""" ==========
YOUR CODE HERE
========== """
```

Ζήτημα 2.5: Συνελικτικό Νευρωνικό Δίκτυο - Convolutional Neural Network (CNN) [8 μονάδες]

Εδώ θα υλοποιήσετε ένα CNN με την ακόλουθη αρχιτεκτονική:

- n=10 (output features or filters)
- ReLU(Conv(kernel_size=5x5, stride=2, output_features=n))
- ReLU(Conv(kernel_size=5x5, stride=2, output_features=n*2))
- ReLU(Linear(hidden units = 64))
- Linear(output_features=classes)

Δηλαδή, 2 συνελικτικά επίπεδα (Conv Layers) όπου απεικονίζουν μη-γραμμικά (ReLU) την είσοδο του προηγούμενου επιπέδου, ακολουθούμενα από 1 πλήρως συνδεδεμένο κρυμμένο επίπεδο (FC hidden layer) με μη γραμμική ενεργοποίηση (ReLU) και μετά το επίπεδο εξόδου (output layer) όπου συνδυάζει γραμμικά τις τιμές του προηγούμενου επιπέδου.

Εμφανίστε τον πίνακα σύγχυσης και την ακρίβεια μετά την εκπαίδευση. Θα πρέπει να έχετε περίπου ~98% ακρίβεια για 10 εποχές και μέγεθος παρτίδας 50.

Σημείωση: Δεν επιτρέπεται να χρησιμοποιείτε τις torch.nn.Conv2d() και torch.nn.Linear(). Η χρήση αυτών θα οδηγήσει σε αφαίρεση μονάδων. Χρησιμοποιήστε τις δηλωμένες συναρτήσεις conv2d(), weight_variable() και bias_variable(). Ωστόσο στην πράξη, όταν προχωρήσετε μετά από αυτό το μάθημα, θα χρησιμοποιήσετε torch.nn.Conv2d() που κάνει τη ζωή πιο εύκολη και αποκρύπτει όλες τις υποφαινόμενες λειτουργίες.

Μην ξεχάσετε να σχολιάσετε τον κώδικά σας όπου χρειάζεται (π.χ. στον τρόπο υπολογισμού των διαστάσεων της εξόδου σε κάθε επίπεδο).

```
def conv2d(x, W, stride, bias=None):
    # x: input
    # W: weights (out, in, kH, kW)
    return F.conv2d(x, W, bias, stride=stride, padding=2)

# Defining a Convolutional Neural Network
class CNNClassifer(DNN):
    def __init__(self, classes=10, n=10):
        super(CNNClassifer, self).__init__()
        """ ==========
        YOUR CODE HERE
        ========== """

    def forward(self, x):
        """ ==========
        YOUR CODE HERE
        ========== """
```

```
        return y

cnnClassifer = CNNClassifer()
cnnClassifer.train_net(X_train, y_train, epochs=10, batchSize=50)

# Plot confusion matrix and print the test accuracy of the classifier
""" ==========
YOUR CODE HERE
 ========== """

print ('Confusion matrix - MLP classifier accuracy: %f'%acc_cnn)

# Check also standard accucary of test() for consistency
print ('MLP classifier accuracy: %f'%test(X_test, y_test,
cnnClassifer))

VisualizeConfussion(M_cnn)
```

- Σημειώστε ότι οι προσεγγίσεις MLP/ConvNet οδηγούν σε λίγο μεγαλύτερη ακρίβεια ταξινόμησης από την προσέγγιση K-NN.
- Στη γενική περίπτωση, οι προσεγγίσεις Νευρωνικών Δικτύων οδηγούν σε σημαντική αύξηση της ακρίβειας, αλλά, σε αυτή την περίπτωση, εφόσον το πρόβλημα δεν είναι ιδιαίτερα δύσκολο, η αύξηση της ακρίβειας δεν είναι και τόσο υψηλή.
- Ωστόσο, αυτό εξακολουθεί να είναι αρκετά σημαντικό, δεδομένου του γεγονότος ότι τα ConvNets που χρησιμοποιήσαμε είναι σχετικά απλά, ενώ η ακρίβεια που επιτυγχάνεται χρησιμοποιώντας το K-NN είναι αποτέλεσμα αναζήτησης σε πάνω από 60.000 εικόνες εκπαίδευσης για κάθε εικόνα ελέγχου.
- Συνιστάται ιδιαίτερα να αναζητήσετε περισσότερα για τα νευρωνικά δίκτυα/PyTorch στη διεύθυνση https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html καθώς και στο σχετικό tutorial στη σελίδα ecourse του μαθήματος **tutorial1_pytorch_introduction.ipynb**.
- Τέλος, μπορείτε ακόμη να πειραματιστείτε (από δικό σας ενδιαφέρον) με ένα demo νευρωνικού δικτύου που δημιουργήθηκε από τους Daniel Smilkov και Shan Carter στη διεύθυνση https://playground.tensorflow.org/ (για πλατφόρμα TensorFlow).

Οδηγίες υποβολής

Μην ξεχάσετε να κάνετε turnin το αρχείο Jupyter notebook **και** το PDF αρχείο αυτού του notebook μαζί με το συνοδευτικό αρχείο `onoma.txt`: **turnin assignment_3@mye046 onoma.txt assignment3.ipynb assignment3.pdf**

Βεβαιωθείτε ότι το περιεχόμενο σε **κάθε κελί εμφανίζεται** καθαρά στο τελικό σας αρχείο PDF. Για να μετατρέψετε το σημειωματάριο σε PDF, μπορείτε να επιλέξετε **έναν** από τους παρακάτω τρόπους:

1. Google Colab (Συνιστάται): You can `print` the web page and save as PDF (e.g. Chrome: Right click the web page → Print... → Choose "Destination: Save as PDF" and click "Save"). Προσοχή στην περίπτωση όπου κώδικας/σχόλια εμφανίζονται εκτός των ορίων της σελίδας. Μια λύση είναι η αλλαγή γραμμής π.χ. σε σχόλια που υπερβαίνουν το πλάτος της σελίδας.

- Στην περίπτωση που οι εικόνες εξόδου δεν εμφανίζονται σωστά, μια λύση μέσω colab είναι (εργαλείο nbconvert):
  - Ανέβασμα του αρχείου `assignment3.ipynb` στο home directory του Colaboratory (ο κατάλογος home είναι: /content/).
  - Εκτελέστε σε ένα κελί colab ενός νέου notebook: `!jupyter nbconvert --to html /content/assignment3.ipynb`
  - Κάνετε λήψη του assignment3.html τοπικά στον υπολογιστή σας και ανοίξτε το αρχείο μέσω browser ώστε να το εξάγετε ως PDF.

1. Local Jupyter/JupyterLab(Συνιστάται): You can `print` the web page and save as PDF (File → Print... → Choose "Destination: Save as PDF" and click "Save"). Προσοχή στην περίπτωση όπου κώδικας/σχόλια εμφανίζονται εκτός των ορίων της σελίδας. Μια λύση είναι η αλλαγή γραμμής π.χ. σε σχόλια που υπερβαίνουν το πλάτος της σελίδας.

2. Local Jupyter/JupyterLab(Συνιστάται!): You can `export` and save as HTML (File → Save & Export Notebook as... → HTML). Στη συνέχεια μπορείτε να μετατρέψεται το HTML αρχείο αποθηκεύοντάς το ως PDF μέσω ενός browser.