인공지능 과제 리포트

과제 제목: KNN MNIST Classification

학번: B511086
이름: 백성진

1. 과제 개요

이번 과제에서는 이전 과제였던 KNN 알고리즘을 사용한 IRIS 분류에서와 같이, KNN 알고리즘을 사용하여 MNIST 손글씨 데이터셋을 분류하는 과제입니다.
총 6만개의 훈련용 데이터와 1만개의 테스트 데이터가 있으며, 6만개의 훈련용 데이터를 사용하여, 1만개중 랜덤하게 선택된 데이터셋을 분류하고 이때 정확도가 어느정도인지 확인하였습니다. 과제의 핵심은 데이터를 변형하여 적절한 hand-crafted features를 생성하고 분류에 사용하는 것입니다.

2. 구현 환경

OS: Windows 10
IDE: Pycharm
Python version: Python 3.8.2
RAM: 16GB
CPU: Intel i5-8500 3.00Ghz
GPU: GeForce GTX 1050

3. 알고리즘에 대한 설명

이번 과제에서 사용한 주요 알고리즘은 크게 2가지입니다.

첫 번째로는, KNN 알고리즘입니다. KNN 알고리즘은 이전 과제에서 다루었으며 변수이름 변경 이외에는 특별한 추가적인 기능 구현이 없었습니다. 다만, 데이터의 feature간 거리를 계산할 때 생기는 추가적인 제곱 및 제곱근 연산을 막아 속도 향상을 꾀하기 위해 이번 과제에서는 Euclidean Distance 가 아닌 Manhattan distance를 사용하였습니다.
또한, 일반 majority vote 보다 형평성 있는 결과를 나타내기 위해 가중치가 고려된 weighted majority vote 방식으로 분류를 하도록 하였습니다.
이때, k 값으로는 이전 IRIS Classification에서 가장 정확한 값을 나타내었던 10을 사용했습니다.

두 번째로는, data hand-craft 알고리즘입니다.
거리 계산을 위해서는, 원래 데이터 상에서는 28*28개, 즉 원소에 784번 접근해야 했습니다. 이를 줄이기 위해, 평균값 필터 알고리즘을 적용하였습니다. 평균값 필터 알고리즘이란, 이미지의 각 픽셀에서 일정한 범위의 픽셀값의 평균치를 구하여 현재 픽셀 값을 그것으로 대체하는 알고리즘입니다. 이번 과제의 특성상, 평균 픽셀값은 사람이 눈으로 확인하기 위해 위화감 없는 이미지를 생산하는 과정이 아닌 계산 데이터의 수 자체를 줄이는 데에 그 목적이 있으므로 상하좌우 n개의 픽셀이 아닌 우측 n개의 픽셀(리스트의 원소)을 범위로 잡았습니다. 단, 원래 이미지(28*28)과 최대한 맞추고자 하여(즉, matrix에서 n번째 행의 27번째 원소가 n+1

번째 행의 0번째 원소에 영향을 끼치지 못하도록 하기 위해) 28의 약수인 4와 7 두 수중 하나를 n의 범위로 선택하였습니다.
따라서, 입력 데이터의 feature 수는 n이 4일 경우 784/4, 7일 경우 784/7개입니다.

4. 데이터에 대한 설명

4.1 Input Feature

MNIST 데이터 셋의 input feature의 차수는 28*28입니다. 데이터 셋을 받아 올 때 데이터의 자료형을 0~1 사이의 실수와, 0~255 사이의 정수로 지정할 수 있었습니다. 이번 과제에서는 0~255 사이의 정수로 input feature를 설정하였습니다. 그 이유는, 제곱 연산이 들어가지 않는 Manhattan distance를 사용한 거리계산에서 제곱근 계산이 아닌 계산을 하여 속도 향상을 꾀하기 위함입니다.

각 28*28의 features에는 0~255 사이의 값이 들어있습니다. 이는 픽셀에서의 수치를 의미합니다. 0에 근접할수록 어두운 색, 255에 근접할수록 밝은 색을 의미합니다.

4.2 Target Output

output class는 0부터 9까지의 정수입니다. 각 MNIST이미지당 의미하는 숫자가 1개씩 지정되어있습니다. KNN알고리즘을 통해 이미지가 어느 숫자에 속하는지, 즉 어느 숫자일 확률이 가장 높은지를 계산하게 됩니다.

5. 소스코드에 대한 설명

- 본 과제에 사용된 hand crafting method입니다.

hand_craft_data

def hand_craft_data(li):
    craftedlist_4 = []
    craftedlist_7 = []
 	
    for i in range(len(li)):
        tmpli = li[i]
        crafted_4 = []
        crafted_7 = []
        sum_4 = 0
        sum_7 = 0

```
        cnt_4 = 0
        cnt_7 = 0

        for it in range(len(tmpli)):
            cnt_4 += 1
            cnt_7 += 1
            # 4개 픽셀의 평균
            if cnt_4 % 4 == 0:
                crafted_4.append(sum_4 / 4)
                sum_4 = 0
            # 7개 픽셀의 평균
            if cnt_7 % 7 == 0:
                crafted_7.append(sum_7 / 7)
                sum_7 = 0
            sum_4 += tmpli[it]
            sum_7 += tmpli[it]

        craftedlist_4.append(crafted_4)
        craftedlist_7.append(crafted_7)

    return craftedlist_4, craftedlist_7
```

본 함수는 mnist 데이터 셋을 hand crafting 하는 함수입니다.
이번 과제에 사용한 hand crafting 방법은, 본래 데이터 중 모든 행에서, 4개, 7개의 데이터를 1개의 데이터로 각각 묶은 후, 묶인 1개의 데이터에 4개, 7개의 평균값을 저장하는 방식을 채택했습니다.
결과적으로 28*28의 원래 데이터셋에서, 4개로 묶었을 경우엔 7*28, 7개로 묶었을 경우엔 4*28의 features를 가지도록 하였습니다.
본 함수는 두 개의 리스트(4개 픽셀로 묶인 리스트, 7개 픽셀로 묶인 리스트)를 반환합니다.

6. 학습 과정에 대한 설명

본 과제에서는 6만개의 훈련용 데이터를 분류하는 것 뿐만 아니라, 시간을 단축시키고 정확도도 일정 수준까지 가지도록 훈련용 데이터를 hand-crafting 하는 것이 목적입니다.
따라서, hand_craft_data()를 구현 후 knn 분류 시작 전 데이터 셋을 임의로 조정하는 작업을 거치도록 하였습니다.
임의로 조작한 데이터를 분류할 때, input features가 산술적으로 줄어든 만큼 데이터 클래스 분류 작업에 걸리는 시간 또한 감소할 것으로 기대하였습니다.

7. 결과 및 분석

본 과제에서는 hand crafting이 완료 된 데이터 셋과 원본 데이터 셋을 분류하는데 걸리는 시간과 정확도를 분석하여 성능좋은 hand crafting 방법을 찾는 것이 목적입니다.

분석을 위해 테스트 케이스는 10, 100, 500, 1000을, 그리고 대조군을 위해 4개 원소씩 압축한 경우, 7개 원소씩 압축한 경우, 그리고 압축을 실시하지 않은 경우의 데이터를 사용하였습니다.

knn 알고리즘에서 사용되는 k의 값은, IRIS 분류에서 좋은 결과를 거두었던 10을 선택하였으며 features 거리 계산에는 계산 부하를 줄이기 위해 Manhattan Distance를 사용하였습니다. 테스트 데이터 선정을 위해 numpy의 랜덤함수를 사용하였으며, 각 테스트 횟수 간에는 동일한 테스트 데이터 세트를 사용하였습니다.

테스트 경과시간을 구할 때에는, 입력 데이터를 조작하는 시간은 포함하지 않았습니다. 보통의 경우 프로그램 런 타임으로 crafting을 하는 것이 아닌, 미리 crafting한 데이터를 파일에 저장해 두기 때문 입니다.

1. 테스트 케이스 개수 : 10

```
Test size : 10
None-crafted accuracy : 0.6     correct : 6     miss : 4    elapsed time(sec) : 386.67146490000005
4-averaged crafted accuracy : 1.0   correct : 10    miss : 0    elapsed time(sec) : 45.765648799999994
7-averaged crafted accuracy : 1.0   correct : 10    miss : 0    elapsed time(sec) : 30.331820800000003
```

2. 테스트 케이스 개수 : 100

```
Test size : 100
None-crafted accuracy : 0.79    correct : 79    miss : 21   elapsed time(sec) : 3651.8035016999997
4-averaged crafted accuracy : 0.96  correct : 96    miss : 4    elapsed time(sec) : 578.2355664
7-averaged crafted accuracy : 0.94  correct : 94    miss : 6    elapsed time(sec) : 344.4817221999999
```

3. 테스트 케이스 개수 : 500

```
Test size : 500
None-crafted accuracy : 0.782   correct : 391   miss : 109  elapsed time(sec) : 18048.062937900002
4-averaged crafted accuracy : 0.952     correct : 476   miss : 24   elapsed time(sec) : 2855.466554
7-averaged crafted accuracy : 0.928     correct : 464   miss : 36   elapsed time(sec) : 1711.6331457000001
```

4. 테스트 케이스 개수 : 1000

```
Test size : 1000
None-crafted accuracy : 0.816   correct : 816   miss : 184  elapsed time(sec) : 34970.272689
4-averaged crafted accuracy : 0.953     correct : 953   miss : 47   elapsed time(sec) : 5545.653004700001
7-averaged crafted accuracy : 0.939     correct : 939   miss : 61   elapsed time(sec) : 3321.3002049000006
```

위 결과들로 미루어보았을 때, 조작하지 않은 데이터를 사용하였을 때에 비해 4개를 묶은 후 평균을 내는 방식으로 hand-crafting 한 경우에는 약 10배의 속도를 가졌으며, 7개를 묶은 후 평균을 내는 방식은 약 6배의 시간 단축 효율을 가졌습니다.

정확도 또한 데이터 조작을 하지 않은 경우보다, 조작한 경우가 더 높았습니다.
4개를 묶은 후 평균을 내는 방식으로 hand-crafting 한 경우에는 약 1.2배의 더 높은 정확도를 가졌으며, 7개를 묶은 후 평균을 내는 방식은 약 1.17배의 시간 단축 효율을 가졌습니다.
 더 높은 정확도를 가졌습니다.

따라서, (효율 = 정확도 비/속도 비) 수식을 바탕으로, 7개를 묶은 후 평균을 내어 하나의 픽셀 값으로 지정하는 방식이 hand-crafting이 4개를 묶은 후 평균을 내어 하나의 픽셀 값으로 지정하는 방식에 비해 1.625 배(= 0.195/0.12) 더 높은 효율을 가지는 것으로 분석하였습니다.