


REPORT



- K-Nearest Neighbor -

HW2 : mnist data classification



과목명	1	인공지능
담당교수	1	박 준
학년	1	4학년
학번	1	B411241
이름	1	홍성민
제출일	1	20.04.27

1. 과제 개요

과제목표는 K-Nearest Neighbor 알고리즘을 이용하여 mnist data classification(분류) 문제 해결해 보고 정확도를 측정하는 것이다.

Input Feature로는 모든 feature를 사용하는 경우와 input feature를 자신만의 방식으로 가공해서 차수를 줄여서 사용하는 경우 두가지가 있다.

알고리즘은 K-Nearest Neighbor에서 majority vote 와 weighted majority vote 두가지를 모두 사용하였고 K = 3, 5, 10 으로 총 세가지 종류의 K 값에 대해 테스트 하였다.

2. 구현 환경

언어 : python3 / tool : pycharm2019.3.4 community / 운영체제 : window10

3. Hand-crafted feature (Feature Compression) 방안

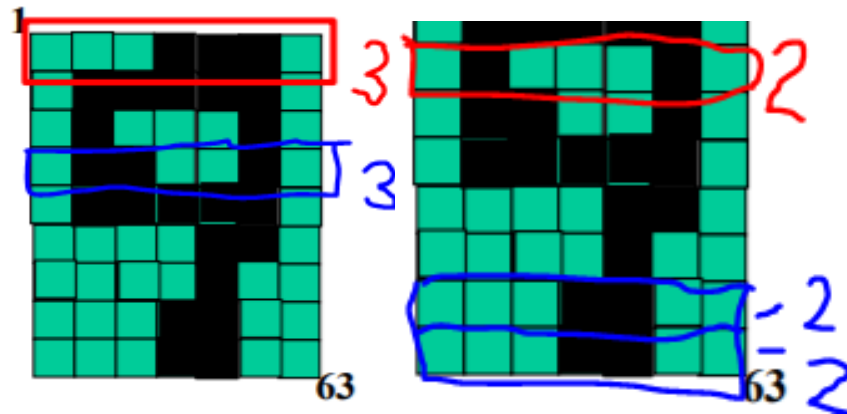
- KNN (K-Nearest Neighbor) 알고리즘

KNN 알고리즘에 대해서는 Iris data classification에서 설명하였으므로 간략히 적도록 하겠다. KNN은 지도학습 중 분류 문제에 사용하는 알고리즘이며 Train 데이터 셋을 세팅한 뒤 새로 들어온 Test 데이터 X에 대해 가장 가까운 거리를 가진 데이터 K개를 순서대로 찾은 뒤 K개의 데이터의 정보를 가지고 majority vote나 weighted majority vote를 적용하는 것이다. majority vote의 경우 K개의 데이터 중 가장 많이 나온 집합의 것을 답으로 상정하는 것이며 weighted majority vote는 거리에 반비례한 가중치를 적용하여 그것의 합이 가장 높은 집합을 답으로 상정하는 방법을 말한다. 본 과제에서는 두가지 방법을 모두 사용한다.

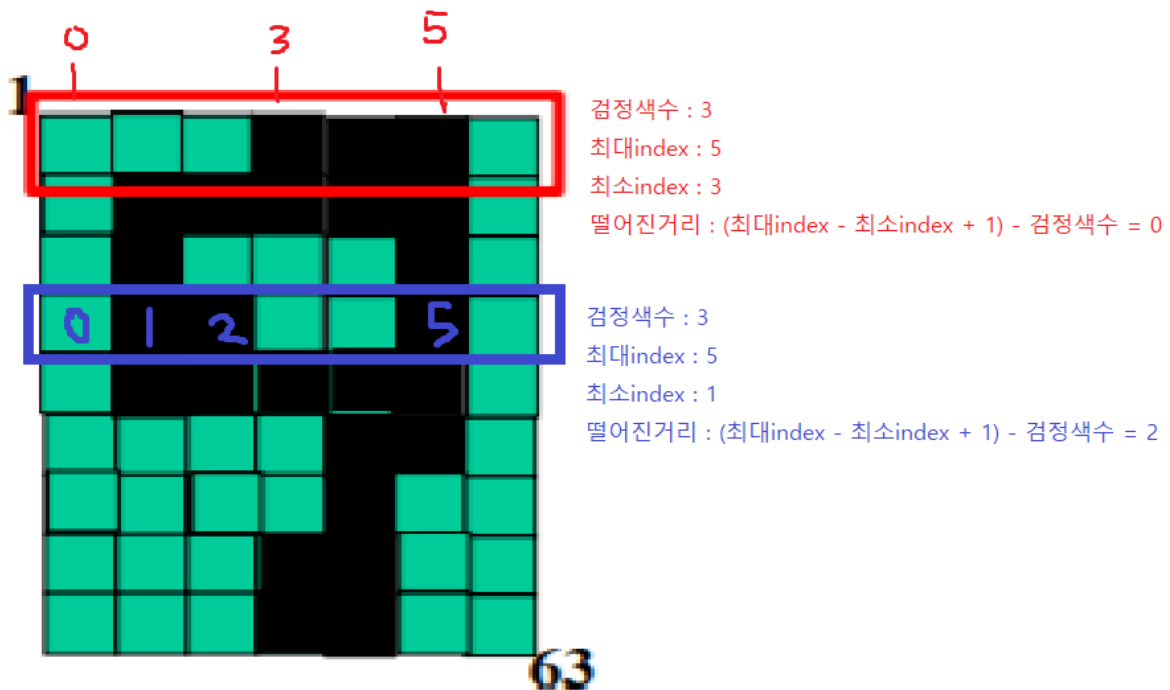
- Hand-crafted feature 방안

MNIST 데이터는 28x28으로 총 784개의 feature을 가진 이미지 데이터이다. 784개의 모든 feature를 통해 학습을 진행하게 될 경우 상당히 오랜 시간이 소요된다. 그러므로 모든 feature을 사용하는 것이 아니라 적당히 feature를 가공하여 차수를 줄이는 방안이 요구된다.

처음으로 생각한 Hand-crafted feature로는 (x, y축) 28개의 행과 28개의 열에 대해 색칠된 즉, 0이 아닌 값을 카운트한 것을 feature로 하는 방법이다. 이렇게 될 경우 feature의 총 개수는 56개가 되어 feature의 개수는 1/14로 줄어들게 된다.



그러나 위와 같이 7x7의 이미지 예시를 보게 되면 Hand-crafted feature에 문제가 있다는 것을 알 수 있다. 빨간색으로 표시된 행과 파란색으로 표시된 행을 보면 둘의 feature값이 같다는 것을 알 수 있다. 왼쪽 그림의 경우 빨간색 구역은 총 3개의 붙어있는 검은색으로 3값을 가지게 되며 파란색 구역은 2개, 1개가 떨어져 있는 3값을 가지게 된다. 그러므로 검은색(칠해진 구역)이 붙어 있는지 붙어있지 않은지 판별할 수 없다는 것이다. 그러므로 칠해진 구간이 얼마나 떨어져 있는지도 feature로 반영해야 한다고 생각하였고 이것에 대한 방안은 다음과 같다.



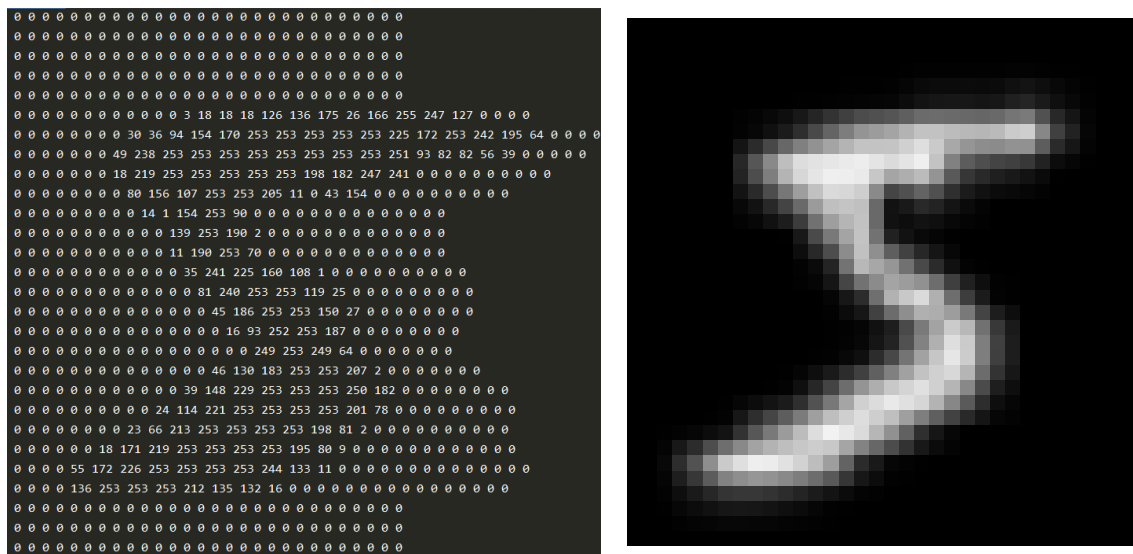
떨어진 거리는 색칠되어 있지 않는 빈공간의 크기와 같다. 그러므로 빈공간의 크기를 구하면 된다. “떨어진 거리 = (최대index - 최소index + 1) - 검정색 수” 이와 같은 식을 통해 복잡한 조건문이 들어가는 코딩을 하지 않더라도 검은색 사이의 빈 공간 크기를 구할 수 있다. 이렇게 각 행과 열에 대해 검은색 사이의 빈 공간크기를 feature로 넣게 되면 56(28+28)개의 feature가 추가되고 기존 feature와 더해져 총 112개의 feature를 가지게 될 것이다. 최종적으로 feature는 1/7로 줄어들게 된다.

4. MNIST 데이터에 대한 설명

MNIST는 인공지능 분야 LeCun교수가 만든 데이터 셋이고 현재 딥러닝을 공부할 때 매우 많이 이용되는 데이터 셋이다. MNIST는 60,000개의 Train 데이터 셋과 10,000개의 Test 데이터 셋으로 이루어져 있고 Train 데이터 셋이 학습데이터로 사용되고 Test 데이터 셋을 프로그램을 검증하는 데에 사용한다. MNIST는 손으로 쓰여진 숫자 이미지들로 구성되어 있고 각각의 이미지 데이터는 0에서 255까지의 값을 갖는 고정 크기 이미지 (28x28 픽셀)로 표준화 되어있고 숫자는 중심에 배치 되어있다.

4.1 Input Feature

data set feature로는 이미지 픽셀 28x28에 대해 0에서 255의 값을 가지는 총 784개의 feature로 되어 있다.



위는 실제로 Train data의 첫번째 이미지를 출력한 결과이다. 왼쪽은 데이터가 실제로 가지고 있는 값을 출력한 결과이며 오른쪽은 이미지로 변환한 출력 결과이다. 0에 가까울수록 칠해지지 않은 구역을 의미하며 255에 가까울수록 진하게 칠한 구역을 의미한다. 검은색 바탕의 경우 모두 0 값을 가지고 있다. 이 데이터는 normalize를 통해 0~1 사이의 실수 값으로 바꿀수도있다.

4.2 Target Output

data set 답으로는 0부터 9까지로 총 10개의 분류 집합이 있다. 10진수 숫자를 의미한다.

5. 소스코드 설명

5.1 KNN 클래스 (knnclass.py) 변경점, 소스코드 및 구현설명

<변경점>

(1) 클래스 변수의 변경점

```
self.ylen = yname.__len__()          # yname 길이
self.disarr = []                     # distance array 줄임말, [거리, 답] 을 담고있는 리스트
self.vote_cnt = [[0., 0], [0., 1], [0., 2], [0., 3], [0., 4], [0., 5], [0., 6], [0., 7], [0., 8], [0., 9]]
self.grapcor = []                    # 그래프 출력을 하기위해 mv, wmv 의 결과를 순서대로 저장하는 전역 리스트
self.sqrtdim_double = int((xtrain[0].__len__() ** 0.5)*2) # 784개 feature을 (28+28)개의 feature로 계산한값
self.hand_xtrain = []                # (56+56)개의 feature로 변환한 train data를 저장하는 리스트
```

답(yname) 리스트의 길이를 가지고 있는 ylen 변수를 추가하였다. 기존 Iris data 분류에서 크기가 3이었던 vote_cnt 리스트의 크기를 10으로 증가시켰다. sqrtdim_double 이라는 변수가 추가 되었으며 이것은 hand craft feature에서 이용하는 변수로 NxN 개의 feature를 가진 Train 데이터가 들어왔을 때 2N값을 계산하여 저장하는 변수이다. 마지막으로 hand_xtrain 리스트가 추가 되었다. 마찬가지로 hand_xtrain 리스트도 hand craft feature에서 이용하는 리스트로 변경된 feature를 가진 데이터 셋을 저장하는 리스트이다.

(2) 수정된 함수

```
def get_nearest_k(self, xtest_i, option):
```

```
def get_nearest_k(self, xtest_i, option):          # 가장가까운 점부터 순서대로 disarr 리스트에 저장해주는 함수
    if option == 'feature_compression':            # option이 'feature_compression' 인 경우
        dim_len = self.sqrtdim_double * 2          # dim_len = 56 * 2 = 108
        xtesti = self.change_feature(xtest_i)      # test데이터의 feature를 변경한다.
        x_train = self.hand_xtrain                  # 변경된 feature를 가진 train데이터를 사용한다.
    else :
        dim_len = self.dim                          # feature수 = self.dim
        xtesti = xtest_i                            # test데이터 그대로 사용
        x_train = self.xtrain                        # train데이터 그대로 사용

    for j in range(len(self.hand_xtrain)):          # for 문 (train data set 모두에 대하여)
        res = 0.                                     # disarr 리스트에 담을 거리 값
        for i in range(0, dim_len):                  # dimension 즉, feature 갯수만큼에 대하여 거리를 계산
            res = res + (float(xtesti[i]) - float(x_train[j][i]))**2 # (train0 - test0)^2 + ... + (trainN - testN)^2
        res = res**0.5                                # 마지막에 0.5 승으로 루트를 씌워준다.
        self.disarr.append((res, int(self.ytrain[j]))) # disarr에 [res, 답] 을 append 해준다.
    self.disarr.sort(key=itemgetter(0))              # 최종적으로 disarr를 res 기준으로 sorting 한다 (오름차순)
```

option이라는 인자를 추가하여 option이 'feature_compression' 인 경우 hand craft feature를 사용하는 get_nearest_k로 적용되게끔 하였다. option이 'feature_compression' 이 아닌 경우 기존 get_nearest_k과 동일하게 적용되며 if else 문을 통해 for문에 사용되는 거리, train 데이터 셋, test 데이터를 설정하였다.

```
- def reset(self):
```

vote_cnt 리스트가 변경되었으므로 reset 함수에서도 동일하게 vote_cnt를 변경하였다.

(3) 추가된 함수

```
def change_feature(self, data_i):
```

```
def change_feature(self, data_i):                # 데이터 하나를 넣으면 784개 feature를 (56 + 56)개의 feature로 변환하는 함수
    dim_len = int(self.sqrtdim_double / 2)        # sqrtdim_double = 56이다. dim_len = 28
    flist = []                                    # (56 + 56)개의 feature로 변환된 데이터를 담을 리스트
    # 행에 대하여
    for i in range(0, dim_len):                  # 28 행
        (x_max, x_min, res) = (-1, dim_len, 0)    # 색칠되어있는 최대 index, 최소 index, 색칠된 갯수 res 초기값 설정
        for j in range(0, dim_len):              # 28 열
            if data_i[i*dim_len + j] > 5:         # train데이터의 [i][j]값이 5이상이면 칠해져있다고 판단
                res += 1                          # 색칠된 수 +1
                if x_max < j:                     # 최대 index가 현재 index보다 작다면
                    x_max = j                    # 최대 index는 현재 index로
                if x_min > j:                     # 최소 index가 현재 index보다 크다면
                    x_min = j                    # 최소 index는 현재 index로
            if x_max == -1 or x_min == dim_len:    # 만일 아무것도 칠해지지 않은 경우이다
                flist.append(0)                  # i 행 검은색 사이 빈 공간 수(0) feature로 append
            else:                                  # 칠해져 있는게 있는 경우
                flist.append(x_max - x_min - res + 1) # i 행 검은색 사이 빈 공간 수 feature로 append
            flist.append(res)                     # i 행에 칠해져 있는 갯수 feature로 append
    # 열에 대하여
    for i in range(0, dim_len):                  # 28 행
        (y_max, y_min, res) = (-1, dim_len, 0)    # 색칠되어있는 최대 index, 최소 index, 색칠된 갯수 res 초기값 설정
        for j in range(0, dim_len):              # 28 열
            if data_i[j*dim_len + i] > 5:         # train데이터의 [j][i]값이 5이상이면 칠해져있다고 판단 열을
                res += 1                          # 색칠된 수 +1
                if y_max < i:                     # 최대 index가 현재 index보다 작다면
                    y_max = i                    # 최대 index는 현재 index로
                if y_min > i:                     # 최소 index가 현재 index보다 크다면
                    y_min = i                    # 최소 index는 현재 index로
            if y_max == -1 or y_min == dim_len:    # 만일 아무것도 칠해지지 않은 경우이다
                flist.append(0)                  # i 열 검은색 사이 빈 공간 수(0) feature로 append
            else:                                  # 칠해져 있는게 있는 경우
                flist.append(y_max - y_min - res + 1) # i 열 검은색 사이 빈 공간 수 feature로 append
            flist.append(res)                     # i 열에 칠해져 있는 갯수 feature로 append
    return flist                                  # flist 반환
```

data_i(data 하나)가 들어왔을 때 784(28x28)개의 feature를 108개의 hand craft feature로 변경시켜 반환해주는 함수이다. 행과 열에 대한 for문으로 나누어져 있으며 각각의 행과 열에 대하여 칠해진 공간을 카운트하여 flist 리스트에 append한다. 또한 최대index - 최소index - res + 1 값 즉, 칠해진 공간 사이의 빈 공간 크기도 행과 열에 대해 모두 flist 리스트에 append한다. 최종적으로 flist를 반환함으로서 변경된 feature를 가진 데이터를 얻을 수 있다

```
def handcraft_feature(self):
```

```
def handcraft_feature(self):                # 기존 train 데이터의 feature를 변환한 데이터를 hand_xtrain 리스트에 채워주는 함수
    for k in range(len(self.xtrain)):        # train 데이터의 크기 만큼 for 문
        self.hand_xtrain.append(self.change_feature(self.xtrain[k])) # change_feature함수를 적용, hand_xtrain에 append
```

change_feature 함수를 이용하여 기존 train 데이터를 hand craft feature로 변경하여 hand_xtrain 리스트에 저장해주는 함수이다. train 데이터 크기만큼 for문을 돌며 change_feature 함수를 실행한다.

<소스 코드 및 구현설명>

```
1 from operator import itemgetter # 리스트 sorting 시 특정 key 값으로 정렬하기 위함
2
3 class KNN():
4     def __init__(self, k, xtrain, ytrain, yname):
5         self.k = k # KNN 에서의 k 값
6         self.xtrain = xtrain # train data set feature
7         self.ytrain = ytrain # train data set 답
8         self.yname = yname # 숫자 0~9 이름 리스트
9         self.dim = xtrain[0].__len__() # dimension 즉, data set feature 종류
10        self.ylen = yname.__len__() # yname 길이
11        self.disarr = [] # distance array 출임말, [거리, 답] 을 담고있는 리스트
12        self.vote_cnt = [[0., 0], [0., 1], [0., 2], [0., 3], [0., 4], [0., 5], [0., 6], [0., 7], [0., 8], [0., 9]]
13        self.grapcor = [] # 그래프 출력을 하기위해 mv, vmv 의 결과를 순서대로 저장하는 전역 리스트
14        self.sqrtdim_double = int((xtrain[0].__len__() ** 0.5)*2) # 784개 feature를 (28+28)개의 feature로 계산한값
15        self.hand_xtrain = [] # (56+56)개의 feature로 변환한 train data를 저장하는 리스트
16
17    def show_dim(self): # 변수 dim 을 출력하는 함수
18        print("dim : ", self.dim)
19
20    def change_feature(self, data_i): # 데이터 하나를 넣으면 784개 feature를 (56 + 56)개의 feature로 변환하는 함수
21        dim_len = int(self.sqrtdim_double / 2) # sqrtdim_double = 56이다. dim_len = 28
22        flist = [] # (56 + 56)개의 feature로 변환된 데이터를 담은 리스트
23        # 행에 대하여
24        for i in range(0, dim_len): # 28 행
25            (x_max, x_min, res) = (-1, dim_len, 0) # 색칠되어있는 최대 index, 최소 index, 색칠된 갯수 res 초기값 설정
26            for j in range(0, dim_len): # 28 열
27                if data_i[i*dim_len + j] > (5/255): # train데이터의 [i][j]값이 5/255 이상이면 칠해졌다고 판단
28                    res += 1 # 색칠된 수 +1
29                    if x_max < j: # 최대 index가 현재 index보다 작다면
30                        x_max = j # 최대 index는 현재 index로
31                    if x_min > j: # 최소 index가 현재 index보다 크다면
32                        x_min = j # 최소 index는 현재 index로
33            if x_max == -1 or x_min == dim_len: # 만일 아무것도 칠해지지 않은 경우이다
34                flist.append(0) # i 행 검은색 사이 빈 공간 수(0) feature로 append
35            else: # 칠해져 있는게 있는 경우
36                flist.append(x_max - x_min - res + 1) # i 행 검은색 사이 빈 공간 수 feature로 append
37            flist.append(res) # i 행에 칠해져 있는 갯수 feature로 append
38        # 열에 대하여
39        for i in range(0, dim_len): # 28 행
40            (y_max, y_min, res) = (-1, dim_len, 0) # 색칠되어있는 최대 index, 최소 index, 색칠된 갯수 res 초기값 설정
41            for j in range(0, dim_len): # 28 열
42                if data_i[j*dim_len + i] > (5/255): # train데이터의 [j][i]값이 5/255 이상이면 칠해졌다고 판단 열을
43                    res += 1 # 색칠된 수 +1
44                    if y_max < i: # 최대 index가 현재 index보다 작다면
45                        y_max = i # 최대 index는 현재 index로
46                    if y_min > i: # 최소 index가 현재 index보다 크다면
47                        y_min = i # 최소 index는 현재 index로
48            if y_max == -1 or y_min == dim_len: # 만일 아무것도 칠해지지 않은 경우이다
49                flist.append(0) # i 열 검은색 사이 빈 공간 수(0) feature로 append
50            else: # 칠해져 있는게 있는 경우
51                flist.append(y_max - y_min - res + 1) # i 열 검은색 사이 빈 공간 수 feature로 append
52            flist.append(res) # i 열에 칠해져 있는 갯수 feature로 append
53        return flist # flist 반환
54
55    def handcraft_feature(self): # 기존 train 데이터의 feature를 변환한 데이터를 hand_xtrain 리스트에 채워주는 함수
56        for k in range(len(self.xtrain)): # train 데이터의 크기 만큼 for 문
57            self.hand_xtrain.append(self.change_feature(self.xtrain[k])) # change_feature함수를 적용, hand_xtrain에 append
58
59    def get_nearest_k(self, xtest_i, option): # 가장가까운 점부터 순서대로 disarr 리스트에 저장해주는 함수
60        if option == 'feature_compression': # option이 'feature_compression' 인 경우
61            dim_len = self.sqrtdim_double * 2 # dim_len = 56 * 2 = 112
62            xtesti = self.change_feature(xtest_i) # test데이터의 feature를 변경한다.
63            x_train = self.hand_xtrain # 변경된 feature를 가진 train데이터를 사용한다.
64        else :
65            dim_len = self.dim # feature수 = self.dim
66            xtesti = xtest_i # test데이터 그대로 사용
```

```

67         x_train = self.xtrain                # train데이터 그대로 사용
68
69         for j in range(len(x_train)): # for 문 (train data set 모두에 대하여)
70             res = 0.                        # disarr 리스트에 담을 거리 값
71             for i in range(0, dim_len):     # dimension 즉, feature 갯수만큼에 대하여 거리를 계산
72                 res = res + (float(xtesti[i]) - float(x_train[j][i]))**2 # (train0 - test0)^2 + ... + (trainN - testN)^2
73             res = res**0.5                  # 마지막에 0.5 승으로 루트를 씌워준다.
74             self.disarr.append((res, int(self.ytrain[j]))) # disarr에 [res, 답] 을 append 해준다.
75             self.disarr.sort(key=itemgetter(0)) # 최종적으로 disarr를 res 기준으로 sorting 한다 (오름차순)
76
77     def mv(self): # Majority Vote 함수
78         for j in range(0, self.k):          # for 문 (k 값 만큼)
79             self.vote_cnt[self.disarr[j][1]][0] += 1 # 가중치는 그냥 갯수이므로 +1 로 해주고, disarr의 0부터 k개를 본다.
80             self.vote_cnt.sort(key=itemgetter(0), reverse=True) # vote_cnt에서 가장큰 가중치를 알기 위해 sorting reverse (내림차순)
81             self.grapcor.append(self.vote_cnt[0][1]) # 결과를 grapcor 리스트에 넣어준다.
82             return self.ynome[self.vote_cnt[0][1]] # 결과를 이름으로 반환한다.
83
84     def wmv(self): # Weighted Majority Vote 함수
85         for j in range(0, self.k):          # for 문 (k 값 만큼)
86             self.vote_cnt[self.disarr[j][1]][0] += (1 / self.disarr[j][0]) # 가중치는 1/거리, disarr의 0부터 k개를 본다.
87             self.vote_cnt.sort(key=itemgetter(0), reverse=True) # vote_cnt에서 가장큰 가중치를 알기 위해 sorting reverse (내림차순)
88             self.grapcor.append(self.vote_cnt[0][1]) # 결과를 grapcor 리스트에 넣어준다.
89             return self.ynome[self.vote_cnt[0][1]] # 결과를 이름으로 반환한다.
90
91     def set_k(self, k): # k 값을 설정하는 set 함수
92         self.k = k
93
94     def get_grapcor(self): # grapcor 리스트를 반환해주는 get 함수
95         return self.grapcor
96
97     def reset_grapcor(self): # grapcor 리스트를 초기화하는 함수
98         self.grapcor = []
99
100    def reset(self): # disarr 리스트와 vote_cnt 리스트를 초기화 하는 함수
101        self.disarr = []
102        self.vote_cnt = [[0., 0], [0., 1], [0., 2], [0., 3], [0., 4], [0., 5], [0., 6], [0., 7], [0., 8], [0., 9]]

```

클래스 형식은 Iris data classification 때와 동일 하다. 하지만 다른 보편적인 데이터가 들어와도 클래스가 작동할 수 있도록 고정값 대신 들어오는 데이터 셋에 따라 크기를 측정하여 변수값에 넣어 유동적으로 클래스 함수가 작동할 수 있게끔 구현하였다. `sqrt(dim_double)` 변수의 경우 들어오는 train 데이터 셋의 feature 수를 루트를 씌운 후 x2한 값으로 행과 열의 개수를 의미한다. 결과가 양의 정수이지만 28.0 과 같이 float으로 인식되는 경우가 있으므로 인덱스 형식으로 사용되므로 `int()` 를 통해 형변환을 해주었다.

`change_feature` 함수는 새로 추가된 함수로 기존 데이터를 hand craft feature로 변경하여 반환해주는 함수이다. 인자는 `data_i`로 데이터 셋이 아니라 데이터 하나를 의미한다. train 데이터 셋만 변경해야 하는 것이 아니라 test 데이터가 하나씩 들어올 때 test 데이터에 대해서도 feature를 변경해주어야 하므로 데이터 셋이 아닌 데이터 하나씩을 인자로 받아 반환하게 한 것이다. hand craft feature는 56개의 행과 열을 기준으로 생성했으며 행과 열에 대한 for문 두개로 나누어져 구현 되어있다. 결론적으로 기존 784개의 feature를 두번씩 보며 feature를 변환하고 있는 형식이며 칠해진 칸을 판단하는 기준은 $0/255 \sim 255/255$ 값을 가지는 feature가 $5/255$ 가 넘는 경우에만 칠해져 있다고 판단한다. 칠해진 개수는 카운트하여 `klist` 리스트에 append하였고 칠해진 경우가 생길 때마다 최대, 최소 index를 계산해주며 칠해진 공간의 크기를 계산하는 식을 마지막에 적용하였다. 단, 칠해진 공간이 아예 없게 될 경우 $\text{최대index} - \text{최소index} - \text{res} + 1$ 값이 0이 아니라 음수 값이 들어가게 되므로 해당 경우에는 0을 append 하도록 예외 처리 하였다. 각각의 행과 열은 두개의 feature를 가지게 되는 것으로 처음엔 빈공간의 크기, 두번째는 칠해진 공간의 개수

가 들어가게 된다.

handcraft_feature 함수 또한 새로 추가된 함수로 train 데이터 셋 전체에 chang_feature 함수를 적용하기 위해 만들었다. train 데이터 셋 크기만큼 for문을 돌면서 change_feature 함수를 적용해 반환되는 리스트를 hand_xtrain 리스트에 append 하도록 구현 되어있다.

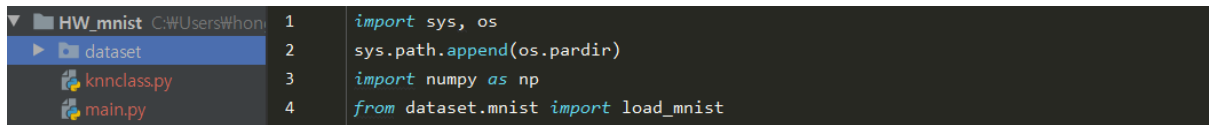
get_nearest_k 함수는 기존과 동일하게 xtest_i가 테스트 데이터를 의미한다. 하지만 모든 feature를 사용하는 경우와 hand craft feature를 적용하는 경우에 따라 계산에 들어가는 리스트가 달라지므로 함수에 option이라는 인자를 추가하여 option에 따라 두가지 경우에 대해 모두 계산할 수 있게끔 하였다. feature 압축을 의미하는 option == 'feature_compression' 인 경우 handcraft feature를 적용한 get_nearest 함수가 실행 되며 xtest_i가 거리를 비교하는 train 데이터셋의 종류가 hand_xtrain으로 설정된다. feature 수를 의미하는 dim_len은 sqrt(dim_double * 2) 로 112개의 feature 수를 의미하는 값이 된다. xtest_i 도 그대로 쓰는 것이 아니라 feature를 변환하는 함수인 chang_feature 함수를 적용하여 xtest_i 변수에 바뀐 데이터를 넣어주어 사용한다. 반대로, option이 'feature_compression' 이 아닌 경우에는 기존과 동일하게 함수가 실행되고 이것은 모든 feature를 사용하게 되어 어떤 feature를 가진 데이터에 대해서도 돌아가는 통상적인 함수가 된다.

majority vote와 weighted majority vote를 의미하는 함수 mv와 wmv는 수정한 부분이 없으며 기존과 동일하다.

5.2 메인 함수 (main.py) 변경점, 소스코드 및 구현설명

<변경점>

- 데이터 참조 변경사항



```
1 import sys, os
2 sys.path.append(os.pardir)
3 import numpy as np
4 from dataset.mnist import load_mnist
```

일단 Iris data가 아닌 mnist 데이터를 불러오게 된다. mnist 데이터는 7만개의 이미지에 해당하는 데이터로 폴더에 따로 저장되어 있으며 해당 데이터는 main.py 상위 폴더에 함께 들어있다. 그리고 test 데이터의 경우 10,00개의 데이터를 모두 사용하면 시간 관계상 출력결과를 보는 것이 매우 어렵기 때문에 test 데이터는 10,00개의 데이터 중 랜덤한 100개의 데이터를 사용하게 되었다.

- all feature , hand craft feature 상황 추가

$K = 3, 5, 10$ 그리고 majority vote와 weighted majority vote 는 동일하게 쓰이나 이번 과제의 경우 모든 feature을 이용하는 경우와 feature을 압축한 hand craft feature 두가지 경우가 추가되었다. 그러므로 총 결과는 12개가 나오게 된다. 그러므로 기존의 큰 for문을 동일하게 두 번 반복하여 사용하는 형태로 바뀌었다.

- output의 변경사항

Iris data classification에서는 feature의 수가 적은 관계로 기존에 4차원을 2차원 그래프를 출력하더라도 직관적인 결과를 관찰할 수 있었다. 그러나 mnist 데이터는 모든 feature을 사용할 경우 784개, hand craft feature을 사용할 경우 112개로 그래프로 이것을 표현하기에는 너무나 많은 feature라고 판단되어 그래프 출력은 하지 않게 되었다.

```
TestData 95 index( 749 ) Computed class: 1 , True class: 1
TestData 96 index( 6740 ) Computed class: 0 , True class: 9
TestData 97 index( 5816 ) Computed class: 4 , True class: 4
TestData 98 index( 6653 ) Computed class: 7 , True class: 7
TestData 99 index( 6248 ) Computed class: 8 , True class: 8
Accuracy : 0.93
```

추가적으로 computed class와 true class를 비교하여 최종적으로 정확도(Accuracy)를 출력하는 부분을 추가 하였으며 이를 통해 각각의 알고리즘과 변수에 따라 정확도를 비교할 수 있다. 특히 hand craft feature의 경우 내가 고려하고 있는 방법이 얼마나 정확한지 혹은 정확하지 못 한지 알게 되어 hand craft feature 방법을 보완하는데 도움이 된다.

- 구조 변경으로 시간 복잡도 단축

MNIST 데이터의 양이 너무 많기 때문에 모든 K값에 대해, 그리고 majority vote와 weighted majority vote 경우(총 6가지 경우)에 대해 일일이 get_nearest_k 함수를 실행하게 되면 6배의 시간이 들게 된다. 하지만 get_nearest_k 함수를 실행하면 현재 테스트 데이터에 대해서는 거리에 따라 sorting 한 disarr 리스트가 생성되므로 이것을 통해 여러가지 K값과 더불어 majority vote와 weighted majority vote를 계산 할 수 있다. 그러나 결과를 보여주기에 하나의 테스트 데이터에 대해서 K값이 바뀌고, mv와 wmv를 보여주게 되면 직관성이 떨어지므로 출력 결과를 리스트에 정렬하여 모든 함수와 함수의 계산이 끝난 뒤 리스트에 저장한 출력 결과를 최종적으로 출력하는 형식으로 변경하였다.

실제로 모든 feature을 이용하는 경우 $K = 3$, majority vote에 대해 결과를 내는데 1시간 정도의 시간이 걸렸고 기존의 for문 형식으로 6가지 경우 모두를 출력하기 위해서는 6시간이 걸리게 된다. 그러나 한번의 disarr 리스트 정렬로 모든 경우에 대한 결과를 리스트에 저장했다가 출력하게 되면 한번만 실행하는 것과 동일하며 시간은 1시간만 걸리게 된다.

```
for i in range(0, size): # Test data set 에 대한 for 문
    mnist_allfeature.get_nearest_k(x_test[sample[i]], 'Use all Features') # x_test를 하나씩 넣으며 거리가준 disarr
    for j in range(0, 3): # klist 인자로 k = 3, 5, 10 을 돌려주기 위한 for 문
        mnist_allfeature.set_k(klist[j])
        (computed, trueclass) = (mnist_allfeature.mv(), label_name[t_test[sample[i]]])
        mnist_allfeature.reset_vote_cnt() # vote_cnt 리스트만 초기화, disarr 리스트는 그대로
        output[i][j][0] = computed
        if computed == trueclass: # 계산결과 == 실제답 이면
            accuclist[j][0] += 1 / size # 정확도 1 / size 만큼 증가
        (computed, trueclass) = (mnist_allfeature.wmv(), label_name[t_test[sample[i]]])
```

위의 논리로 인해 바깥 for문과 안쪽 for문이 뒤바뀌어야 한다. 먼저 각 테스트 데이터에 대해 for문을 돌게 되며 테스트 데이터 하나에 대해서 K값을 바꿔주며 mv, wmv 함수를 실행하게 된다. 그러므로 실제 거리 계산 연산이 들어가는 get_nearest_k 함수는 데이터 하나에 대해 한번만 실행하게 된다. 구체적인 구현 방법은 코드를 보며 설명하도록 하겠다.

<소스코드 및 구현설명>

```
1 import sys, os
2 sys.path.append(os.pardir)
3 import numpy as np
4 from dataset.mnist import load_mnist
5 from PIL import Image
6 from knnclass import KNN # knnclass import
7 (x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=True)
8 size = 100
9 sample = np.random.randint(0, t_test.shape[0], size)
10 label_name = ['0','1','2','3','4','5','6','7','8','9']
11 mnist_allfeature = KNN(10, x_train, t_train, label_name) # KNN class 객체 생성 Use All Features = 784
12 mnist_handcraft = KNN(10, x_train, t_train, label_name) # KNN class 객체 생성 Compression Features = 102
13 mnist_handcraft.handcraft_feature() # Train data set Features 변환
14 klist = [3, 5, 10] # k = 3, 5, 10 에 대해 돌려주기 위한 리스트
15 # Use All Features 784
16 output = np.zeros((size,3,2))
17 acculist = np.zeros((3,2))
18 for i in range(0, size): # Test data set 에 대한 for 문
19     mnist_allfeature.get_nearest_k(x_test[sample[i]], 'Use all Features') # x_test를 하나씩 넣으며 거리기준 disarr 리스트 완성
20     for j in range(0, 3): # klist 인자로 k = 3, 5, 10 을 돌려주기 위한 for 문
21         mnist_allfeature.set_k(klist[j])
22         (computed, trueclass) = (mnist_allfeature.mv(), label_name[t_test[sample[i]]])
23         mnist_allfeature.rest_voteCnt() # vote_cnt 리스트만 초기화, disarr 리스트는 그대로
24         output[i][j][0] = computed
25         if computed == trueclass: # 계산결과 == 실제답 이면
26             acculist[j][0] += 1 / size # 정확도 1 / size 만큼 증가
27         (computed, trueclass) = (mnist_allfeature.wmv(), label_name[t_test[sample[i]]])
28         mnist_allfeature.rest_voteCnt() # vote_cnt 리스트만 초기화, disarr 리스트는 그대로
29         output[i][j][1] = computed
30         if computed == trueclass: # 계산결과 == 실제답 이면
31             acculist[j][1] += 1 / size # 정확도 1 / size 만큼 증가
32     mnist_allfeature.reset() # 가장 가까운 순으로 정렬된 point 리스트 초기화, majority vote 리스트 초기화
33 for i in range(0, len(klist)):
34     print("Use all Features Majority vote / k = ", klist[i])
35
36 for j in range(0, size):
37     print("TestData", j, "index(", sample[j], ") Computedclass:", int(output[j][i][0]), ", \tTrueclass: ", label_name[t_test[sample[j]]])
38     print("Accuracy :", round(acculist[i][0], 3))
39     print("Use all Features, Weighted Majority vote / k = ", klist[i])
40     for j in range(0, size):
41         print("TestData", j, "index(", sample[j], ") Computedclass:", int(output[j][i][1]), ", \tTrueclass: ", label_name[t_test[sample[j]]])
42         print("Accuracy :", round(acculist[i][1], 3))
43 # Features Compression 112
44 output = np.zeros((size,3,2))
45 acculist = np.zeros((3,2))
46 for i in range(0, size): # Test data set 에 대한 for 문
47     mnist_handcraft.get_nearest_k(x_test[sample[i]], 'feature_compression') # x_test를 하나씩 넣으며 거리기준 disarr 리스트 완성
48     for j in range(0, 3): # klist 인자로 k = 3, 5, 10 을 돌려주기 위한 for 문
49         mnist_handcraft.set_k(klist[j])
50         (computed, trueclass) = (mnist_handcraft.mv(), label_name[t_test[sample[i]]])
51         mnist_handcraft.rest_voteCnt() # vote_cnt 리스트만 초기화, disarr 리스트는 그대로
52         output[i][j][0] = computed
53         if computed == trueclass: # 계산결과 == 실제답 이면
54             acculist[j][0] += 1 / size # 정확도 1 / size 만큼 증가
55         (computed, trueclass) = (mnist_handcraft.wmv(), label_name[t_test[sample[i]]])
56         mnist_handcraft.rest_voteCnt() # vote_cnt 리스트만 초기화, disarr 리스트는 그대로
57         output[i][j][1] = computed
58         if computed == trueclass: # 계산결과 == 실제답 이면
59             acculist[j][1] += 1 / size # 정확도 1 / size 만큼 증가
60     mnist_handcraft.reset() # 가장 가까운 순으로 정렬된 point 리스트 초기화, majority vote 리스트 초기화
61 for i in range(0, len(klist)):
62     print("Features Compression, Majority vote / k = ", klist[i])
63     for j in range(0, size):
64         print("TestData", j, "index(", sample[j], ") Computedclass:", int(output[j][i][0]), ", \tTrueclass: ", label_name[t_test[sample[j]]])
65         print("Accuracy :", round(acculist[i][0], 3))
66     print("Features Compression, Weighted Majority vote / k = ", klist[i])
67     for j in range(0, size):
68         print("TestData", j, "index(", sample[j], ") Computedclass:", int(output[j][i][1]), ", \tTrueclass: ", label_name[t_test[sample[j]]])
69         print("Accuracy :", round(acculist[i][1], 3))
```

import sys, os, sys.path.append(os.pardir)을 통해 부모 디렉토리에서 import할 수 있도록 설정한 뒤 from dataset.mnist import load_mnist로 mnist 데이터를 load 할 수 있는 함수를 import 하였다. load_mnist 함수를 호출하여 (x_train, t_train)과 (x_test, t_test) 리스트에 mnist 데이터를 저장하였다. load_mnist 함수의 인자로 flatten = True를 통해 이미지를 1차원 배열로 읽는 것을 지정해주었고 normalize = True로 설정해 데이터의 feature은 RGB값 0~255사이의 정수를 압축한 0~1사이의 실수 값을 가지게 된다.

```
mnist_handcraft.get_nearest_k(x_test[sample[i]], 'feature_compression')
```

Test 데이터의 경우 시간 관계상 10,00개의 데이터를 모두 사용할 수 없으므로 10,00개의 데이터 중 100개를 random하게 설정하였다. numpy의 randint라는 함수를 사용했으며 randint는 0이상 t_test.shape[0](=10,000) 미만의 int를 random하게 size 개수만큼 뽑아내어 sample이라는 리스트에 저장하게 된다. 이후 sample 리스트에 저장 되어있는 인덱스를 통해 test 데이터에 접근하도록 코드를 구현하였다.

```
mnist_allfeature = KNN(10, x_train, t_train, label_name)
mnist_handcraft = KNN(10, x_train, t_train, label_name)
mnist_handcraft.handcraft_feature()
```

KNN 클래스 객체는 두개를 생성하였으며 allfeature는 모든 feature을 사용하는 클래스 객체를 말하며 handcraft 객체는 hand craft feature을 사용하는 것을 말한다. handcraft의 경우 선언과 같이 handcraft_feature 함수를 호출하여 test 데이터를 변환하여 hand_xtrain 리스트에 저장해주었다. label_name은 0에서 9까지 숫자를 문자로 저장하고 있는 정답의 이름을 저장하고 있는 리스트이다.

```
output = np.zeros((size,3,2))
acculist = np.zeros((3,2))
```

출력결과를 저장해주기 위해 computed class를 모으는 역할인 output numpy array와 정확도를 측정해 저장하고 있는 acculist numpy array를 생성해주었다.

학습이 실행되는 부분은 큰 for문 두개로 구성되어 있으며 첫 번째 for문은 모든 feature을 사용하는 부분이고 두 번째 for문은 hand craft feature을 사용하는 부분이다. 각각의 for문은 우선 test 데이터 셋의 크기만큼 for문을 돌게 되고 get_nearest_k 함수를 실행하여 짧은 거리 순으로 정렬해 disarr 리스트를 완성한다. disarr 리스트가 완성된 이후에 K값을 변경하게 되는 for문을 돌게 되며 majority vote와 weighted majority vote를 차례대로 실행한다. mv 함수와 wmv 함수는 미리 계산되어 있는 disarr 리스트에서 K개의 값을 참조하여 계산하면 vote_cnt 리스트가 완성이 된다. 최종 산출 결과는 vote_cnt 리스트의 값을 보고 결과를 출력하게 된다.

```
mnist_handcraft.reset_voteCnt() # vote_cnt 리스트만 초기화, disarr 리스트는 그대로
```

단 vote_cnt 리스트의 경우 disarr 리스트처럼 반복해서 사용하면 기존 vote 결과 때문에 이후 결과가 제대로 나오지 않게 되므로 vote_cnt 리스트는 mv와 wmv 함수 호출 이후에 reset_voteCnt 함수를 호출하여 항상 초기화 해주었다. 결과인 computed class는 true class와 비교하여 두 값이 같은 경우 accuracy 변수에 1/size 값을 더해줌으로 정확도를 측정하게끔 하였다.

```

for i in range(0, len(klist)):
    print("Features Compression, Majority vote / k = ", klist[i])
    for j in range(0, size):
        print("TestData", j, "index(", sample[j], ") Computedclass:", int(output[j][i][0]), ", \tTrueclass: ", label_name[t_test[sample[j]]])
    print("Accuracy :", round(acculist[i][0], 3))
    print("Features Compression, Weighted Majority vote / k = ", klist[i])
    for j in range(0, size):
        print("TestData", j, "index(", sample[j], ") Computedclass:", int(output[j][i][1]), ", \tTrueclass: ", label_name[t_test[sample[j]]])
    print("Accuracy :", round(acculist[i][1], 3))

```

실제 클래스 함수가 돌아가는 for문이 끝난 이후에는 결과를 저장하고 있는 numpy array 인 output과 acculist를 통해 한번에 결과를 모두 출력해주었다. accuracy의 경우에는 round 함수를 통해 소수점 세 자리 까지 출력하였다.

6. 학습 과정에 대한 설명

학습 과정이 제대로 진행되고 있는지 보기 위하여 get_nearest_k() 함수가 실행 된 후 가까운 거리 순으로 disarr 리스트가 정렬되었는지 출력 해보고 k개의 disarr 리스트 값을 보고 majority vote와 weightedmajority vote 결과를 뽑아내는지 vote_cnt 리스트와 답을 출력해 보았다.

- use all features 학습 과정

```

TestData 5 index( 1295 ) Computed class: 1 , True class: 1
9.424957285618184, 3), (9.424989105857355, 7), (9.425062539094958, 8), (9.425109858278296, 9), (9.42516370584547, 4), (9.42516
vote_cnt : [[3.0, 7], [0.0, 0], [0.0, 1], [0.0, 2], [0.0, 3], [0.0, 4], [0.0, 5], [0.0, 6], [0.0, 8], [0.0, 9]]

TestData 6 index( 2377 ) Computed class: 7 , True class: 7
162648, 3), (9.039255713541445, 3), (9.039261664164227, 5), (9.039384134447763, 7), (9.039474320465516, 5), (9.0395993486
vote_cnt : [[3.0, 1], [0.0, 0], [0.0, 2], [0.0, 3], [0.0, 4], [0.0, 5], [0.0, 6], [0.0, 7], [0.0, 8], [0.0, 9]]

TestData 7 index( 8219 ) Computed class: 1 , True class: 1
82762, 9), (8.486734846996027, 9), (8.486843566941204, 3), (8.48698674358626, 6), (8.487157068610976, 5), (8.4872413286
vote_cnt : [[3.0, 1], [0.0, 0], [0.0, 2], [0.0, 3], [0.0, 4], [0.0, 5], [0.0, 6], [0.0, 7], [0.0, 8], [0.0, 9]]

```

- hand craft features 학습 과정

```

TestData 5 index( 6253 ) Computed class: 7 , True class: 7
, 8), (47.07440918375928, 0), (47.085029467974216, 3), (47.085029467974216, 8), (47.085029467974216, 5), (47.085029467974216, 6)
vote_cnt : [[3.0, 0], [0.0, 1], [0.0, 2], [0.0, 3], [0.0, 4], [0.0, 5], [0.0, 6], [0.0, 7], [0.0, 8], [0.0, 9]]

TestData 8 index( 8320 ) Computed class: 2 , True class: 2
172789842982326, 7), (52.172789842982326, 4), (52.172789842982326, 3), (52.172789842982326, 6), (52.172789842982326, 7), (52.172789842982326, 8)
vote_cnt : [[3.0, 0], [0.0, 1], [0.0, 2], [0.0, 3], [0.0, 4], [0.0, 5], [0.0, 6], [0.0, 7], [0.0, 8], [0.0, 9]]

TestData 2 index( 8257 ) Computed class: 7 , True class: 7
1726, 9), (56.868268832451726, 3), (56.868268832451726, 7), (56.868268832451726, 4), (56.868268832451726, 8), (56.868268832451726, 5)
vote_cnt : [[5.0, 0], [0.0, 1], [0.0, 2], [0.0, 3], [0.0, 4], [0.0, 5], [0.0, 6], [0.0, 7], [0.0, 8], [0.0, 9]]

52.172789842982326, 4), (52.172789842982326, 3), (52.172789842982326, 6), (52.172789842982326, 7), (52.172789842982326, 8), (52.172789842982326, 9)
vote_cnt : [[10.0, 0], [0.0, 1], [0.0, 2], [0.0, 3], [0.0, 4], [0.0, 5], [0.0, 6], [0.0, 7], [0.0, 8], [0.0, 9]]
TestData 9 index( 9239 ) Computed class: 0 , True class: 0
Accuracy : 1.0
Features Compression, Weighted Majority vote / k = 10
TestData 0 index( 1069 ) Computed class: 3 , True class: 3
TestData 1 index( 25 ) Computed class: 0 , True class: 0
TestData 2 index( 8257 ) Computed class: 7 , True class: 7
TestData 3 index( 3608 ) Computed class: 0 , True class: 0
TestData 4 index( 1745 ) Computed class: 9 , True class: 9
TestData 5 index( 6253 ) Computed class: 7 , True class: 7
TestData 6 index( 7907 ) Computed class: 0 , True class: 0
TestData 7 index( 1825 ) Computed class: 9 , True class: 9
TestData 8 index( 8320 ) Computed class: 2 , True class: 2
TestData 9 index( 9239 ) Computed class: 0 , True class: 0

```


use all features와 hand craft features 두 경우 모두 disarr 리스트에는 성공적으로 거리가 정렬되었고 vote_cnt 리스트에 계산이 올바르게 들어가 결과를 성공적으로 책정했음을 알 수 있다. k=3 이외 k=5, k=10에 대해서도 올바르게 정렬되고 계산된 가중치가 들어가고 있음을 확인하였다.

use all features의 경우 feature가 784개 임에도 불구하고 distance(거리)의 값이 작게 나오는 이유는 data를 normalize하여 0~1사이의 실수 값이 데이터로 들어가게 되어 distance가 크지 않게 된다. 그러나 hand craft features의 경우 색칠된 공간 개수를 세는 것, 색칠된 공간 사이의 빈 공간 크기 0~28사이의 정수 값이 feature로 들어가게 되므로 distance 계산이 충분히 크게 나올 수 있다. 거리 측정결과인 disarr 리스트에 정렬되어 있는 값을 보면 확인 할 수 있다.

7. 결과 및 분석

<결과>

출력결과는 feature를 어떻게 사용했고, k값, majority vote인지 weighted majority vote 인지 제목으로 출력하였다. 이후 랜덤하게 선정된 각각의 테스트 데이터 100개에 대하여 프로그램이 계산한 class와 실제 class 답을 비교하였고 최종적으로는 정확도를 출력하였다. 먼저 모든 feature 784개를 사용한 경우부터 출력 하였으며 이후에 hand craft feature를 사용한 것을 출력하였다.

- Use All Feature

K=3 / Majority Vote / 정확도 : 96%

Use all Features Majority vote / k = 3			
TestData 0 index(4672) Computedclass: 0 , Trueclass: 0	TestData 32 index(7581) Computedclass: 1 , Trueclass: 1	TestData 68 index(1588) Computedclass: 3 , Trueclass: 3	
TestData 1 index(1136) Computedclass: 1 , Trueclass: 1	TestData 33 index(9286) Computedclass: 8 , Trueclass: 8	TestData 69 index(4902) Computedclass: 5 , Trueclass: 5	
TestData 2 index(8352) Computedclass: 1 , Trueclass: 1	TestData 34 index(5301) Computedclass: 0 , Trueclass: 0	TestData 70 index(4348) Computedclass: 0 , Trueclass: 0	
TestData 3 index(9874) Computedclass: 2 , Trueclass: 2	TestData 35 index(1016) Computedclass: 2 , Trueclass: 2	TestData 71 index(2832) Computedclass: 5 , Trueclass: 5	
TestData 4 index(1773) Computedclass: 1 , Trueclass: 1	TestData 36 index(3975) Computedclass: 3 , Trueclass: 3	TestData 72 index(5745) Computedclass: 7 , Trueclass: 7	
TestData 5 index(6681) Computedclass: 3 , Trueclass: 3	TestData 37 index(161) Computedclass: 6 , Trueclass: 6	TestData 73 index(1406) Computedclass: 5 , Trueclass: 5	
TestData 6 index(567) Computedclass: 0 , Trueclass: 0	TestData 38 index(9903) Computedclass: 1 , Trueclass: 1	TestData 74 index(6644) Computedclass: 1 , Trueclass: 1	
TestData 7 index(7640) Computedclass: 9 , Trueclass: 9	TestData 39 index(4759) Computedclass: 1 , Trueclass: 1	TestData 75 index(8627) Computedclass: 8 , Trueclass: 8	
TestData 8 index(8465) Computedclass: 7 , Trueclass: 7	TestData 40 index(4308) Computedclass: 8 , Trueclass: 8	TestData 76 index(6603) Computedclass: 8 , Trueclass: 8	
TestData 9 index(9335) Computedclass: 1 , Trueclass: 1	TestData 41 index(6220) Computedclass: 4 , Trueclass: 4	TestData 77 index(2511) Computedclass: 3 , Trueclass: 3	
TestData 10 index(2031) Computedclass: 3 , Trueclass: 3	TestData 42 index(6471) Computedclass: 7 , Trueclass: 7	TestData 78 index(9806) Computedclass: 7 , Trueclass: 7	
TestData 11 index(5529) Computedclass: 0 , Trueclass: 0	TestData 43 index(6752) Computedclass: 0 , Trueclass: 0	TestData 79 index(5355) Computedclass: 4 , Trueclass: 4	
TestData 12 index(371) Computedclass: 2 , Trueclass: 2	TestData 44 index(5577) Computedclass: 4 , Trueclass: 4	TestData 80 index(6795) Computedclass: 0 , Trueclass: 0	
TestData 13 index(6912) Computedclass: 2 , Trueclass: 2	TestData 45 index(8037) Computedclass: 8 , Trueclass: 8	TestData 81 index(7488) Computedclass: 4 , Trueclass: 4	
TestData 14 index(5655) Computedclass: 2 , Trueclass: 2	TestData 46 index(4737) Computedclass: 6 , Trueclass: 8	TestData 82 index(2483) Computedclass: 6 , Trueclass: 6	
TestData 15 index(8579) Computedclass: 6 , Trueclass: 6	TestData 47 index(6925) Computedclass: 0 , Trueclass: 0	TestData 83 index(3811) Computedclass: 3 , Trueclass: 2	
TestData 16 index(979) Computedclass: 1 , Trueclass: 1	TestData 48 index(2008) Computedclass: 3 , Trueclass: 3	TestData 84 index(3132) Computedclass: 1 , Trueclass: 1	
TestData 17 index(8904) Computedclass: 7 , Trueclass: 7	TestData 49 index(1800) Computedclass: 6 , Trueclass: 6	TestData 85 index(1885) Computedclass: 1 , Trueclass: 1	
TestData 18 index(7284) Computedclass: 5 , Trueclass: 5	TestData 50 index(9672) Computedclass: 4 , Trueclass: 4	TestData 86 index(3038) Computedclass: 7 , Trueclass: 7	
TestData 19 index(2624) Computedclass: 3 , Trueclass: 3	TestData 51 index(9583) Computedclass: 5 , Trueclass: 5	TestData 87 index(4670) Computedclass: 1 , Trueclass: 1	
TestData 20 index(3770) Computedclass: 4 , Trueclass: 4	TestData 52 index(4041) Computedclass: 8 , Trueclass: 8	TestData 88 index(3309) Computedclass: 7 , Trueclass: 7	
TestData 21 index(586) Computedclass: 0 , Trueclass: 0	TestData 53 index(1921) Computedclass: 6 , Trueclass: 6	TestData 89 index(3133) Computedclass: 4 , Trueclass: 4	
TestData 22 index(6800) Computedclass: 2 , Trueclass: 2	TestData 54 index(7695) Computedclass: 7 , Trueclass: 7	TestData 90 index(775) Computedclass: 2 , Trueclass: 2	
TestData 23 index(1463) Computedclass: 3 , Trueclass: 3	TestData 55 index(3533) Computedclass: 4 , Trueclass: 4	TestData 91 index(2738) Computedclass: 6 , Trueclass: 6	
TestData 24 index(6302) Computedclass: 9 , Trueclass: 9	TestData 56 index(7269) Computedclass: 0 , Trueclass: 0	TestData 92 index(7819) Computedclass: 5 , Trueclass: 5	
TestData 25 index(1456) Computedclass: 6 , Trueclass: 6	TestData 57 index(560) Computedclass: 9 , Trueclass: 9	TestData 93 index(6666) Computedclass: 7 , Trueclass: 7	
TestData 26 index(2745) Computedclass: 3 , Trueclass: 3	TestData 58 index(8852) Computedclass: 6 , Trueclass: 6	TestData 94 index(9101) Computedclass: 7 , Trueclass: 7	
TestData 27 index(8443) Computedclass: 1 , Trueclass: 1	TestData 59 index(6751) Computedclass: 1 , Trueclass: 1	TestData 95 index(8348) Computedclass: 5 , Trueclass: 5	
TestData 28 index(3065) Computedclass: 8 , Trueclass: 8	TestData 60 index(3660) Computedclass: 0 , Trueclass: 0	TestData 96 index(5856) Computedclass: 9 , Trueclass: 9	
TestData 29 index(3616) Computedclass: 7 , Trueclass: 7	TestData 61 index(1122) Computedclass: 7 , Trueclass: 7	TestData 97 index(2466) Computedclass: 6 , Trueclass: 6	
TestData 30 index(6924) Computedclass: 4 , Trueclass: 4	TestData 62 index(2058) Computedclass: 2 , Trueclass: 2	TestData 98 index(1111) Computedclass: 4 , Trueclass: 4	
TestData 31 index(3921) Computedclass: 2 , Trueclass: 2	TestData 63 index(5562) Computedclass: 2 , Trueclass: 2	TestData 99 index(9810) Computedclass: 1 , Trueclass: 1	
	TestData 64 index(3922) Computedclass: 1 , Trueclass: 1	Accuracy : 0.96	

K=10 / Weighted Majority Vote / 정확도 : 97%

```
Use all Features, Weighted Majority vote / k = 10
TestData 0 index( 4672 ) Computedclass: 0 , Trueclass: 0
TestData 1 index( 1136 ) Computedclass: 1 , Trueclass: 1
TestData 2 index( 8352 ) Computedclass: 1 , Trueclass: 1
TestData 3 index( 9874 ) Computedclass: 2 , Trueclass: 2
TestData 4 index( 1773 ) Computedclass: 1 , Trueclass: 1
TestData 5 index( 6681 ) Computedclass: 3 , Trueclass: 3
TestData 6 index( 567 ) Computedclass: 0 , Trueclass: 0
TestData 7 index( 7640 ) Computedclass: 9 , Trueclass: 9
TestData 8 index( 8465 ) Computedclass: 7 , Trueclass: 7
TestData 9 index( 9335 ) Computedclass: 1 , Trueclass: 1
TestData 10 index( 2031 ) Computedclass: 3 , Trueclass: 3
TestData 11 index( 5529 ) Computedclass: 0 , Trueclass: 0
TestData 12 index( 371 ) Computedclass: 2 , Trueclass: 2
TestData 13 index( 6912 ) Computedclass: 2 , Trueclass: 2
TestData 14 index( 5655 ) Computedclass: 7 , Trueclass: 7
TestData 15 index( 8579 ) Computedclass: 6 , Trueclass: 6
TestData 16 index( 979 ) Computedclass: 1 , Trueclass: 1
TestData 17 index( 8904 ) Computedclass: 7 , Trueclass: 7
TestData 18 index( 7284 ) Computedclass: 5 , Trueclass: 5
TestData 19 index( 2624 ) Computedclass: 3 , Trueclass: 3
TestData 20 index( 3770 ) Computedclass: 4 , Trueclass: 4
TestData 21 index( 586 ) Computedclass: 0 , Trueclass: 0
TestData 22 index( 6800 ) Computedclass: 2 , Trueclass: 2
TestData 23 index( 1463 ) Computedclass: 3 , Trueclass: 3
TestData 24 index( 6302 ) Computedclass: 9 , Trueclass: 9
TestData 25 index( 1456 ) Computedclass: 6 , Trueclass: 6
TestData 26 index( 2745 ) Computedclass: 3 , Trueclass: 3
TestData 27 index( 8443 ) Computedclass: 1 , Trueclass: 1
TestData 28 index( 3065 ) Computedclass: 8 , Trueclass: 8
TestData 29 index( 3616 ) Computedclass: 7 , Trueclass: 7
TestData 30 index( 6924 ) Computedclass: 4 , Trueclass: 4
TestData 31 index( 3921 ) Computedclass: 2 , Trueclass: 2
TestData 32 index( 5301 ) Computedclass: 0 , Trueclass: 0
TestData 33 index( 1016 ) Computedclass: 2 , Trueclass: 2
TestData 34 index( 3975 ) Computedclass: 3 , Trueclass: 3
TestData 35 index( 161 ) Computedclass: 6 , Trueclass: 6
TestData 36 index( 9903 ) Computedclass: 1 , Trueclass: 1
TestData 37 index( 4759 ) Computedclass: 1 , Trueclass: 1
TestData 38 index( 4380 ) Computedclass: 8 , Trueclass: 8
TestData 39 index( 6220 ) Computedclass: 4 , Trueclass: 4
TestData 40 index( 6471 ) Computedclass: 7 , Trueclass: 7
TestData 41 index( 6752 ) Computedclass: 0 , Trueclass: 0
TestData 42 index( 5577 ) Computedclass: 4 , Trueclass: 4
TestData 43 index( 8037 ) Computedclass: 8 , Trueclass: 8
TestData 44 index( 4737 ) Computedclass: 6 , Trueclass: 6
TestData 45 index( 6925 ) Computedclass: 0 , Trueclass: 0
TestData 46 index( 2008 ) Computedclass: 3 , Trueclass: 3
TestData 47 index( 1800 ) Computedclass: 6 , Trueclass: 6
TestData 48 index( 9672 ) Computedclass: 4 , Trueclass: 4
TestData 49 index( 9583 ) Computedclass: 5 , Trueclass: 5
TestData 50 index( 4041 ) Computedclass: 8 , Trueclass: 8
TestData 51 index( 1921 ) Computedclass: 6 , Trueclass: 6
TestData 52 index( 7695 ) Computedclass: 7 , Trueclass: 7
TestData 53 index( 3533 ) Computedclass: 4 , Trueclass: 4
TestData 54 index( 7269 ) Computedclass: 0 , Trueclass: 0
TestData 55 index( 560 ) Computedclass: 9 , Trueclass: 9
TestData 56 index( 8852 ) Computedclass: 6 , Trueclass: 6
TestData 57 index( 6751 ) Computedclass: 1 , Trueclass: 1
TestData 58 index( 3660 ) Computedclass: 0 , Trueclass: 0
TestData 59 index( 1122 ) Computedclass: 7 , Trueclass: 7
TestData 60 index( 2058 ) Computedclass: 2 , Trueclass: 2
TestData 61 index( 5562 ) Computedclass: 2 , Trueclass: 2
TestData 62 index( 3922 ) Computedclass: 1 , Trueclass: 1
TestData 63 index( 2896 ) Computedclass: 0 , Trueclass: 0
TestData 64 index( 4800 ) Computedclass: 7 , Trueclass: 7
TestData 65 index( 1500 ) Computedclass: 3 , Trueclass: 3
TestData 66 index( 4902 ) Computedclass: 5 , Trueclass: 5
TestData 67 index( 4348 ) Computedclass: 0 , Trueclass: 0
TestData 68 index( 2832 ) Computedclass: 5 , Trueclass: 5
TestData 69 index( 5745 ) Computedclass: 7 , Trueclass: 7
TestData 70 index( 1406 ) Computedclass: 5 , Trueclass: 5
TestData 71 index( 6644 ) Computedclass: 1 , Trueclass: 1
TestData 72 index( 8627 ) Computedclass: 8 , Trueclass: 8
TestData 73 index( 6603 ) Computedclass: 8 , Trueclass: 8
TestData 74 index( 2511 ) Computedclass: 3 , Trueclass: 3
TestData 75 index( 9806 ) Computedclass: 7 , Trueclass: 7
TestData 76 index( 5355 ) Computedclass: 4 , Trueclass: 4
TestData 77 index( 6795 ) Computedclass: 0 , Trueclass: 0
TestData 78 index( 7488 ) Computedclass: 4 , Trueclass: 4
TestData 79 index( 2483 ) Computedclass: 6 , Trueclass: 6
TestData 80 index( 3811 ) Computedclass: 3 , Trueclass: 2
TestData 81 index( 3132 ) Computedclass: 1 , Trueclass: 1
TestData 82 index( 1885 ) Computedclass: 1 , Trueclass: 1
TestData 83 index( 3038 ) Computedclass: 7 , Trueclass: 7
TestData 84 index( 4670 ) Computedclass: 1 , Trueclass: 1
TestData 85 index( 3309 ) Computedclass: 7 , Trueclass: 7
TestData 86 index( 3133 ) Computedclass: 4 , Trueclass: 4
TestData 87 index( 775 ) Computedclass: 2 , Trueclass: 2
TestData 88 index( 2738 ) Computedclass: 6 , Trueclass: 6
TestData 89 index( 7819 ) Computedclass: 5 , Trueclass: 5
TestData 90 index( 6666 ) Computedclass: 7 , Trueclass: 7
TestData 91 index( 9101 ) Computedclass: 7 , Trueclass: 7
TestData 92 index( 8348 ) Computedclass: 5 , Trueclass: 5
TestData 93 index( 5856 ) Computedclass: 9 , Trueclass: 9
TestData 94 index( 2466 ) Computedclass: 6 , Trueclass: 6
TestData 95 index( 1111 ) Computedclass: 4 , Trueclass: 4
TestData 96 index( 9810 ) Computedclass: 1 , Trueclass: 1
Accuracy : 0.97
```


K=5 / Majority Vote / 정확도 : 93%

[illegible]

K=5 / Weighted Majority Vote / 정확도 : 93%

Features	Compression, Weighted Majority vote / k = 5	TestData 34 index(4)	Computed class: 4 , True class: 4	TestData 68 index(3955)	Computed class: 3 , True class: 3
TestData 0 index(2211)	Computed class: 8 , True class: 8	TestData 35 index(9218)	Computed class: 9 , True class: 9	TestData 69 index(7100)	Computed class: 2 , True class: 2
TestData 1 index(648)	Computed class: 1 , True class: 1	TestData 36 index(1809)	Computed class: 7 , True class: 7	TestData 70 index(6980)	Computed class: 9 , True class: 9
TestData 2 index(6696)	Computed class: 4 , True class: 4	TestData 37 index(2355)	Computed class: 1 , True class: 1	TestData 71 index(6975)	Computed class: 3 , True class: 3
TestData 3 index(8809)	Computed class: 1 , True class: 1	TestData 38 index(4774)	Computed class: 1 , True class: 1	TestData 72 index(7460)	Computed class: 7 , True class: 7
TestData 4 index(8356)	Computed class: 7 , True class: 7	TestData 39 index(1008)	Computed class: 1 , True class: 1	TestData 73 index(2824)	Computed class: 0 , True class: 0
TestData 5 index(7947)	Computed class: 4 , True class: 4	TestData 40 index(922)	Computed class: 2 , True class: 2	TestData 74 index(1187)	Computed class: 2 , True class: 2
TestData 6 index(7585)	Computed class: 6 , True class: 6	TestData 41 index(8364)	Computed class: 7 , True class: 7	TestData 75 index(5662)	Computed class: 3 , True class: 5
TestData 7 index(4662)	Computed class: 9 , True class: 9	TestData 42 index(6426)	Computed class: 0 , True class: 0	TestData 76 index(3103)	Computed class: 7 , True class: 7
TestData 8 index(91)	Computed class: 6 , True class: 6	TestData 43 index(1576)	Computed class: 7 , True class: 7	TestData 77 index(3917)	Computed class: 5 , True class: 5
TestData 9 index(9741)	Computed class: 7 , True class: 9	TestData 44 index(3681)	Computed class: 2 , True class: 2	TestData 78 index(2393)	Computed class: 8 , True class: 8
TestData 10 index(4274)	Computed class: 2 , True class: 2	TestData 45 index(2234)	Computed class: 7 , True class: 7	TestData 79 index(7059)	Computed class: 7 , True class: 7
TestData 11 index(6862)	Computed class: 3 , True class: 3	TestData 46 index(4775)	Computed class: 2 , True class: 2	TestData 80 index(1198)	Computed class: 8 , True class: 8
TestData 12 index(1868)	Computed class: 1 , True class: 1	TestData 47 index(1405)	Computed class: 5 , True class: 5	TestData 81 index(2878)	Computed class: 1 , True class: 1
TestData 13 index(3895)	Computed class: 0 , True class: 0	TestData 48 index(6522)	Computed class: 5 , True class: 5	TestData 82 index(4421)	Computed class: 4 , True class: 4
TestData 14 index(7468)	Computed class: 1 , True class: 1	TestData 49 index(752)	Computed class: 9 , True class: 4	TestData 83 index(8229)	Computed class: 1 , True class: 1
TestData 15 index(5040)	Computed class: 7 , True class: 7	TestData 50 index(4397)	Computed class: 3 , True class: 3	TestData 84 index(5493)	Computed class: 8 , True class: 8
TestData 16 index(3573)	Computed class: 5 , True class: 7	TestData 51 index(8106)	Computed class: 7 , True class: 7	TestData 85 index(1750)	Computed class: 7 , True class: 7
TestData 17 index(9894)	Computed class: 7 , True class: 7	TestData 52 index(333)	Computed class: 8 , True class: 5	TestData 86 index(3480)	Computed class: 1 , True class: 1
TestData 18 index(7303)	Computed class: 1 , True class: 1	TestData 53 index(9141)	Computed class: 7 , True class: 7	TestData 87 index(3579)	Computed class: 3 , True class: 3
TestData 19 index(710)	Computed class: 5 , True class: 5	TestData 54 index(1053)	Computed class: 2 , True class: 2	TestData 88 index(1177)	Computed class: 2 , True class: 2
TestData 20 index(1870)	Computed class: 0 , True class: 0	TestData 55 index(6321)	Computed class: 2 , True class: 2	TestData 89 index(9731)	Computed class: 7 , True class: 7
TestData 21 index(5237)	Computed class: 3 , True class: 3	TestData 56 index(6373)	Computed class: 4 , True class: 4	TestData 90 index(9494)	Computed class: 6 , True class: 6
TestData 22 index(3029)	Computed class: 4 , True class: 4	TestData 57 index(4267)	Computed class: 1 , True class: 1	TestData 91 index(632)	Computed class: 6 , True class: 2
TestData 23 index(4908)	Computed class: 6 , True class: 6	TestData 58 index(5865)	Computed class: 0 , True class: 0	TestData 92 index(291)	Computed class: 2 , True class: 2
TestData 24 index(3251)	Computed class: 0 , True class: 0	TestData 59 index(4269)	Computed class: 4 , True class: 4	TestData 93 index(9274)	Computed class: 1 , True class: 1
TestData 25 index(5376)	Computed class: 8 , True class: 8	TestData 60 index(4473)	Computed class: 0 , True class: 0	TestData 94 index(8648)	Computed class: 2 , True class: 2
TestData 26 index(5438)	Computed class: 8 , True class: 8	TestData 61 index(5936)	Computed class: 4 , True class: 4	TestData 95 index(749)	Computed class: 1 , True class: 1
TestData 27 index(3182)	Computed class: 6 , True class: 6	TestData 62 index(9153)	Computed class: 9 , True class: 9	TestData 96 index(6740)	Computed class: 0 , True class: 9
TestData 28 index(8327)	Computed class: 5 , True class: 5	TestData 63 index(9020)	Computed class: 8 , True class: 8	TestData 97 index(5816)	Computed class: 4 , True class: 4
TestData 29 index(8141)	Computed class: 4 , True class: 4	TestData 64 index(6917)	Computed class: 1 , True class: 1	TestData 98 index(6653)	Computed class: 7 , True class: 7
TestData 30 index(3724)	Computed class: 0 , True class: 0	TestData 65 index(8205)	Computed class: 4 , True class: 4	TestData 99 index(6248)	Computed class: 8 , True class: 8
TestData 31 index(7366)	Computed class: 4 , True class: 4			Accuracy : 0.93	

<결과 분석>

전체적으로 결과는 Majority vote와 Weighted Majority vote 동일하게 나왔으며 K 값에 따라서도 결과가 동일하게 나왔는데 이것은 랜덤한 테스트 데이터이지만 테스트를 모두 동일한 테스트 데이터 셋을 가지고 평가를 했기 때문으로 볼 수 있다. 그 이유로는 Compute가 틀린 이미지 파일이 동일하기 때문인데 이것은 프로그램이 해당 이미지 자체에 대해서 Compute를 하지 못한다는 것을 보여준다. 모든 features들을 사용한 경우와 hand craft feature를 사용한 경우에 대해서 각기 보고 실제 어떤 이미지를 틀린 것인지 관찰하였다.

- Use All Feature

784개의 모든 feature를 이용한 결과 K = 3 majority vote에서는 96%의 정확도를 보여주었고 나머지 5가지 경우는 97%의 정확도가 나온 것을 관찰할 수 있었다. 784개의 모든 feature를 사용하더라도 100%의 정확도가 나오지 않는다는 것은 단순히 칠해진 공간의 진하기로 distance를 구하는 것이 숫자의 모양을 완벽하게 표현하지 않는다는 것을 알 수 있다. 그렇다면 어떤 모양의 숫자를 틀리게 상정했는지 이미지 출력을 통해 직접 살펴보자.



index : 2896, computed : 0 / index : 6740, computed : 0 / index : 333, computed : 8

위 이미지를 보다시피 숫자는 알아볼 수 있으나 모양이 뒤틀어져 있거나 어느 쪽이 과하게 크다는 것을 알 수 있다. 784개의 모든 feature를 사용했음에도 불구하고 이런 이미지를 계산해내지 못하는 것은 유동적인 숫자의 모습을 feature로 사용한 것이 아니라 고정된 자리의 값을 feature로 사용하기 때문이다. 해당 이미지 데이터는 위치가 center로 정렬된 데이터이지만 숫자 모양이 뒤틀어지거나 모양이 이상한 부분에 대해서는 위치가 feature로서 재기능을 하지 못한다는 것을 알 수 있다.

- hand craft feature

112개의 압축된 hand craft feature를 이용한 결과 모든 K값, 그리고 KNN 알고리즘에 대해 정확도가 93%로 동일하게 나왔으며 동일한 이미지에 대해서 계산이 틀렸다. 우선 내가 만든 hand craft feature에서 56개는 위치를 기반으로 카운트한 feature이고 두번째는 위치 기반이기는 하지만 위치를 기준으로 상대 값을 구해 숫자안의 검은색 공백 크기를 구한 feature였다. 하지만 위치 기반 feature만을 사용하였을 때 보다 공백 크기를 구한 feature를 추가하였을 때 정확도가 더 떨어졌다. 이것은 내가 추가한 공백에 대한 feature가 KNN을 이용한 이미지 추론에 있어서 크게 도움이 되지 않는다는 것을 말한다. KNN에서는 Linear regression이나 Logistic regression과 다르게

반복학습을 통한 가중치의 값의 변화가 있거나 학습결과가 다르게 나오지 않는다. 물론 Train 데이터 셋에 의해 즉, 학습 데이터가 다를 경우 다른 결과 값을 내놓겠지만 실제 기계가 반복적으로 학습하는 과정은 없다는 것이다. 그래서 실제로 112개의 feature에 대해서도 분명 어떤 feature는 더 높은 가중치를 가져야 했을 것이고 어떤 feature는 약한 가중치를 가져야 했을 것이다. 그러나 이번 과제 of 알고리즘은 KNN이고 가중치를 주더라도 사람이 주관적으로 가중치를 주고 시작해야 하므로 결과가 잘 나오는지 예측할 수는 없다. 틀린 이미지 7개를 보도록 하자



idx : 4274, computed : 3 / idx : 3573, computed : 5 / idx : 752, computed : 9 / idx : 333, computed : 8



index : 5562, computed : 3 / index : 632, computed : 6 / index : 6740, computed : 0

숫자의 모양이 일반적이지 않은 경우에 대해서도 틀렸지만 일반적인 모양의 숫자에 대해서도 계산이 틀렸음을 알 수 있다.

사실 hand craft feature를 만드는데 있어서 시간이 충분하지 않고, 결과에 대해 예측할 수 없는 이유는 **hand craft feature가 heuristics 한 접근 방식이기 때문이다**. hand craft feature의 목표는 단순히 feature수를 줄이는 것이 아니라 정확도를 올리는 것이기 때문에 어떤 feature를 이용하고 이것의 가중치는 어떻게 상정해야 할지 모두 사람이 정해야 하기 때문에 알고리즘을 테스트하는데 시간이 오래 걸린다. 물론 결과를 장담할 수도 없다.

feature수를 줄이는 것이 목표였다면 hand craft feature를 사용하기 보다는 **PCA**를 이용해 차원 압축(feature 수를 줄임)을 하는 것이 정확하고 수학적 방법일 것이다.