Lab1_Christine_Schmiedler_Vachana_Javali

November 12, 2025

Fundamentals of Machine Learning and Word Embeddings

Christine Schmiedler (Chrsch-5@student.ltu.se)

Vachana Javali (vacjav-2@student.ltu.se)

Part 1

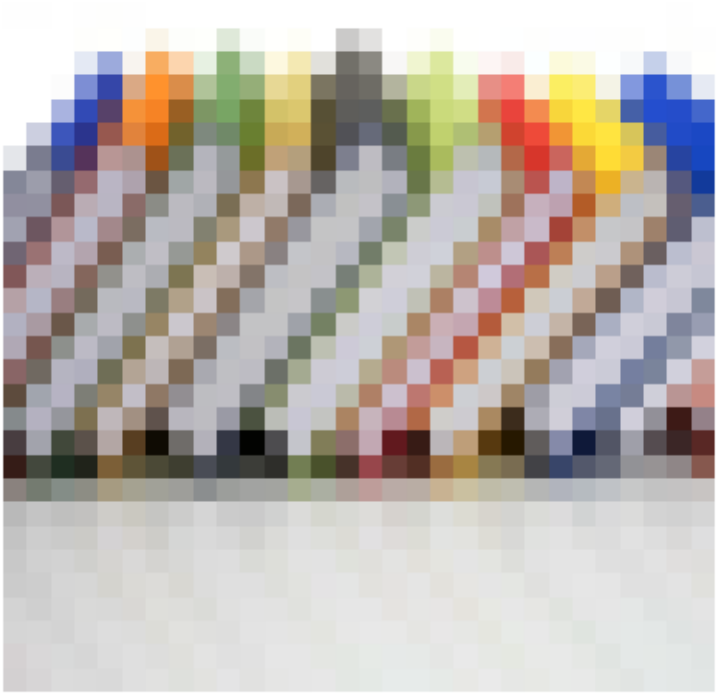
Task 1.1

1.1.1

[40]:
```
import numpy as np
from matplotlib import pyplot as plt
import PIL
from os import listdir
from os.path import isfile, join


imagefilenames = [f for f in listdir("./our_dataset") if isfile(join("./
 ↪our_dataset", f))]
images = []

for imagename in imagefilenames:
    with PIL.Image.open("./our_dataset/" + imagename) as im:
        im = im.resize((30,30))
        image_array = np.array(im)
        images.append(image_array)
        x, y = image_array.shape[:2]

images = np.array(images)
```

1.1.2

[41]:
```
def PlotSample(index: int, images):
    plt.imshow(images[index])
    plt.axis('off')
    plt.show()

PlotSample(21, images)
```



1.1.3

[42]:
```
images_flat = images.reshape(images.shape[0], images.shape[1]*images.
↪shape[2]*images.shape[3])


def plotImage(X):
    plt.figure(figsize=(1.5, 1.5))
    plt.imshow(X.reshape(30,30,3))
    plt.axis('off')
    plt.show()
    plt.close()


images_norm = images_flat/ 255

images_norm = images_norm - images_norm.mean(axis=0)

cov = np.cov(images_norm, rowvar=False)

U,S,V = np.linalg.svd(cov)

epsilon = 0.1
```

```
images_ZCA = U.dot(np.diag(1.0/np.sqrt(S + epsilon))).dot(U.T).dot(images_norm.
 ↪T).T

images_ZCA_rescaled = (images_ZCA - images_ZCA.min()) / (images_ZCA.max() -␣
 ↪images_ZCA.min())

plotImage(images_flat[12, :])
plotImage(images_ZCA_rescaled[12, :])
```

Task 1.2

1.2.1

[43]:
```
# Vachana Code here
```

1.2.2.

[44]:
```
# Vachana Code here
```

1.2.3

[45]:
```
# Vachana Code here
```

Part 2

Provided Code

[46]:
```
import numpy as np
from matplotlib import pyplot as plt
from keras.datasets import mnist


(Xtr, Ltr), (X_test, L_test)=mnist.load_data()

#Traing phase
num_sample=500
Tr_set=Xtr[:num_sample,:,:]
Ltr_set=Ltr[:num_sample]

Tr_set=Tr_set.reshape(num_sample,Tr_set.shape[1]*Tr_set.shape[2])

#Tr_set=Tr_set.reshape(num_sample,Tr_set.shape[1]*Tr_set.shape[2]).astype()
Tr_set.shape

def predict(X):
    num_test=X.shape[0]
    Lpred=np.zeros(num_test, dtype=Ltr_set.dtype)

    for i in range(num_test):
        distances=np.sum(np.abs(Tr_set-X[i,:]),axis=1)

        min_index= np.argmin(distances)
        Lpred[i]=Ltr_set[min_index]
    return Lpred

Test_images=X_test.reshape(X_test.shape[0],X_test.shape[1]* X_test.shape[2])
Labels_predicted=predict(Test_images)

print("Accuracy:", np.mean(Labels_predicted==L_test))
```

```
Accuracy: 0.2649
```

Task 2.1

2.1.1 The accuracy is **0.2649**

2.1.2 For L2 the accuracy is **0.19**

[47]:
```
def predict_L2(X, Tr_set, Ltr_set):
    num_test=X.shape[0]
    Lpred=np.zeros(num_test, dtype=Ltr_set.dtype)
```

```
    for i in range(num_test):
        distances=np.sqrt(np.sum(np.square(Tr_set-X[i,:]),axis=1))

        min_index= np.argmin(distances)
        Lpred[i]=Ltr_set[min_index]
    return Lpred

Test_images=X_test.reshape(X_test.shape[0],X_test.shape[1]* X_test.shape[2])
Labels_predicted=predict_L2(Test_images, Tr_set, Ltr_set)

print("Accuracy L2:", np.mean(Labels_predicted==L_test))
```

```
Accuracy L2: 0.19
```

2.2.3 The problem is that the distance array is of type uint8, which leads to overflowing values and nonsense distances. The fix is to transform it to a float32

With this the accuracy is **0.811** for L1 and **0.8294** for L2

[48]:
```
def predict_L1(X, Tr_set, Ltr_set):
    num_test=X.shape[0]
    Lpred=np.zeros(num_test, dtype=Ltr_set.dtype)

    X = X.astype(np.float32)
    Tr = Tr_set.astype(np.float32)

    for i in range(num_test):
        distances= np.sum(np.abs(Tr-X[i]),axis=1)

        min_index= np.argmin(distances)
        Lpred[i]=Ltr_set[min_index]
    return Lpred

def predict_L2(X, Tr_set, Ltr_set):

    num_test=X.shape[0]
    Lpred=np.zeros(num_test, dtype=Ltr_set.dtype)

    X = X.astype(np.float32)
    Tr = Tr_set.astype(np.float32)

    for i in range(num_test):
        distances=np.sqrt(np.sum(np.square(Tr-X[i]), axis=1))
        min_index= np.argmin(distances)
        Lpred[i]=Ltr_set[min_index]

    return Lpred
```

```
Test_images=X_test.reshape(X_test.shape[0],X_test.shape[1]* X_test.shape[2])
Labels_predicted_L1=predict_L1(Test_images, Tr_set, Ltr_set)
Labels_predicted_L2=predict_L2(Test_images, Tr_set, Ltr_set)

print("Accuracy L1:", np.mean(Labels_predicted_L1==L_test))
print("Accuracy L2:", np.mean(Labels_predicted_L2==L_test))
```

```
Accuracy L1: 0.811
Accuracy L2: 0.8294
```

2.1.4

[49]:
```
def majority_vote(array):
    # returns the element found most often in the array.
    count = np.zeros(array.size)
    for i in range (0, array.size):
        count[i] = np.count_nonzero(array == array[i])
    return array[np.argmax(count)]


def predict_L1(X, k, Tr_set, Ltr_set):
    num_test=X.shape[0]
    Lpred=np.zeros(num_test, dtype=Ltr_set.dtype)

    X = X.astype(np.float32)
    Tr = Tr_set.astype(np.float32)

    for i in range(num_test):
        distances= np.sum(np.abs(Tr-X[i]),axis=1)

        min_indexes = np.argpartition(distances, k)[:k]
        Lpred[i]= majority_vote(Ltr_set[min_indexes])
    return Lpred

def predict_L2(X, k, Tr_set, Ltr_set):

    num_test=X.shape[0]
    Lpred=np.zeros(num_test, dtype=Ltr_set.dtype)

    X = X.astype(np.float32)
    Tr = Tr_set.astype(np.float32)

    for i in range(num_test):
        distances=np.sqrt(np.sum(np.square(Tr-X[i]), axis=1))
        min_indexes = np.argpartition(distances, k)[:k]
        Lpred[i]= majority_vote(Ltr_set[min_indexes])

    return Lpred
```

Task 2.2

2.2.1 The best value of k is **3**.

[50]:
```
def Test_k_values(number_folds):
    # Split data into three parts
    fold_size = int(np.floor(Ltr_set.size / number_folds))
    folds = []
    for i in range (0, number_folds-1):
        folds.append((Tr_set[i*fold_size:((i+1)*fold_size):1],␣
 ↪Ltr_set[i*fold_size:((i+1)*fold_size):1]))
    folds.append((Tr_set[(number_folds-1)*fold_size: Ltr_set.size:1],␣
 ↪Ltr_set[(number_folds-1)*fold_size: Ltr_set.size:1]))

    #get accuracy for each k
    average_accuracy = np.zeros(20)
    for k in range (1, 21):
        accuracy_folds = np.zeros(number_folds)


    #run training/tests for all the folds and calulate accuracy for each
        for i in range(0,number_folds):
            training_data = np.empty((0, folds[0][0].shape[1]))
            training_data_labels = np.empty((0,))


            for j in range(0 ,number_folds):
                if (j == i):
                    test_data = folds[i][0]
                    test_data_labels = folds [i][1]
                else:
                    training_data = np.append(training_data, folds[j][0],␣
 ↪axis=0)
                    training_data_labels = np.append(training_data_labels,␣
 ↪folds[j][1], axis = 0)


            Labels_predicted_L2 = predict_L2(test_data, k, training_data,␣
 ↪training_data_labels)
            accuracy_folds[i] = np.mean(Labels_predicted_L2 == test_data_labels)


        average_accuracy[k-1] = np.average(accuracy_folds)
    return np.argmax(average_accuracy)+1



print("The best value of k is", Test_k_values(3))
```

```
The best value of k is 3
```

2.2.2 The tested accuracy for k = 3 and L2 is **0.8189**

[51]:
```
Labels_predicted_L2=predict_L2(Test_images, 3, Tr_set, Ltr_set)
print("Accuracy for k=3 and L2:", np.mean(Labels_predicted_L2==L_test))
```

```
Accuracy for k=3 and L2: 0.8189
```

[ ]:
```

```