Final Project ECE 532

Introduction to K nearest neighbors algorithm and its application in hand-written digit recognition with PCA

Yebo Cao, Yuhua Chen, Zhaoning Wang

December 11, 2019

Abstract:

In pattern recognition, the k-nearest neighbors algorithm (KNN) is a method used for classification. KNN is a supervised learning method and does not require a lengthy training phrase, as we will see later[1]. KNN shares the same kind of intuition as clustering as they both rely on the distance of data points in the feature space. Similar to clustering, KNN does not require extensive mathematical background and the implementation is relatively straightforward compared to some other more complicated methods studied in class, such as SVD. The sample implementation included in this activity can be easily modified into C code and be used with a real camera.

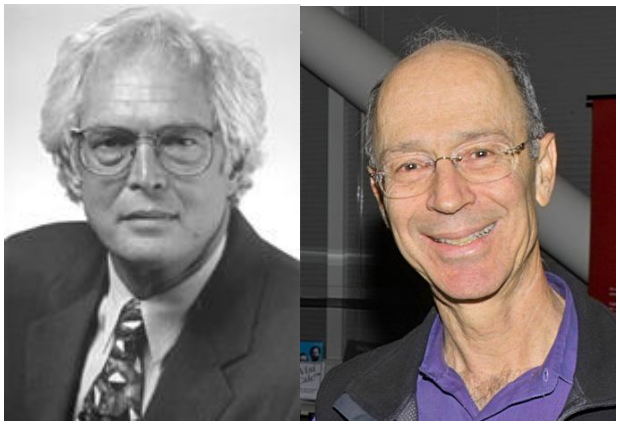
Dimension is a significant concept in machine learning. The sparseness of the training sample greatly reduces its ability to represent the overall distribution, thereby reducing the generalization ability of the learner; at the same time, When the dimension is high, calculating the distance becomes very complicated, and even calculating the inner product is no longer easy. KNN usually brings an astronomical number of dimensions, and thus Principal Component Analysis (PCA) is needed to make these data samples feasible by discarding trivial dimensions.

Learning objectives:

- Understand what is K nearest neighbors algorithm and its applications
- Learn about how to select features, the value of k and distance types in such clustering classification problems
- Learn how to use PCA to reduce features and reduce the computational complexity.

- Learn how to visualize the data to gain intuition about the problem.
- Apply KNN to the problem of hand-written digit recognition

Optional Historical Note



The k-nearest neighbors (KNN) algorithm was proposed by Cover and Hart in 1968. [2], [3]

Before Cover and Hart, the rule was mentioned by Nilsson (1965) who called it “minimum distance classifier” and by Sebestyen (1962), who called it “proximity algorithm”. Fix and Hodges in their very early discussion on non-parametric discrimination (1951) already pointed to the NN rule as well[4]. The fundamental principle is known as Ockham’s razor: “select the hypothesis with the fewest assumptions” can be understood as the NN rule for nominal properties. It is, however, not formulated in terms of observations. Ockham worked in the 14th century and emphasized observations above ideas.

Background:

The idea of the algorithm: 1) Calculate the distance or similarity between the sample to be classified and the training sample of the known category; 2) Find the distance or similarity with The k nearest neighbors of the sample data to be classified; 3) Determine the category of the sample data to be classified according to the category to which the k neighbors belong. If most of the k neighbors of the sample data to be classified belong to a certain category, the test samples to be classified also belong to this category[5].

Hence, finding an appropriate method to calculate distance is significant for building a good KNN algorithm. Generally, there are three main methods been used widely.

1) Euclidean Distance

The most common representation of the distance between two or more points, also known as the Euclidean metric, is defined in Euclidean space, such as points x = (x1, ..., xn) and The distance between y = (y1, ..., yn) is:

$$d(x,y) := \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

2) Manhattan Distance

It is used to indicate the sum of the absolute wheelbases of two points on the standard coordinate system.

Manhattan distance between n-dimensional space points a (x11, x12, ..., x1n) and b (x21, x22, ..., x2n):

$$d_{12} = \sum_{k=1}^{n} |x_{1k} - x_{2k}|$$

3) Chebyshev Distance

Mathematically, the Chebyshev distance or L∞ metric is a metric in vector space. The

distance between two points is defined as the maximum value of the difference between their coordinates.

Chebyshev distance between n-dimensional space points a (x11, x12, ..., x1n) and b

(x21, x22, ..., x2n):

$$d_{12} = \max_i(|x_{1i} - x_{2i}|)$$

Next, the choice of K value is also important.

If you choose a smaller value of K, it is equivalent to using the training examples in the smaller field to make predictions. The "learning" approximation error will be reduced. Only training examples that are close to or similar to the input instance will affect the prediction result. At the same time, the problem is that the estimation error of "learning" will increase. In other words, a decrease in the value of K means that the overall model becomes complicated and prone to overfitting;

If a larger value of K is selected, it is equivalent to using a training example in a larger field for prediction. The advantage is that the estimation error of learning can be reduced, but the disadvantage is that the approximate error of learning will increase. At this time, training examples that are far away (not similar) from the input instance will also act on the predictor, causing prediction errors, and an increase in the value of K means that the overall model becomes simple.

 Yann LeCun proposed LeNet 5 in 1998

The problem we are going to investigate is also interesting and classic: the problem of handwritten character recognition. The task is to identify handwritten or printed text in picture and convert them to digital ASCII text. The ability to convert picture to text is important for the management of a large amount of information, as text stored in computer is searchable. The famous LeNet5 was originally developed as a new approach to this problem. Also, the dataset we use is part of the MNIST dataset, which is the dataset Yann LeCun used more than 20 years ago. When Yann LeCun studied the problem of character recognition, he included KNN as a benchmark reference. However, even though KNN is outperformed by LeNet 5 with a huge margin in the more complicated task of text recognition, we are only using KNN for single digit recognition, where KNN still has satisfying performance.[6]

KNN algorithm advantages

The KNN algorithm is simple and easy to use. Compared with other algorithms, KNN is relatively simple and clear. It is possible to understand its principle even without a high mathematical foundation. In addition, the model training time is fast. As mentioned above, the KNN algorithm is lazy, so it will not be described too much here. Moreover, the prediction effect is decent and it is not sensitive to outliers [7].

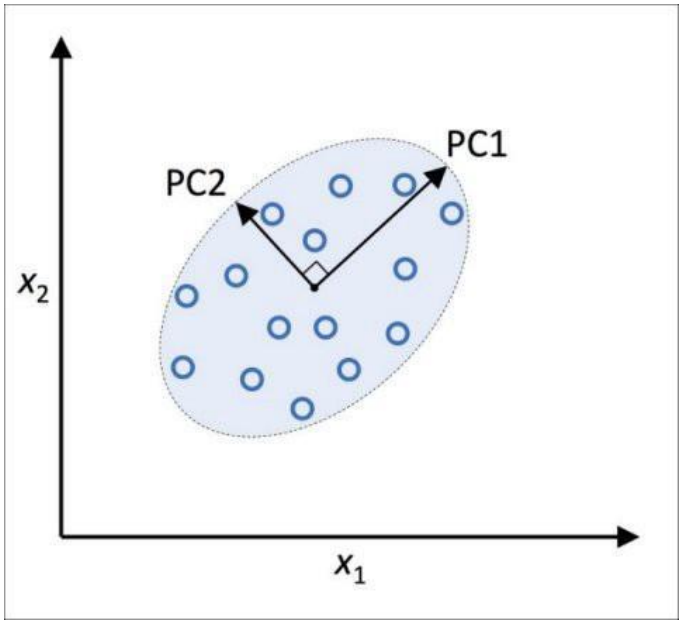
KNN algorithm disadvantages

It requires high memory because the KNN algorithm stores all training data. The prediction phase can be slow. Moreover, It is sensitive to unrelated features and data size[7].

In the real world, although KNN is widely used in image and digit processing, it can be significantly slow and memory-consuming because of the high dimension of the image. (Think about a 960 x 640 image with RGB value of 255.) This is where PCA kicks in and save the day! PCA can significantly extract the feature and reduce the dimension and thus computational complexity. In our activity, it decreases the dimension by 100x!

PCA review and dimension reduction

PCA (Principal Component Analysis) is one of the most widely used linear transformation techniques. PCA aims to find the directions of maximum variance in high-dimensional data and projects it onto a new subspace with equal or fewer dimensions than the original one.



In the preceding figure, *x1* and *x2* are the original feature axes, and **PC1** and **PC2** are the principal components[8].

One of the main goals of using PCA is to reduce dimensions, such as reducing N-dimensional data to K-dimensions.Let's start with two very simple examples of dimension reduction:

For a set of 2D data recall from the class material that the principal components are obtained from the maximum variance among all the data points, for the first component:

$$\mathbf{w}_{(1)} = \arg\max \left\{ \frac{\mathbf{w}^T \mathbf{X}^{\mathrm{T}} \mathbf{X} \mathbf{w}}{\mathbf{w}^T \mathbf{w}} \right\}$$

and further components:

$$\hat{\mathbf{X}}_k = \mathbf{X} - \sum_{s=1}^{k-1} \mathbf{X} \mathbf{w}_{(s)} \mathbf{w}_{(s)}^{\mathrm{T}}$$

Those principal components also indicate features. Thus, we can map the data point to first N principal components to complete the dimension reduction[8].

For example:

2D to 1D[9]

As the principal component gets smaller which indicates meaningless when the number goes up, we will be able to omit many of them and significantly reduce the dimension and degree of computational complexity. You will get some real-world experience in the last part of the Activity.

Potential ethical issues

Based on the discussion above, even with perfectly implemented the KNN algorithm and PCA, it is still quite possible to generate a wrong prediction. Therefore, the involvement of people judges is needed when it is necessary. Moreover, the KNN and PCA completely rely on the data, and thus the data collected should be preprocessed before implementing these algorithms to get rid of some uncontrollable parameters such as race and family condition.

References:

[1]J. M. Keller, M. R. Gray, and J. A. Givens, “A fuzzy K-nearest neighbor algorithm,” *IEEE Trans. Syst. Man Cybern.*, vol. SMC-15, no. 4, pp. 580 – 585, Jul. 1985.

[2]“Peter E. Hart,” *Wikipedia*. 22-Apr-2018.

[3]“Thomas M. Cover,” *Wikipedia*. 05-Dec-2019.

[4]“Who invented the nearest neighbor rule?,” *Pattern Recognition Tools*, 28-Jan-2014. [Online]. Available: http://37steps.com/4370/nn-rule-invention/. [Accessed: 11-Dec-2019].

[5]J. Brownlee, “K-Nearest Neighbors for Machine Learning,” *Machine Learning Mastery*, 14-Apr-2016. .

[6]Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278 – 2324, Nov. 1998.

[7]N. S. Chauhan, “Implement K-Nearest Neighbors classification Algorithm,” *Medium*, 09-Jul-2019. [Online]. Available: https://towardsdatascience.com/implement-k-nearest-neighbors-classification-algorithm-c99be8f14052. [Accessed: 11-Dec-2019].

[8]L. Li, “Principal Component Analysis for Dimensionality Reduction,” *Medium*, 27-May-2019. [Online]. Available: https://towardsdatascience.com/principal-component-analysis-for-dimensionality-reduction-115a3d157bad. [Accessed: 11-Dec-2019].

[9]J. VanderPlas, *Python Data Science Handbook: Essential Tools for Working with Data*. Sebastopol, UNITED STATES: O'Reilly Media, Incorporated, 2016.

Activity:

K-Nearest-Neighbors basics:

1. KNN basics

   a. What are the popular distances metric used in KNN?

   b. Does KNN has a clear decision boundary? In general, what are the effects of a k that is too large or too small for KNN?

   c. Download and open the KNN_tutorial.ipynb. Run the file and observe the result. Now read about the four functions in this tutorial and understand their functionality. Then change the distance calculation from Manhattan distance to Euclidean distance. Run the notebook again and comment on the change in performance. (Only change one line of code)

2. Now try to run the knn_digits python script to train and test different handwritten digits. The training data consists of 1790 samples with 179 samples for each number. **Do not run drawing and PCA for now.** Each time the code would randomly pick 1/10 of the cases for tests and others for training. What is the precision when we take 200 neighbors for prediction? Does it looks accurate and why? find a neighbor number/range that prediction rate is good enough (error rate < 7.5%).

PCA for dimension reduction:

3. Suppose you have 4 points in a 2-D dimension:

$$[1,0],[0,1],[4,5],[5,4]$$

Try to find α for the first two principal components and corresponding variance without using python or Matlab script.

4. Retrain the data and run PCA partition after that. Observe the dimension change before and after the PCA and the plot of the first two components.

   a. Are the numbers in clusters in PCA plot?

b. According to the PCA plot, which numbers can easily confuse the KNN if we use just two dimensions and Why? How do you identify the numbers that are easier to predict?
c. Run the KNN again. How is precision with only two dimensions (principal component)? Does the outcome correspond with your statement in b? what is a good dimension or range of dimensions that would yield a good precision?
d. retrain and run KNN with and without the PCA separately, how is the time consumption used in these two cases? explain why PCA maintains precision while reducing the time used to predict.

5. Now retrain the data and run the drawYourOwnDigit part to create your own digit. Try to draw some numbers using the drawing program included with this activity. How can you draw the numbers so that the KNN predictor can easily recognize your number? Try to draw numbers at the edge and observe the change in the accuracy of the KNN predictor. Explain your observation. Try to draw numbers that are much smaller than the canvas and observe the change in the accuracy of the KNN predictor. Explain your observation. How can you improve the KNN classifier based on your observation?

Solutions:

1.
   a. Euclidean Distance, Manhattan Distance, Chebyshev Distance are all d ecent choices to calculate distance between points.
   b. KNN doesn't have clear linear boundary as it depends on the newpoin t's neighbor. A k that is too small would be more prone to overfitt ing and a super large k would make the predictor subject to noise an d alike elements
   c. The performance should improve as the border between the three color is less vertical, which reflects the distribution of the data.

      code:

      *C. distance += abs(row1[i] - row2[i]) -> distance += (row1[i] - row2 [i]) ** 2*

2. The predictor will not work fine with K = 200, since it is greater than the sample number of any single digit, it takes great consideration of noise an d other digits. the good range for K would be 20 - 70.

3. α:1, -1 variance: [10.66666667 0.66666667]

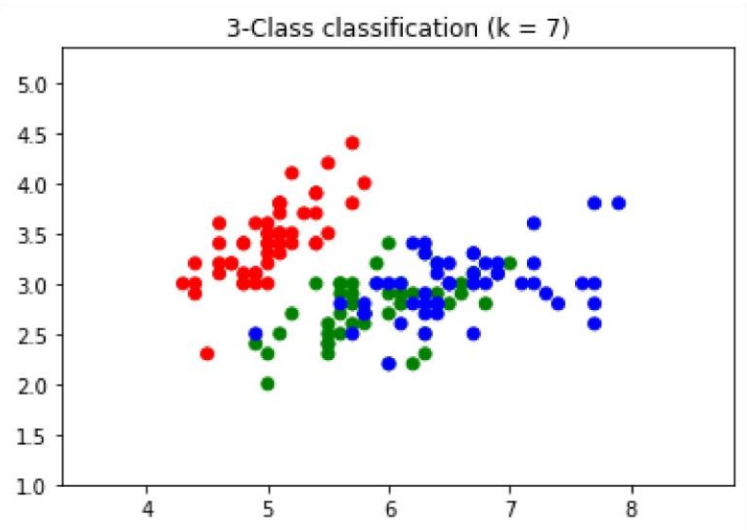
4.
   a. Numbers can be easily seen that they are in cluster in different col ors.
   b. (other answers also apply) 5 and 8. They have similar layouts in han dwriting and thus they've got similar principle component with PCA. The eaiser ones would be 4, 7 and 0 as they distinct from other numb ers in the plot
   c. The precision is very low (error rate ~45% - 60%) . Yes as the miscl assified numbers are mostly those in the middle part of the plot. di mension greater than 10 all performs pretty well.
   d. The time consumption is significantly reduced with 10 dimensions (0. 012s vs 0.5s). As PCA maintains the meaningful features and decrease the dimension from 1024 to 10, the computational complexity is decre ased by 100 times.

5. Drawing numbers near the edge of the canvas will reduce the accuracy, since the neighbours are all far away from the vector generated in the drawing, the nearest neighbour does not give much information about your drawing.
   Drawing numbers that are much smaller than the canvas will reduce the accuracy due to a similar reason. There are no neighbours nearby and thus the nearest neighbour is not very relevant in this situation.
   The KNN classifier can be improved by centering the drawings by shifting the drawings until there are around the same number of pixels to the left and right (up and down) of the center. Thus, drawings at the edge of the canvas can be correctly classified. Also, notice that the centering process in PCA will automatically apply centering to your drawing. This is one of the benefits of PCA.
   The KNN classifier can be improved by adjusting the size of the image so that the drawings are of similar size to the training set. Thus, smaller or larger drawings can be correctly classified after scaling.

In [2]:

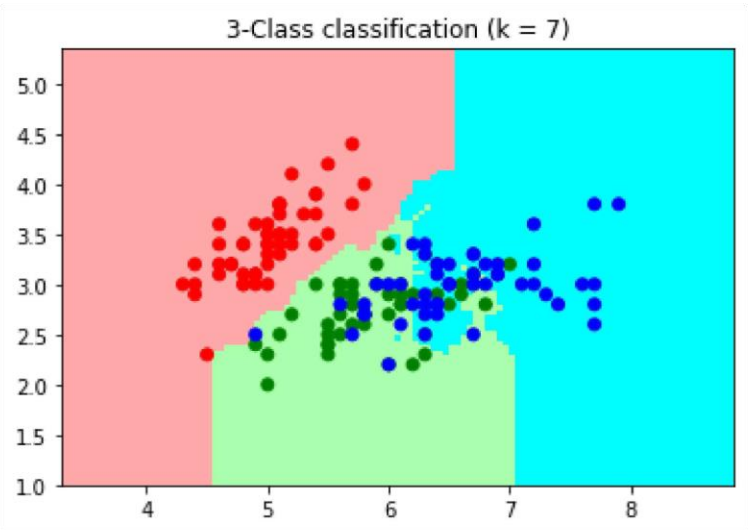
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import datasets

from math import sqrt


# calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
    dis ance = 0.
    for i in range(len(row1 ):
        # distance += (row1[i] - row2[i]) ** 2
        distance += ab (row1[i] - row2[i]
    return sqrt(distance)


# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors


# Make a classification prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction


# kNN Algorithm
def k_nearest_neighbors(train, test, num_neighbors):
    predictions = list()
    for row in test:
        output = predict_classification(train, row, num_neighbors)
        predictions.append(output)
    return predictions


# Adjust k, the number of neighbors here:
n_neighbors = 7

# import some data to play with
iris = datasets.load_iris()

# prepare data
X = iris.data[:, :2]
y = iris.target
```

```
h = .05

# Create color maps
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA','#00FFFF'])
cmap_bold = ListedColormap(['#FF0000', '#008000','#0000FF'])

# calculate min, max and limits
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

plt.sca ter(X[:,  ], X[:, 1], c=y, cmap=cmap_bold)
plt.xlim(xx.min(), xx.max()
plt.ylim(yy.min(), yy.max())
plt.title("3-Class cla sification (k = %i " % n_neighbors)
plt.show()
```



In [3]:

```
# predict class using data and kNN classifier
Z = k_nearest_neighbors(np.concatenate((X, y.reshape(-1,1)), axis=1), np.c_|
Z = np.asarray(Z)

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

# Plot also the training points
plt.sca ter(X[:,  ], X[:, 1], c=y, cmap=cmap_bold)
plt.xlim(xx.min(), xx.max()
plt.ylim(yy.min(), yy.max())
plt.title("3-Class cla sification (k = %i " % n_neighbors)
plt.show()
```



In [ ]:

In [5]:

```
import random
import time

import numpy as np
import matplotlib.pyplot as plt

from os import listdir
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.decomposition import PCA



import sys
!{sys.executable} -m pip install pygame
# !python pygame-1.9.6/setup.py
import draw_digits as game
```

```
Requirement already satisfied: pygame in c:\users\owner\anaconda3\lib\site-
packages (1.9.6)
```

6

```
def img2vector(filename):

    returnVect = np.zeros((1, 1024))
    fr = open(filename)

    for i in range(32):
        lineStr = fr.readline()
        for j in range(32):
            returnVect[0, 32*i+j] = int(lineStr[j])

    return returnVect
def plotNumbers(trainingMat, hwLabels):

    plt.scatter(trainingMat[:, 0], trainingMat[:, 1],
            c=hwLabels, edgecolor='none', alpha=0.5)
    plt.set_cmap(plt.cm.get_cmap('nipy_spectral',10))

    plt.xlabel('component 1')
    plt.ylabel('component 2')
    plt.colorbar()
    plt.show()

def drawYourOwnDigits():
    game.play()
    myDigits = listdir('myDigits')
    m = len(myDigits)
    myDigitMat = np.zeros((m, 1024))
    classNumbers = []
    for i in range(m):
        fileNameStr = myDigits[i]

        classNumbers.append(int(fileNameStr.split('_')[0]))
        myDigitMat[i,:] = img2vector('myDigits/%s' % (fileNameStr))
    return myDigitMat, classNumbers
```

7

```
# data is shuffled and thus the training and testing samples are different e

hwLabels = []
trainingFileList = listdir('trainingDigits')
random.shuffle(trainingFileList)
testFileList = trainingFileList[0:200]
trainingFileList = trainingFileList[201:len(trainingFileList)-1]

m = len(trainingFileList)

trainingMat = np.zeros((m, 1024))
TestMat = np.zeros((200, 1024))
classNumbers = []

for i in range(m):
    fileNameStr = trainingFileList[i]

    classNumber = int(fileNameStr.split('_')[0])
    hwLabels.append(classNumber)
    trainingMat[i,:] = img2vector('trainingDigits/%s' % (fileNameStr))

for i in range(200):

    fileNameStr = testFileList[i]

    classNumbers.append(int(fileNameStr.split('_')[0]))

    TestMat[i,:] = img2vector('trainingDigits/%s' % (fileNameStr))

print(trainingMat.shape)
print(TestMat.shape)
```

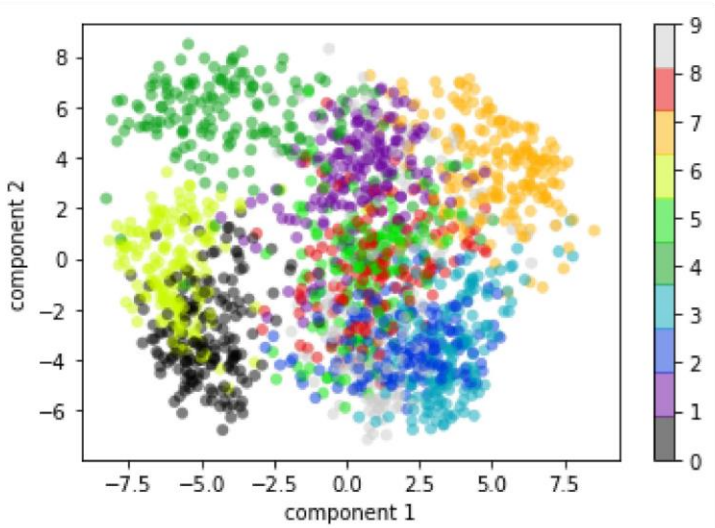
```
(1732, 1024)
(200, 1024)
```

In [ ]:

```
# run this part to make your own test cases :)
# make sure retrain the data before run this part
# Press s or Esc to finish drawing


TestMat, classNumbers = drawYourOwnDigits()
```

8

```
# Run this part to pca the data

NumOfDimensions = 10



pca_model = PCA(n_components=NumOfDimensions)
pca_model.fit(trainingMat)
trainingMat = pca_model.transform(trainingMat)
TestMat = pca_model.transform(TestMat)
plotNumbers(trainingMat, hwLabels)


print(trainingMat.shape)
print(TestMat.shape)
```




```
(1732, 10)
(200, 10)
```

9

```
n_neighbors = 30


neigh =KNN(n_neighbors, algorithm = 'auto')
neigh.fit(trainingMat, hwLabels)
mTest = len(TestMat)



errorCount = 0.0
start_time = time.time()
classifierResult = neigh.predict(TestMat)
for i in range(mTest):
    if(classifierResult[i] != classNumbers[i] or mTest < 15):
        print("classified: %d\ real result:%d" % (classifierResult[i], int(
        if(classifierResult[i] != classNumbers[i]):
            errorCount += 1.0
end_time = time.time()
print("Time used to predict was %g seconds" % (end_time - start_time))
print("Total misclassified data : %d \n in rate of%f%%" % (errorCount, erro
```

```
classified: 7\ real result:9
classified: 5\ real result:8
classified: 8\ real result:3
classified: 9\ real result:5
classified: 9\ real result:1
classified: 5\ real result:3
classified: 9\ real result:5
classified: 1\ real result:8
classified: 4\ real result:0
Time used to predict was 0.0160301 seconds
Total misclassified data : 9
 in rate of4.500000%
```