Project 1 UMAP-CH

Baoping Wan
b2wan@ucsd.edu

Abstract

The MNIST dataset (Deng, 2012), comprising 60,000 images, poses a computational challenge when applying the 1-NN algorithm due to the noticeable time required. This project aims to address this issue by exploring a method to reduce the dataset while maintaining similar performance. The proposed approach defined a score system to classify the images and select the most efficient subset. This project also explored the prediction on dimension reduced plane. The approach utilized UMAP (McInnes et al., 2020) to project the dataset into a 2D plane and subsequently employs the alpha shape (Bellock, 2021) to selectively retain essential points. This method is coined as UMAP-Alpha.

1 Introduction

A significant source of false predictions in the MNIST dataset is attributed to mislabeled figures, potentially stemming from illegibility or typographical errors. When reducing data size, the impact of mislabeled figures intensifies in the sparser dataset. The major source of figures that can be removed without sacrifice accuracy is the duplicate figures. These figures are almost the same, and as long as one of such similar figures is left in the In this project, the predictions on these mislabeled figures is disregarded. By removing such figures from the training dataset, the remaining monotonous data can be efficiently reduced without sacrificing accuracy.

2 Experiments on the original high dimension dataset

2.1 Algorithm

The score for a given figure of digit i is defined by its 10 neighbors $n_{1,2...10}$ as

$$\sum_{n_j=i}(10-j)-\sum_{n_j\neq i}(10-j)$$

. There are 23 possible scores for this system. The mislabeled figure are defined as the figures with a low score. The duplicate figures are defined as figures with a high score.=
Once all the score are calculated, all possible pairs of upper and lower boundary are tested on the training set, except the mislabeled figure. The efficiency is defined as the ratio between the accuracy by training on the selected subset - 0.94 and the size of the selected subset, where 0.94 is a baseline of accuracy from experiment.

Algorithm 1 Algorithm

1: Calculate score for all figures
2: **for** all possible upper and lower bound of scores **do**
3: Train 1-NN on selected subset
4: Test it on the training set
5: Calculate its efficiency
6: **end for**
7: Select the subset with the highest efficiency
8: Random resample from this subset

2.2 Results

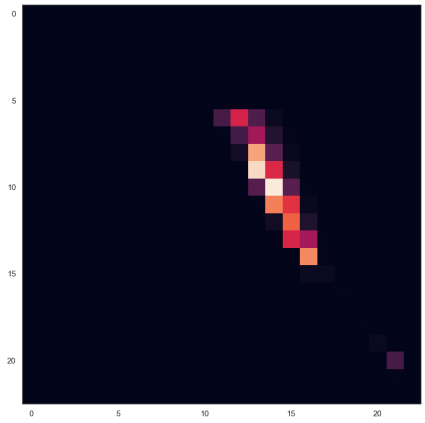


Figure 1: The heat map of testing over low and high boundary pairs. Data are emphasized by exponentiation.

From the testing, the lower bound is the 10th score and the higher bound is the 14th score.

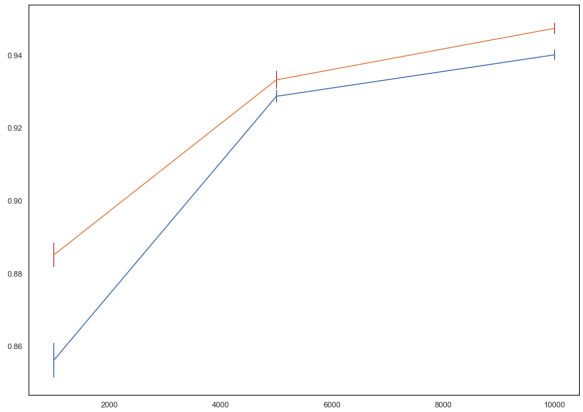


Figure 2: The error bar for testing on the proposed method (blue) and random method (red).

The accuracy for the proposed method is worse than the random method for all sample sizes.

3 Experiments on the dimension reduced dataset

3.1 Preprocess

The UMAP algorithm is chosen for its ability to preserve both global and local structure, low time complexity, and stability across iterations.

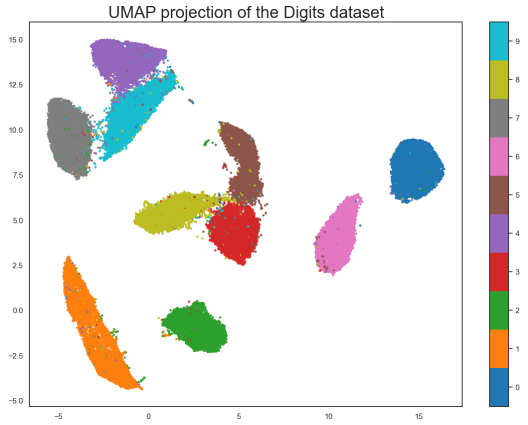


Figure 3: UMAP projection of original dataset.

The clustering of the ten digits into distinct groups is evident in the 2D plane, with only a minimal number of figures overlapping across clusters. These overlapping figures are identified as mislabeled and subsequently eliminated.

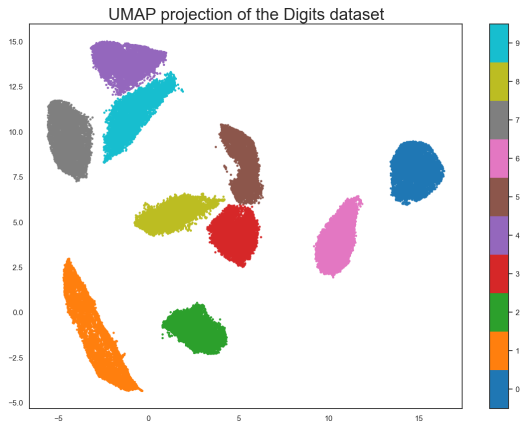


Figure 4: UMAP projection of purified dataset.

3.2 Algorithm

Post removal of mislabeled figures, the remaining data is clustered into ten disjoint clusters. Data points within these clusters can be safely removed without affecting prediction accuracy, with only the boundary points requiring retention. This is achieved by finding the alpha shape using alpha=1 of these clusters.

Algorithm 2 UMAP-Alpha

1: **for** $Cluster = 1, 2, \ldots, 10$ **do**
2: Select points by finding the alpha shape
3: of Cluster.
4: **end for**
5: Combine ten sets of selected points gives the final set of selected points.

3.3 Test Methods

For each alpha of UMAP-Alpha, some data points are selected to be the prototype. For this additional experiment on the dimension reduced dataset, the same size randomly selected data points is tested by 10 times.

3.4 Results

In order to keep the UMAP-Alpha and random resample comparable, both of them are scored by 1-NN on the UMAP reduced 2D plane.
The time cost for UMAP-Alpha is 12.19 second. The sample size of boundary figures is 461. The accuracy achieved by UMAP-Alpha is 0.9558. The mean of random method is 0.93. The standard deviation of random method is 0.0056.
The accuracy of 1-NN for the original whole dataset is 0.9691. The accuracy for 1-NN on the projected dataset is 0.9296. The accuracy for 1-NN on the purified data set is 0.9691.

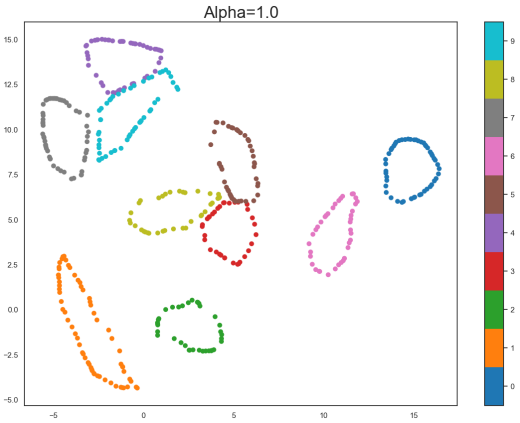


Figure 5: The selected points on the projected 2D plane for alpha 1.0.

4 Discussion

The experiment result show that the proposed method is not getting the desired result. This is because of the uniform weight on these dimensions. Because the the pixel including essential information are equally important as the pixel with noise, the distance between figures can not represent the actual topology relationships between the figures in the dataset.
Dimension reduction do lose some information and decrease the accuracy slightly. On the other hand, dimension reduction also improved the performance of random method on the small sample set, because the number of boundary figures is reduced for the UMAP projection, it's possible to keep all these figures in the resampled dataset.
On the UMAP projection, because the reduce of dimension and weight between dimensions, the boundary figures are reduced to a relatively small amount and is possible to be calculated in a acceptable time. This guarantied the perforce of the proposed method.

5 Conclusion

Removing the outliers and the duplicated figures is a valid method. However, the amount of boundary figures is positive correlated to the dimension. Therefore for a dataset with high dimension, this method can not be a efficient way for creating prototypes.

References

Kenneth E. Bellock. 2021. Alpha shape toolbox.

Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.

Leland McInnes, John Healy, and James Melville. 2020. Umap: Uniform manifold approximation and projection for dimension reduction.

In [36]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_digits
from scipy.spatial import ConvexHull
from sklearn.utils import resample
import matplotlib.pyplot as plt
from os.path  import join
from array import array
import seaborn as sns
import pandas as pd
import numpy as np
import alphashape
import warnings
import random
import struct
import umap
import copy
import time

%matplotlib inline
sns.set(style='white', context='notebook', rc={'figure.figsize':(14,10)})
np.set_printoptions(linewidth=np.inf)
warnings.filterwarnings('ignore')
```

In [2]:
```
class MnistDataloader(object):
    def __init__(self, training_images_filepath,training_labels_filepath,
                 test_images_filepath, test_labels_filepath):
        self.training_images_filepath = training_images_filepath
        self.training_labels_filepath = training_labels_filepath
        self.test_images_filepath = test_images_filepath
        self.test_labels_filepath = test_labels_filepath

    def read_images_labels(self, images_filepath, labels_filepath):
        labels = []
        with open(labels_filepath, 'rb') as file:
            magic, size = struct.unpack(">II", file.read(8))
            if magic != 2049:
                raise ValueError('Magic number mismatch, expected 2049, got {}'.format(
            labels = array("B", file.read())

        with open(images_filepath, 'rb') as file:
            magic, size, rows, cols = struct.unpack(">IIII", file.read(16))
            if magic != 2051:
                raise ValueError('Magic number mismatch, expected 2051, got {}'.format(
            image_data = array("B", file.read())
        images = []
        for i in range(size):
            images.append([0] * rows * cols)
        for i in range(size):
            img = np.array(image_data[i * rows * cols:(i + 1) * rows * cols])
            #img = img.reshape(28, 28)
            images[i][:] = img

        return images, labels

    def load_data(self):
        x_train, y_train = self.read_images_labels(self.training_images_filepath, self.
```

```
        x_test, y_test = self.read_images_labels(self.test_images_filepath, self.test_l
        return (np.array(x_train), np.array(y_train)),(np.array(x_test), np.array(y_te

def show_images(images, title_texts):
    cols = 5
    rows = int(len(images)/cols) + 1
    plt.figure(figsize=(30,20))
    index = 1
    for x in zip(images, title_texts):
        image = x[0]
        title_text = x[1]
        plt.subplot(rows, cols, index)
        plt.imshow(image.reshape(28, 28), cmap=plt.cm.gray)
        if (title_text != ''):
            plt.title(title_text, fontsize = 15);
        index += 1
```

In [38]:
```
input_path = ''
training_images_filepath = join(input_path, 'train-images-idx3-ubyte/train-images-idx3-u
training_labels_filepath = join(input_path, 'train-labels-idx1-ubyte/train-labels-idx1-u
test_images_filepath = join(input_path, 't10k-images-idx3-ubyte/t10k-images-idx3-ubyte')
test_labels_filepath = join(input_path, 't10k-labels-idx1-ubyte/t10k-labels-idx1-ubyte')

mnist_dataloader = MnistDataloader(training_images_filepath, training_labels_filepath,
(x_train, y_train), (x_test, y_test) = mnist_dataloader.load_data()
```

In [16]:
```
nn_classifier=KNeighborsClassifier(n_neighbors=1, algorithm='auto')
nn_classifier.fit(x_train,y_train)
print(nn_classifier.score(x_test,y_test))
```

```
0.9691
```

In [17]:
```
matrix=nn_classifier.kneighbors(x_train, 10, False)
```

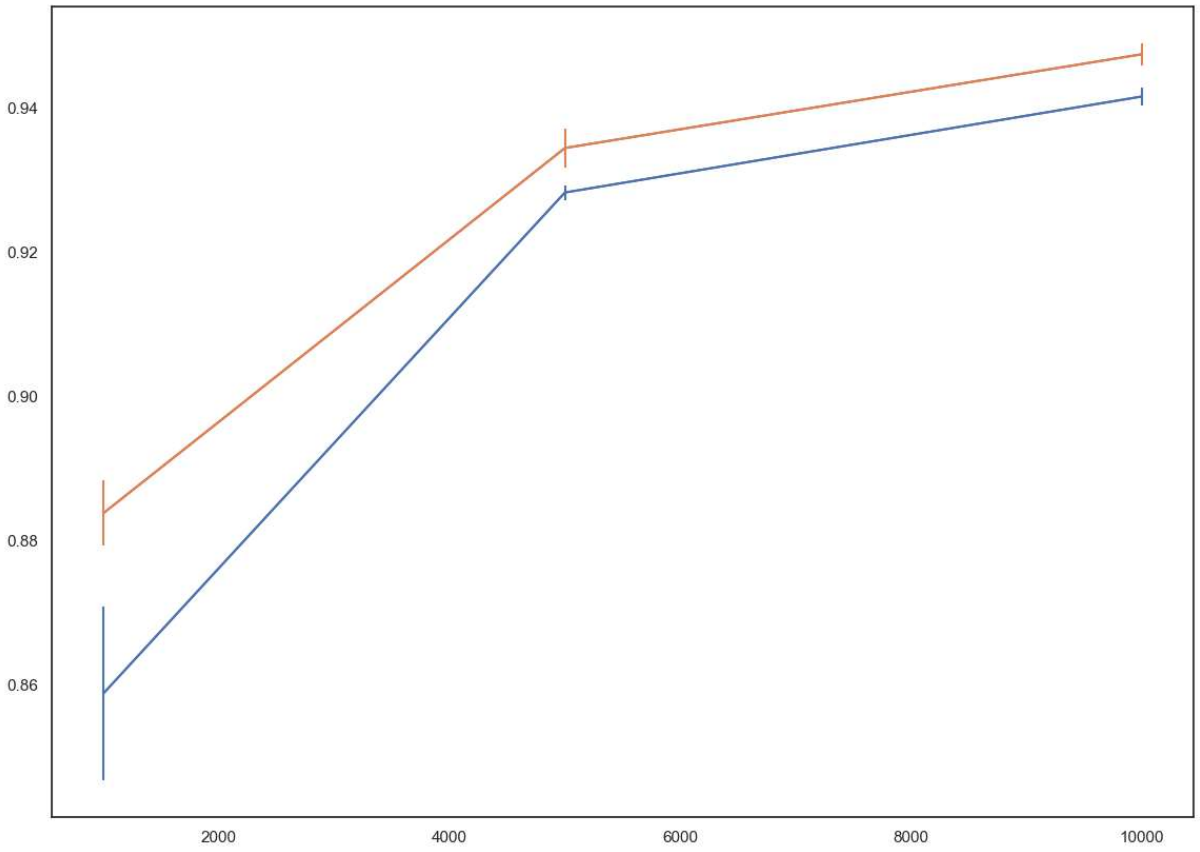
In [269…
```
matrix3=np.zeros((23,23))
matrix4=np.zeros((23,23))
score=np.array([sum([i if o==n else -i for i,o in enumerate(m[::-1])]) for m,n in zip
index=[1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45]
for i in range(22):
    for j in range(22):
        musk=np.unique(matrix[score>=index[i]])
        boundry=np.unique(matrix[np.logical_and(score>=index[i],score<=index[j])])
        matrix4[i,j]=boundry.size
        if boundry.size>0:
            nn_classifier.fit(x_train[boundry,:],y_train[boundry])
            matrix3[i,j]=nn_classifier.score(x_train[musk,:],y_train[musk])
        else:
            matrix3[i,j]=0
```

In [ ]:
```
plt.imshow(np.where(matrix3>0.94,((matrix3-0.94)/matrix4)**40,0))
```

In [ ]:
```
score=np.array([sum([i if o==n else -i for i,o in enumerate(m[::-1])]) for m,n in zip
index=[1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45]
boundry=np.unique(matrix[np.logical_and(score>=index[10],score<=index[14])])
s=[]
for i in [1000,5000,10000]:
    c=[]
    for _ in range(10):
```

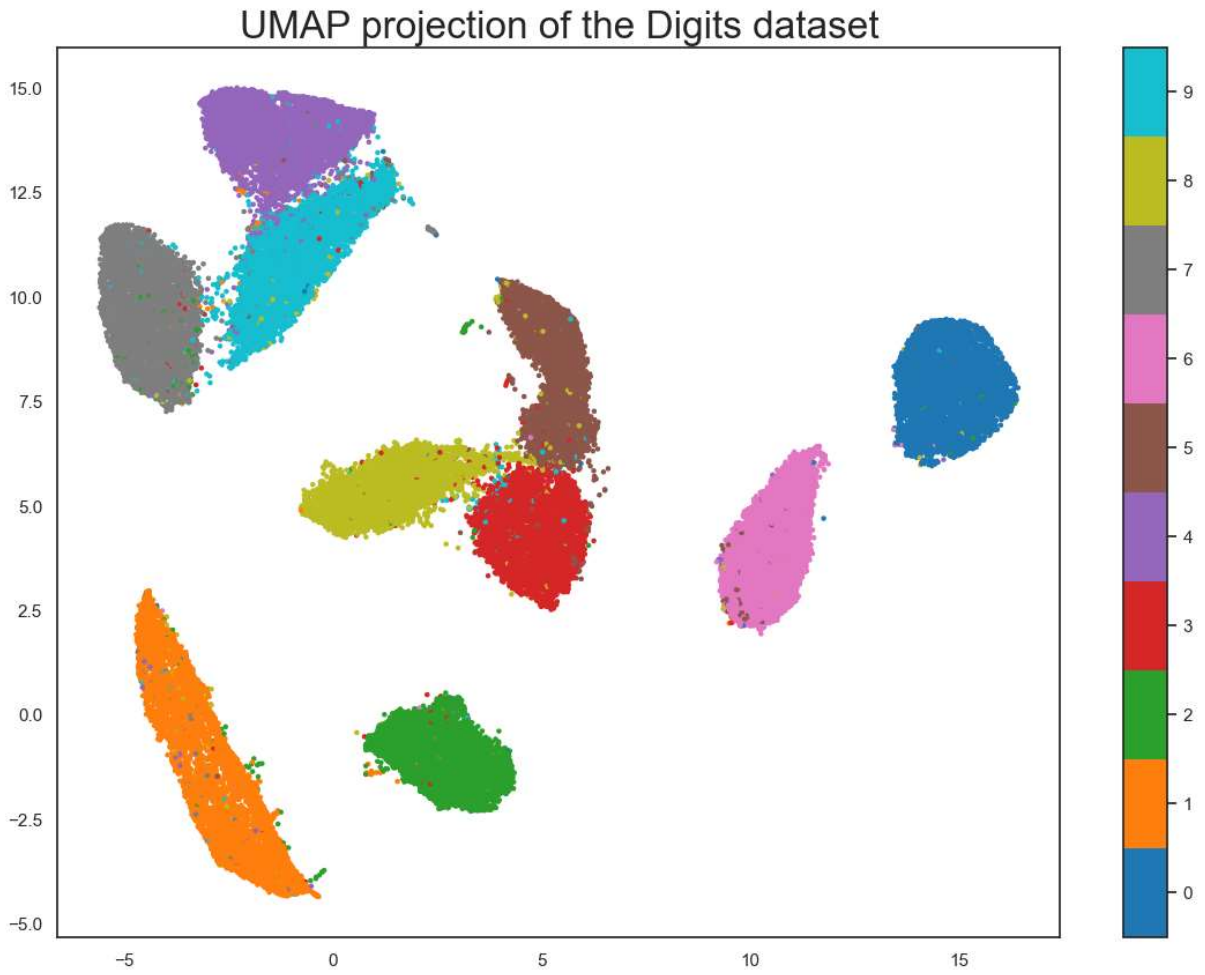
```
        nn_classifier.fit(*resample(x_train[boundry,:],y_train[boundry],n_samples=i))
        c.append(nn_classifier.score(x_test,y_test))
    s.append(c)
s2=[]
for i in [1000,5000,10000]:
    c=[]
    for _ in range(10):
        nn_classifier.fit(*resample(x_train,y_train,n_samples=i))
        c.append(nn_classifier.score(x_test,y_test))
    s2.append(c)
```

In [31]:
```
plt.plot([1000,5000,10000],[np.mean(i) for i in s],c='b')
plt.plot([1000,5000,10000],[np.mean(i) for i in s2],c='r')
plt.errorbar([1000,5000,10000],[np.mean(i) for i in s],yerr=[np.std(i)  for i in s])
plt.errorbar([1000,5000,10000],[np.mean(i) for i in s2],yerr=[np.std(i)  for i in s2])
plt.show()
```



In [ ]:
```
Umap=umap.UMAP(random_state=42).fit(x_train)
embedding = Umap.transform(x_train)
embedding2 = Umap.transform(x_test)
```

In [5]:
```
plt.scatter(embedding[:, 0], embedding[:, 1], c=y_train, cmap='tab10', s=5)
plt.gca().set_aspect('equal', 'datalim')
plt.colorbar(boundaries=np.arange(11)-0.5).set_ticks(np.arange(10))
plt.title('UMAP projection of the Digits dataset', fontsize=24);
```

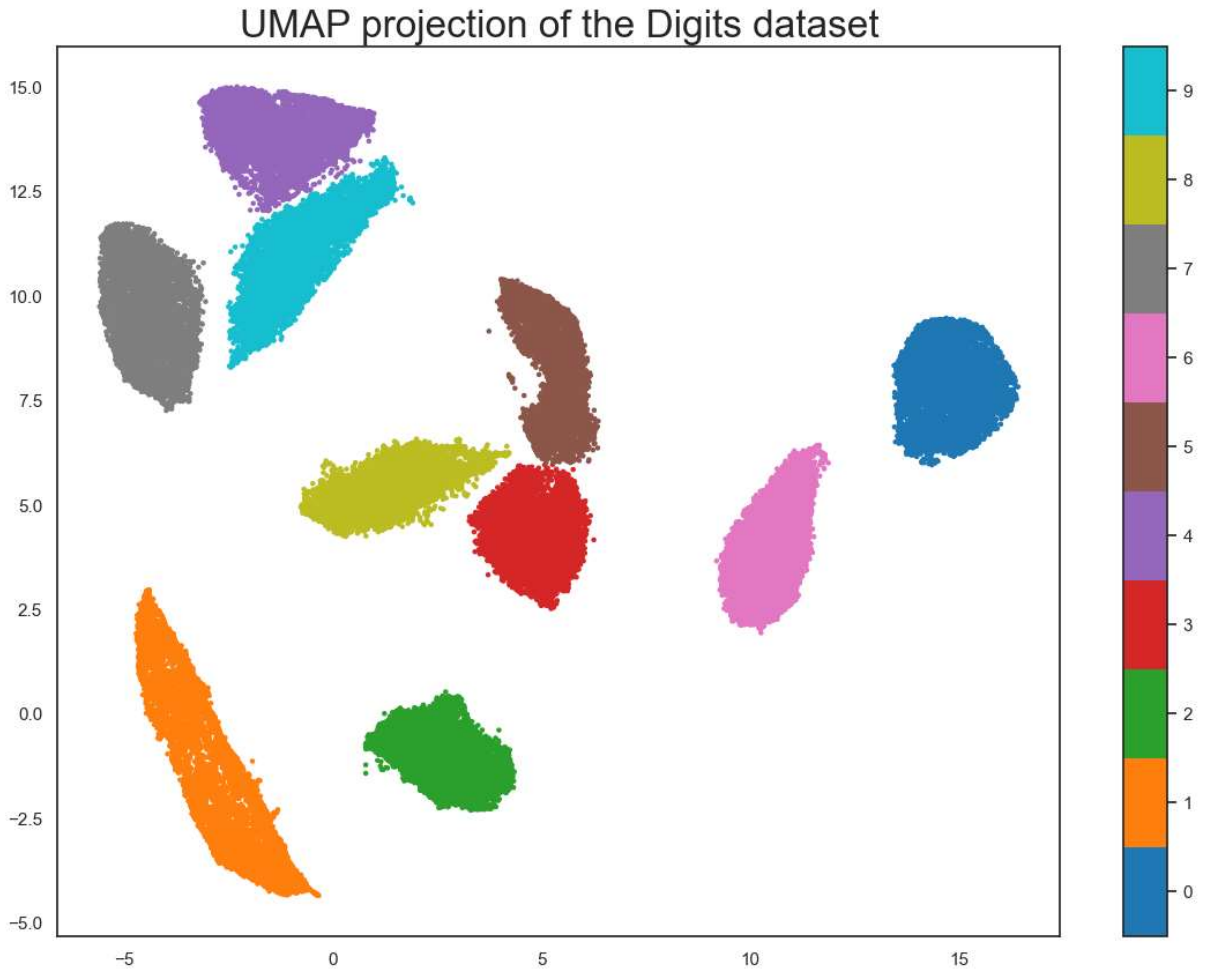


In [6]:
```
l=[[],[],[],[],[],[],[],[],[],[]]
for k,(i,j) in enumerate(zip(embedding,y_train)):
    if i[0]>13 and j==0:
        l[0].append(k)
    elif i[0]<0 and i[1]<3 and j==1:
        l[1].append(k)
    elif i[0]>0 and i[1]<1 and j==2:
        l[2].append(k)
    elif i[1]-1.5<i[0] and i[0]<6.5 and 2<i[1] and i[1]<6 and j==3:
        l[3].append(k)
    elif i[0]<2 and i[1]>i[0]/2+12.8 and i[1]>12 and j==4:
        l[4].append(k)
    elif i[0]>3 and i[0]<6.5 and i[1]>6 and i[1]<12 and 16-2*i[0]<i[1] and j==5:
        l[5].append(k)
    elif i[0]>8 and i[0]<12 and j==6:
        l[6].append(k)
    elif i[0]<-3 and i[1]>7 and i[1]<12 and j==7:
        l[7].append(k)
    elif i[0]>-2 and i[0]<5 and i[1]<7 and i[0]+2<i[1] and 2-i[0]>2*i[1]-15 and j==
        l[8].append(k)
    elif i[0]>-2.5 and i[0]<2 and i[1]>8 and i[1]<14 and i[0]/2+12.7>i[1] and j==9:
        l[9].append(k)
l=[np.array(i) for i in l]
```

In [7]:
```
nn_classifier=KNeighborsClassifier(n_neighbors=1, algorithm='auto')
nn_classifier.fit(x_train[np.concatenate(l),:],y_train[np.concatenate(l)])
print(nn_classifier.score(x_test,y_test))
```

```
xl=np.concatenate(([embedding[l[i],:] for i in range(10)]))
yl=np.concatenate(([y_train[l[i]] for i in range(10)]))
plt.scatter(xl[:, 0], xl[:, 1], c=yl, cmap='tab10', s=5)
plt.gca().set_aspect('equal', 'datalim')
plt.colorbar(boundaries=np.arange(11)-0.5).set_ticks(np.arange(10))
plt.title('UMAP projection of the Digits dataset', fontsize=24);
```

0.9673

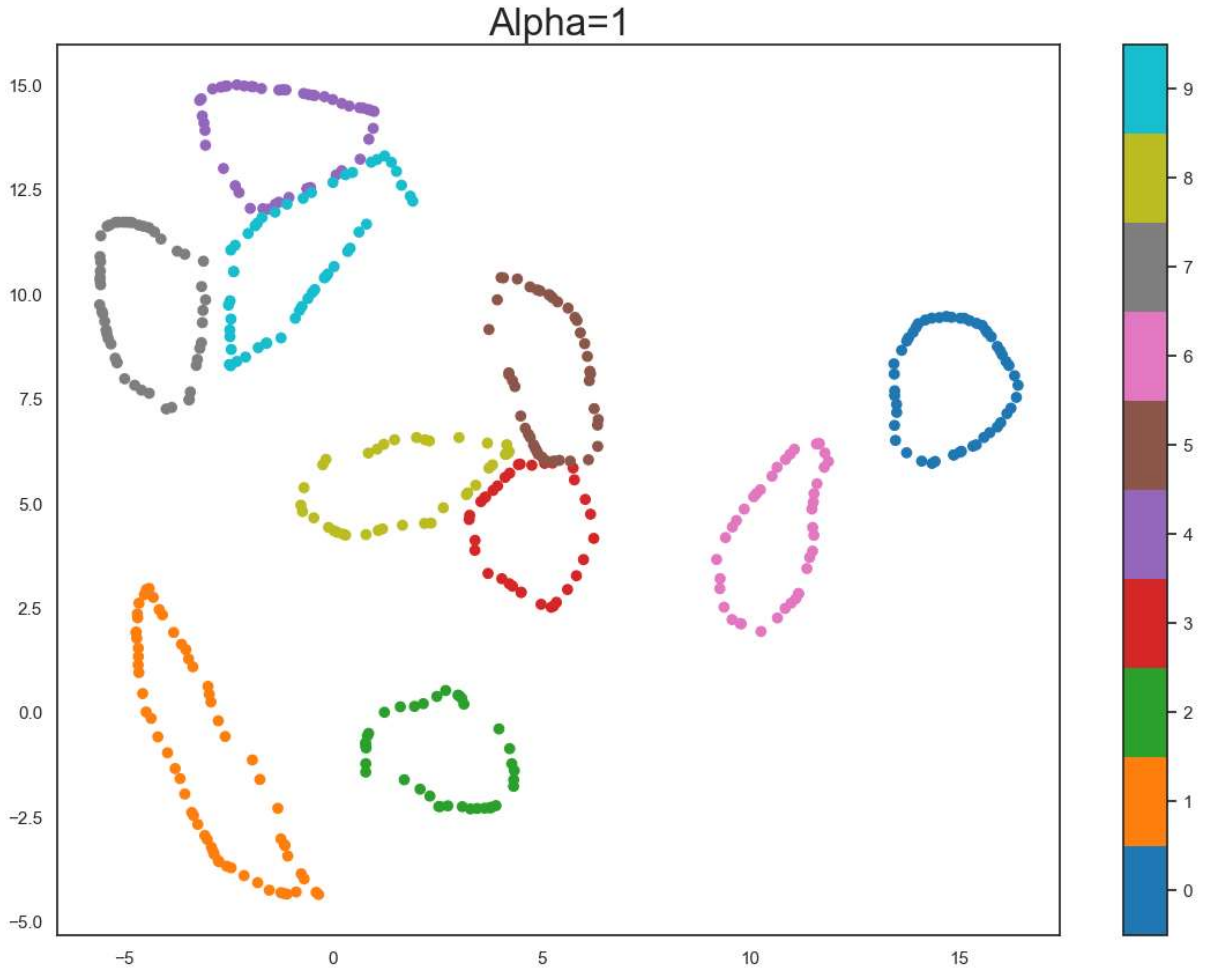


In [9]:

```
t=time.time()
tl = [np.array(alphashape.alphashape(embedding[i,:], 1).exterior.coords.xy).T for i in

yl = np.concatenate(([np.full(tl[i].shape[0],i) for i in range(10)]))
xl = np.concatenate(tl)

nn_classifier2=KNeighborsClassifier(n_neighbors=1, algorithm='auto')
nn_classifier2.fit(xl,yl)
print(nn_classifier2.score(embedding2,y_test))
print(time.time()-t)

'''s=0
for _ in range(10):
    nn_classifier.fit(*resample(embedding,y_train,n_samples=yl.shape[0]))
    s=max(s,nn_classifier.score(embedding2,y_test))
print(s)'''

print(yl.shape[0])
```

```
plt.scatter(xl[:,0], xl[:,1], c=yl, cmap='tab10')
plt.gca().set_aspect('equal', 'datalim')
plt.colorbar(boundaries=np.arange(11)-0.5).set_ticks(np.arange(10))
plt.title('Alpha='+str(1), fontsize=24)
plt.show()
```

```
0.9558
12.194995880126953
461
```



In [35]:
```
s3=[]
for _ in range(10):
    nn_classifier.fit(*resample(embedding,y_train,n_samples=461))
    s3.append(nn_classifier.score(embedding2,y_test))
print(np.mean(s3))
print(np.std(s3))
```

```
0.93093
0.00562513110958314
```

In [ ]: