МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Звіт

до лабораторної роботи №5

з предмету Комп'ютерне бачення та аналіз зображень

Роботу виконала:

Мерцало Ірина Ігорівна,

студентка групи ПМІМ-11

Львів – 2022

Використовуючи бібліотеку scikit-learn, логістична регресія:

```
In [7]: from sklearn import datasets, metrics
        from sklearn.linear_model import LogisticRegression
        mnist = datasets.load_digits()
        images = mnist.images
        data_size = len(images)
        #Preprocessing images
        images = images.reshape(len(images), -1)
        labels = mnist.target
        #Initialize Logistic Regression
        LR_classifier = LogisticRegression(C=0.01, penalty='l2', tol=0.01)
        #Training the data on only 75% of the dataset. Rest of the 25% will be used in testing the Logistic Regression
        LR_classifier.fit(images[:int((data_size / 4) * 3)], labels[:int((data_size
        / 4) * 3)])
        #Testing the data
        predictions = LR_classifier.predict(images[int((data_size / 4)):])
        target = labels[int((data_size/4)):]
        #Print the performance report of the Logistic Regression model that we learnt
        print("Performance Report: \n %s \n" %
        (metrics.classification_report(target, predictions)))
```

```
Performance Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       131
           1       0.97      0.96      0.96       137
           2       1.00      1.00      1.00       131
           3       0.98      0.92      0.95       136
           4       0.99      0.97      0.98       139
           5       0.96      0.99      0.98       136
           6       0.99      0.99      0.99       138
           7       0.97      0.99      0.98       134
           8       0.95      0.97      0.96       130
           9       0.94      0.98      0.96       136

    accuracy                           0.97      1348
   macro avg       0.98      0.97      0.97      1348
weighted avg       0.98      0.97      0.97      1348
```

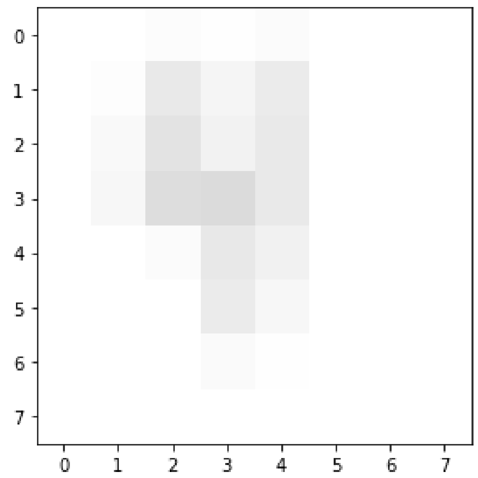
Логістична регресія (розпізнавання знаку):

-вихідне зображення:

digit.png

-результат:

```
In [19]: from sklearn import datasets, metrics
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from skimage import io, color, feature, transform
mnist = datasets.load_digits()
img_tuple=list(zip(mnist.images, mnist.target))
images = mnist.images
data_size = len(images)
#Preprocessing images
images = images.reshape(len(images), -1)
labels = mnist.target
#Initialize Logistic Regression
LR_classifier = LogisticRegression(C=0.01, penalty='l2', tol=0.01)
#Training the data on only 75% of the dataset. Rest of the 25% will be used in testing the Logistic Regression
LR_classifier.fit(images[:int((data_size / 4) * 3)], labels[:int((data_size / 4) * 3)])
#Load a custom image
digit_img = io.imread('digit.png')
#Convert image to grayscale
digit_img = color.rgb2gray(digit_img)
#Resize the image to 28x28
digit_img = transform.resize(digit_img, (8, 8), mode="wrap")
#Run edge detection on the image
digit_edge = feature.canny(digit_img, sigma=5)
io.imshow(digit_img)
io.show()
digit_edge = digit_edge.flatten().reshape(1,-1)
#Testing the data
prediction = LR_classifier.predict(digit_edge)
print(prediction)
```



Опорно-векторні машини:

```
In [22]: from sklearn import datasets, metrics, svm
mnist = datasets.load_digits()
images = mnist.images
data_size = len(images)
#Preprocessing images
images = images.reshape(len(images), -1)
labels = mnist.target
#Initialize Support Vector Machine
SVM_classifier = svm.SVC(gamma=0.001)
#Training the data on only 75% of the dataset. Rest of the 25% will be used in testing the Support Vector Machine
SVM_classifier.fit(images[:int((data_size / 4) * 3)],
labels[:int((data_size / 4) * 3)])
#Testing the data
predictions = SVM_classifier.predict(images[int((data_size / 4)):])
target = labels[int((data_size/4)):]
#Print the performance report of the Support Vector Machine model that we learnt
print("Performance Report: \n %s \n" %
(metrics.classification_report(target, predictions)))
```

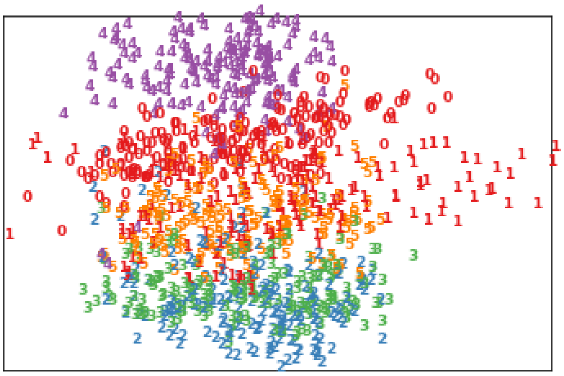
```
Performance Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00       131
           1       0.99      1.00      1.00       137
           2       1.00      1.00      1.00       131
           3       0.99      0.95      0.97       136
           4       0.99      0.98      0.99       139
           5       0.98      0.99      0.99       136
           6       0.99      1.00      1.00       138
           7       0.99      1.00      1.00       134
           8       0.96      0.99      0.98       130
           9       0.99      0.99      0.99       136

    accuracy                           0.99      1348
   macro avg       0.99      0.99      0.99      1348
weighted avg       0.99      0.99      0.99      1348
```

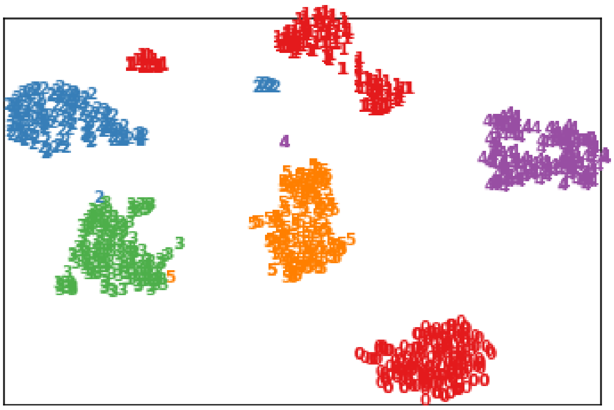
Метод t-SNE:

In [37]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, decomposition, manifold
digits = datasets.load_digits(n_class=6)
X = digits.data
y = digits.target
n_samples, n_features = X.shape
n_neighbors = 30

def plot_embedding(X, title=None):
    x_min, x_max = np.min(X,0), np.max(X,0)
    X=(X-x_min)/(x_max-x_min)
    plt.figure()
    ax=plt.subplot(111)
    for i in range(X.shape[0]):
        plt.text(X[i,0], X[i,1], str(digits.target[i]),
                 color=plt.cm.Set1(y[i]/10.),
                 fontdict={'weight':'bold','size':9})
    plt.xticks([]), plt.yticks([])
    if title is not None:
        plt.title(title)
```

```
n_img_per_row = 20
img = np.zeros((10*n_img_per_row, 10*n_img_per_row))
for i in range(n_img_per_row):
    ix=10*i+1
    for j in range(n_img_per_row):
        iy=10*j+1
        img[ix:ix+8, iy:iy+8]=X[i*n_img_per_row + j].reshape((8,8))
plt.imshow(img, cmap=plt.cm.binary)
plt.xticks([])
plt.yticks([])
plt.title('Numbers')

print("Computing PCA projection")
X_pca = decomposition.TruncatedSVD(n_components=2).fit_transform(X)
plot_embedding(X_pca)

print("Computing t-SNE embedding")
tsne = manifold.TSNE(n_components=2, init='pca', random_state=0)
X_tsne = tsne.fit_transform(X)
plot_embedding(X_tsne)
plt.show()
```

```
Computing PCA projection
Computing t-SNE embedding
```





Метод k-середніх:

In [25]:
```python
from sklearn import datasets, metrics
from sklearn.cluster import KMeans
mnist = datasets.load_digits()
images = mnist.images
data_size = len(images)
#Preprocessing images
images = images.reshape(len(images), -1)
labels = mnist.target
#Initialize Logistic Regression
clustering = KMeans(n_clusters=10, init='k-means++', n_init=10)
#Training the data on only 75% of the dataset. Rest of the 25% will be used in testing the KMeans Clustering
clustering.fit(images[:int((data_size / 4) * 3)])
#Print the centers of the different clusters
print(clustering.labels_)
#Testing the data
predictions = clustering.predict(images[int((data_size / 4)):])
```

```
[5 4 4 ... 7 0 7]
```