8.3

1. 문제 요약

다른 예제(ex. 2_1_knn_handwritten_digits_recognition_varying_k.py)에서 사용했던 get_accuracy() 함수가 여기서(1_1_checking_knn_model_accuracy.py)는 계속 50% 대의 정확도로 고정되어 출력된다.
어떤 오류가 숨겨져 있는지 원인을 파악하여 해결해보자.

2. 결론

`get_accuracy` 함수 내부의 데이터 값 비교 과정을 살펴보면 `np.squeeze(predictions)`과 `labels`를 비교하여 정확도를 계산하는데, 이 두 데이터의 차원과 형태가 일치하지 않아 정확도에 차이를 초래한다. 정확한 비교를 위해 데이터의 shape를 일치시키는 과정이 필요하며, 이를 통해 정확도를 높일 수 있다.

아래와 같이 코드를 수정하면 원하는 높은 정확도를 확인할 수 있다.

```
# 코드 수정 방법 1
def get_accuracy(predictions, labels):
    accuracy = (np.squeeze(predictions) == np.squeeze(labels)).mean()
    return accuracy * 100
```

```
# 코드 수정 방법 2
def get_accuracy(predictions, labels):
    accuracy = (predictions == labels).mean()
    return accuracy * 100
```

3. get_accuracy()의 로직

```
def get_accuracy(predictions, labels):
    accuracy = (np.squeeze(predictions) == labels).mean()
    return accuracy * 100
```

이 함수는 예측 결과인 predictions와 실제 레이블인 labels를 비교하여 정확도를 계산한다.

함수의 주요 로직:

1. **np.squeeze(predictions)**: np.squeeze(predictions)를 사용하여 예측 결과를 1차원 배열로 변환한다. 이는 예측 결과가 2차원 배열이거나 다차원 배열인 경우에도 처리할 수 있도록 한다.

> 💡 np.squeeze는 NumPy에서 사용되는 함수 중 하나. 이 함수는 배열에서 크기가 1인 차원을 제거하여 배열의 모양을 변경.

2. **(np.squeeze(predictions) == labels)**: 변환된 예측 결과와 실제 레이블 labels를 비교하여 같은지 여부를 나타내는 불리언 배열을 생성한다. 이 때, == 연산자를 사용하여 각 요소를 비교해 예측값과 레이블이 일치하는 경우 True를, 그렇지 않은 경우 False를 반환한다.
3. **.mean()**: mean() 함수를 사용하여 불리언 배열의 평균을 계산한다. 이는 True와 False의 비율을 계산하여 정확도를 구하는 것이다. True는 1로 간주되고 False는 0으로 간주되므로, True의 비율은 예측 결과가 실제 레이블과 일치하는 비율, 즉 정확도를 나타낸다.
4. **return accuracy * 100**: 정확도를 백분율로 표시하기 위해 100을 곱하여 반환한다.

위의 과정에서 가장 중요한 부분:

예측값(predictions)과 실제 레이블(labels)을 비교하는 부분이다. 이 부분이 올바르게 작동하지 않으면 함수의 반환값도 올바르지 않을 것이다.

4. 두 예제에서 사용된 `get_accuracy` 함수의 동작을 비교 확인하는 실험

1) 실험 개요

두 가지 예제에서 사용된 `get_accuracy` 함수의 동작을 비교하고 그 차이에 대해 설명하기 위해 실행했다. 예제 1은 `2_1_knn_handwritten_digits_recognition_varying_k.py`이고 예제 2는 `1_1_checking_knn_model_accuracy.py`이나, 현재 코드로 지칭한다.

2) 실험 방법

두 예제에서는 동일한 `get_accuracy` 함수를 사용하여 예측값과 실제 레이블 간의 정확도를 계산한다. 그러나 두 예제 간에 정확도가 다르게 나타나는 것을 관찰했다. 특히, 현재 코드에서 정확도가 50%대의 낮은 정확도를 출력하고 있다. 실험을 통해 이러한 차이를 이해하기 위해 `get_accuracy` 함수의 동작 과정을 확인하였다.

```
def get_accuracy(predictions, labels):
    print(f'results의 shape:             {predictions.shape} ')
    print(f'np.squeeze(results)의 shape: {np.squeeze(predictions).shape} ')
    print(f'labels의 shape:              {labels.shape} ')
    accuracy = (np.squeeze(predictions) == labels).mean()
    return accuracy * 100
```

이처럼 코드를 수정해 실행해 확인 과정을 거쳤다.

3) 실험 결과 및 분석

예제 1: `2_1_knn_handwritten_digits_recognition_varying_k.py`

- 결과 확인:

```
...

results의 shape:             (2500, 1)
np.squeeze(results)의 shape: (2500,)
labels의 shape:              (2500,)
```

```
k=1 : 93.72
results의 shape:              (2500, 1)
np.squeeze(results)의 shape: (2500,)
labels의 shape:               (2500,)
k=2 : 91.96

...
```

- 분석:
  - `results`는 (2500, 1)의 shape를 가지고 있다. 따라서 `np.squeeze(results)`를 사용하여 1차원 배열로 변환한 후, 정확도를 계산하면 적절한 결과가 나타난다. 실제 레이블과의 비교 시 차원이 일치하여 정확도를 올바르게 측정할 수 있기 때문이다. 이 경우, 정확도는 90%대로 측정되었다.

현재 코드: `1_1_checking_knn_model_accuracy.py`

- 결과 확인:

```
results의 shape:              (400, 1)
np.squeeze(results)의 shape: (400,)
labels의 shape:               (400, 1)
k=1: Accuracy2= 50.06
```

- 분석:
  - `results`와 `labels`의 shape가 같다. 이로 인해 `np.squeeze(results)`를 사용하여 1차원 배열로 변환하여 `labels`와의 비교 시 차원이 맞지 않아 정확한 비교가 이루어지지 않는다. 결과적으로 정확도는 실제보다 낮은 값을 반환하게 된다. 이러한 차이는 데이터의 차원과 형태가 정확도 계산에 영향을 미침을 보여준다.

4) 결론 및 개선 방향

위 실험을 통해 데이터의 차원과 형태가 정확도에 중요한 영향을 미친다는 것을 확인할 수 있었다. 따라서 정확도를 계산할 때는 `results`와 `labels`의 차원과 형태를 일치시키는 것이 중요하다. 이를 위해 데이터의 shape를 확인하고 필요에 따라 reshape 작업을 수행하여 일관된 비교를 보장해야 한다. 이러한 수정은 정확도 계산 함수에 적용되어야 하며, 현재 코드에서는 데이터의 shape를 맞추는 전처리 단계를 추가함으로써 문제를 해결할 수 있다. 이를 통해 모델의 성능을 정확하게 측정할 수 있을 것이다.

5. 해결 방법

```
# 코드 수정 방법 1
def get_accuracy(predictions, labels):
    accuracy = (np.squeeze(predictions) == np.squeeze(labels)).mean()
    return accuracy * 100
```

```
# 코드 수정 방법 2
def get_accuracy(predictions, labels):
```

```
    accuracy = (predictions == labels).mean()
    return accuracy * 100
```

`results`와 `labels`의 차원과 형태를 일치시켜 비교하도록 코드를 수정했다.

```
test=train: k=1, num of test data=400 -------
testing time: whole=0.00, unit=0.00
results: <class 'numpy.ndarray'> (400, 1)
neighbours: <class 'numpy.ndarray'> (400, 1)
dist: <class 'numpy.ndarray'> (400, 1)
test=train: L=400, k=1: Accuracy=99.00%
k=1: Accuracy2= 99.00
```

수정 이후, 50%대이던 Accuracy2가 95% 이상의 높은 정확도로 출력된다. 더하여 Accuracy2값이 다른 방법으로 정확도를 계산했던 Accuracy값과 일치하는 것을 확인할 수 있다.