# Title : sklearn Digits Data를 통한 Data Splitting 실습 보고서

💡 **ID:2021220699**
**Name: Eunchan Lee**
**Department: Electronics**

---

**목차**



---

## HoldOut for Digits Data

홀드아웃 방법(Holdout Method)은 가지고 있는 한정된 데이터를 단순히 Training set과 Test set으로 적절한 비율로 나눠서 실험 설계를 하는 매우 basic한 Data Spliting Method입니다.

- HO1: Incorrect Holdout Split
- HO2: Per-class Holdout Split
- HO3: Holdout Split by Random Sampling

- HO5: Stratified Random Sampling

  **for Digits!!**

강의자료에서 다루었던 위의 HO1,2,3,5 방식을 Digits 데이터에 대해 코드를 바꿔 실습해보며 최적의 성능을 찾아가보고자 합니다

## HO1: Incorrect Holdout Split for Digits

HO1: Incorrect Holdout 코드를 digits에 맞게끔 수정하고 train_neuralnet_digits 함수를 만들고 가중치를 변경해 가며 HO1를 진행해보았습니다.

**Digits Data Train Function 튜닝하기**

먼저 train_neuralnet_iris 함수를 모방한 train_neuralnet_digits함수를 만들고 가중치만 데이터에 맞게 피팅,튜닝해보는 작업을 거치는 실험을 해봤습니다

input_size와 output_size는 데이터의 크기에 맞게 64,10으로 고정으로 지정해주었습니다.

hidden_size가 성능에 가장 중요할 것이고 iters_num과 batch_size가 부수적으로 성능에 중요한 영향을 끼칠 것으로 판단하여 결과가 좋은 가중치를 찾아 보았습니다.

**#1. hidden_size = 64 , batch_size = 32 , iters_num = 100**

```
def train_neuralnet_digits(x_train, t_train, x_test, t_test,
                           input_size=64, hidden_size=64, output_size=10,
                           iters_num = 1000, batch_size = 32, learning_rate = 0.1,
                           verbose=True):
```

Incorrect한 Split의 훈련 후 정확도 결과는 아래와 같았습니다. 가중치를 바꿔볼 필요가 있어보였습니다.

```
In [40]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Optimization finished!
Training accuracy: 0.36
Test accuracy: 0.32

In [41]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Optimization finished!
Training accuracy: 0.27
Test accuracy: 0.23

In [42]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Optimization finished!
Training accuracy: 0.10
Test accuracy: 0.10

In [43]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Optimization finished!
Training accuracy: 0.10
Test accuracy: 0.10

In [44]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Optimization finished!
Training accuracy: 0.10
Test accuracy: 0.10

In [45]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Optimization finished!
Training accuracy: 0.40
Test accuracy: 0.35
```

**#2. hidden_size = 128 , batch_size = 32 , iters_num = 100**

```
def train_neuralnet_digits(x_train, t_train, x_test, t_test,
                           input_size=64, hidden_size=128, output_size=10,
                           iters_num = 1000, batch_size = 32, learning_rate = 0.1,
                           verbose=True):
```

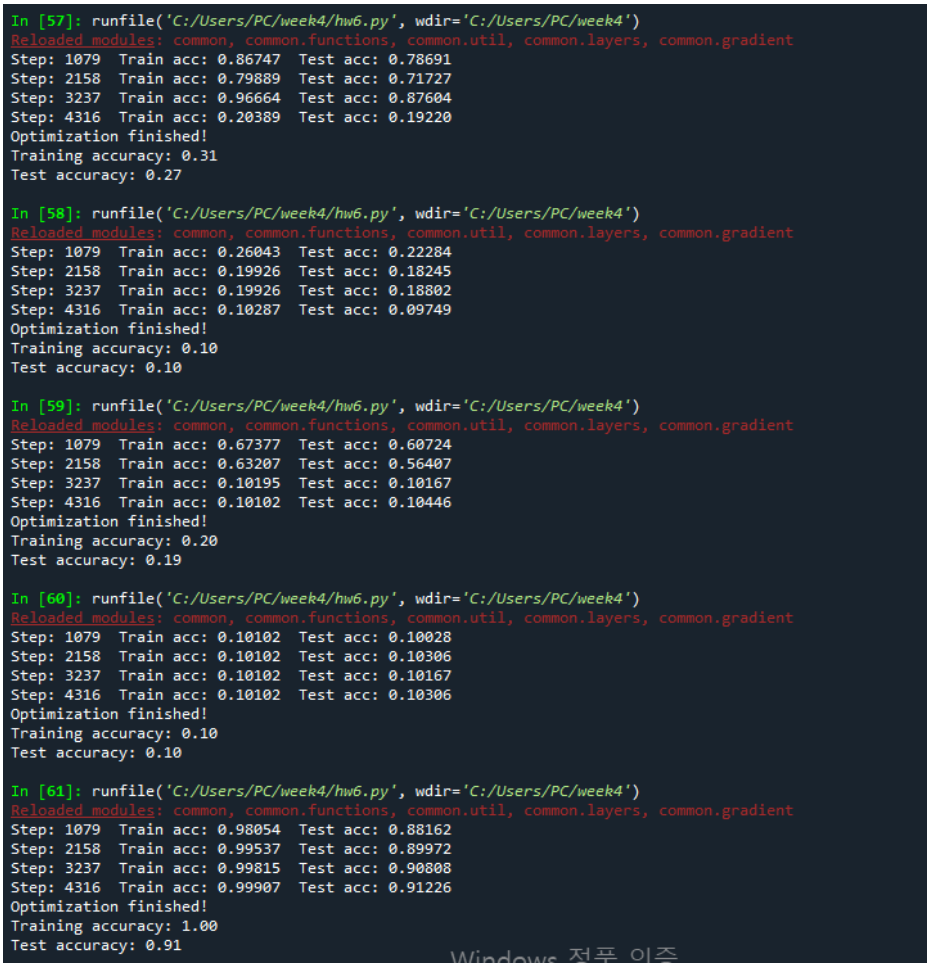
**#1에서 64→128로 hidden_size를 키워보았을때 결과를 보았습니다. 결과는 아래와 같이 복불복 느낌으로 0.10 ~ 0.92로 놀라운 범위폭을 보였습니다. 그러나 이는 별로 의미가 없다고 보여집니다. Incorrect 분류 자체가 test data가 쏠림현상에 의해 accuracy 값은 아예 랜덤일 수 밖에 없게 보여집니다.**

```
In [47]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Optimization finished!
Training accuracy: 1.00
Test accuracy: 0.92

In [48]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Optimization finished!
Training accuracy: 0.10
Test accuracy: 0.10

In [49]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Optimization finished!
Training accuracy: 0.13
Test accuracy: 0.11

In [50]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Optimization finished!
Training accuracy: 0.30
Test accuracy: 0.24

In [51]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Optimization finished!
Training accuracy: 0.63
Test accuracy: 0.58

In [52]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Optimization finished!
Training accuracy: 0.19
Test accuracy: 0.18

In [53]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Optimization finished!
Training accuracy: 0.24
Test accuracy: 0.21

In [54]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Optimization finished!
Training accuracy: 0.98
Test accuracy: 0.88
```

**#3. hidden_size=128, iters_num = 5000, batch_size = 32**

```
def train_neuralnet_digits(x_train, t_train, x_test, t_test,
                           input_size=64, hidden_size=128, output_size=10,
                           iters_num = 5000, batch_size = 32, learning_rate = 0.1,
                           verbose=True):
```

HO1:incorrect Split에 대해서 분석하는것은 더이상 의미가 없다고 판단하여 iters_num을 1000→5000으로 늘려보는 것을 마지막으로 HO1을 대상으로 하는 파라미터 조정은 더이상 하지 않았습니다.  결과는 아래와 같으며 #2와 별 차이를 느낄 수 없습니다.




이제 다음으로 넘어가 보겠습니다.

## HO2: Per-class Holdout Split for Digits

digits에 대해 HO2: Perclass Holdout 코드를 고려해 보았습니다.

→ **문제점** : Per-class를 Digits에서 구현하기 위해서는 Iris처럼 50을 단위로 생각해서 단순히 연산하는 것보다 더욱 복잡한 알고리즘을 필요로 하게 됩니다. 그러한 작업을 해서 0부터 9까지 10개의 Per-class를 구현하는 것은 불필요하다고 판단하여 HO2는 넘어가게 되었습니다.

## HO3: Holdout Split by Random Sampling for Digits

데이터를 Random Sampling하여 Test Train을 Holdout Split하는 HO3 코드를 Digits에 맞게 수정하여 적용시켜 보았습니다.

**최적의 매개변수 조합**

가중치를 피팅하는 과정에서 이번 실습에서 제가 찾은 최적의 매개변수 조합은 아래와 같았습니다. 물론 이는 주관적 판단에 근거하지만 저는 이 실습에서 아래 매개변수 조합으로 train 시키게 되었습니다.

```
def train_neuralnet_digits(x_train, t_train, x_test, t_test,
                           input_size=64, hidden_size=128, output_size=10,
                           iters_num = 5000, batch_size = 64, learning_rate = 0.1,
                           verbose=True):
```

결과는 아래와 같습니다.

평균 5번 진행시 4번은 0.97정도로 좋게 나오는 80% 꼴로 대체적으로 Test Accuracy가 높게 나오며, 낮게 나오는 경우는 Random Spliting이 안 좋게 나온 케이스로 볼 수 있습니다. Random 기반의 Split 치고는 굉장히 정확도가 좋았습니다만 가장 Best는 아니다 라는 것을 알 수 있었습니다.

```
In [3]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Step: 1079  Train acc: 1.00000  Test acc: 0.96518
Step: 2158  Train acc: 1.00000  Test acc: 0.97075
Step: 3237  Train acc: 1.00000  Test acc: 0.96936
Step: 4316  Train acc: 1.00000  Test acc: 0.97075
Optimization finished!
Training accuracy: 1.00
Test accuracy: 0.97

In [4]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Step: 1079  Train acc: 0.38832  Test acc: 0.39833
Step: 2158  Train acc: 0.12234  Test acc: 0.12535
Step: 3237  Train acc: 0.18721  Test acc: 0.15320
Step: 4316  Train acc: 0.21131  Test acc: 0.17967
Optimization finished!
Training accuracy: 0.21
Test accuracy: 0.18

In [5]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Step: 1079  Train acc: 1.00000  Test acc: 0.96936
Step: 2158  Train acc: 1.00000  Test acc: 0.97075
Step: 3237  Train acc: 1.00000  Test acc: 0.97075
Step: 4316  Train acc: 1.00000  Test acc: 0.97075
Optimization finished!
Training accuracy: 1.00
Test accuracy: 0.97

In [6]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Step: 1079  Train acc: 1.00000  Test acc: 0.96797
Step: 2158  Train acc: 1.00000  Test acc: 0.96518
Step: 3237  Train acc: 1.00000  Test acc: 0.96518
Step: 4316  Train acc: 1.00000  Test acc: 0.96518
Optimization finished!
Training accuracy: 1.00
Test accuracy: 0.97

In [7]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gradient
Step: 1079  Train acc: 1.00000  Test acc: 0.97354
Step: 2158  Train acc: 1.00000  Test acc: 0.97493
Step: 3237  Train acc: 1.00000  Test acc: 0.97354
Step: 4316  Train acc: 1.00000  Test acc: 0.97493
Optimization finished!
Training accuracy: 1.00
Test accuracy: 0.97
```

## HO5: Stratified Random Sampling for Digits

Stratified(층으로 나누는) Random Sampling을 Digits 데이터에 적용하는 실습을 해보았습니다.

코드를 단순히 iris → digits로 치환하여 실행해 보았습니다. 결과는 아래와 같았습니다.

0.97 정도의 매우 좋은 결과가 높은 확률로 나오는 것을 확인할 수 있습니다.

간간히 0.2정도의 낮은 성능이 포착되긴 했지만 HO3에 비해 체감상 더 정확도가 좋았습니다.

```
In [11]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gra
test: 71 73 71,  training: 107 109 106
Step:  539  Train acc: 0.99814  Test acc: 0.96106
Step: 1078  Train acc: 1.00000  Test acc: 0.96523
Step: 1617  Train acc: 1.00000  Test acc: 0.96662
Step: 2156  Train acc: 1.00000  Test acc: 0.96801
Step: 2695  Train acc: 1.00000  Test acc: 0.96801
Step: 3234  Train acc: 1.00000  Test acc: 0.96801
Step: 3773  Train acc: 1.00000  Test acc: 0.96523
Step: 4312  Train acc: 1.00000  Test acc: 0.96662
Step: 4851  Train acc: 1.00000  Test acc: 0.96523
Optimization finished!
Training accuracy: 1.00
Test accuracy: 0.97

In [12]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gra
test: 71 73 71,  training: 107 109 106
Step:  539  Train acc: 0.99814  Test acc: 0.96106
Step: 1078  Train acc: 1.00000  Test acc: 0.96523
Step: 1617  Train acc: 1.00000  Test acc: 0.96662
Step: 2156  Train acc: 1.00000  Test acc: 0.96801
Step: 2695  Train acc: 1.00000  Test acc: 0.96801
Step: 3234  Train acc: 1.00000  Test acc: 0.96801
Step: 3773  Train acc: 1.00000  Test acc: 0.96523
Step: 4312  Train acc: 1.00000  Test acc: 0.96662
Step: 4851  Train acc: 1.00000  Test acc: 0.96523
Optimization finished!
Training accuracy: 1.00
Test accuracy: 0.97

In [13]: runfile('C:/Users/PC/week4/hw6.py', wdir='C:/Users/PC/week4')
Reloaded modules: common, common.functions, common.util, common.layers, common.gra
test: 71 73 71,  training: 107 109 106
Step:  539  Train acc: 0.99814  Test acc: 0.96106
Step: 1078  Train acc: 1.00000  Test acc: 0.96523
Step: 1617  Train acc: 1.00000  Test acc: 0.96662
Step: 2156  Train acc: 1.00000  Test acc: 0.96801
Step: 2695  Train acc: 1.00000  Test acc: 0.96801
Step: 3234  Train acc: 1.00000  Test acc: 0.96801
Step: 3773  Train acc: 1.00000  Test acc: 0.96523
Step: 4312  Train acc: 1.00000  Test acc: 0.96662
Step: 4851  Train acc: 1.00000  Test acc: 0.96523
Optimization finished!
Training accuracy: 1.00
Test accuracy: 0.97
```

## Cross Validation for Digits Data

교차 검증법(Cross Validation)은 가지고 있는 한정된 데이터를 가지고 여러번 Training set과 Test set으로 나누어 보면서 가장 적절한 케이스를 골라서 그 케이스를  Split하여 높은 모델 성능을 취하는 방법입니다. 이는 한정적인 데이터에 대해서 the choice of the test set에 매우 의존적인 성능 변화를 보이는 HoldOut Split에 비해 매우 좋은 방법입니다.

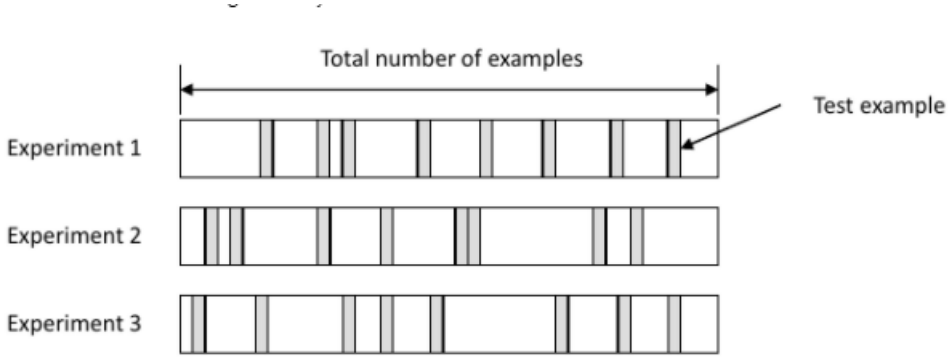
아래 두가지를 Digits에 대해 실습해보고 분석해보도록 하겠습니다.

- Random Subsampling: $K$ Data Splits
- $K$ -fold Cross Validation (KFCV)

**for Digits!!**

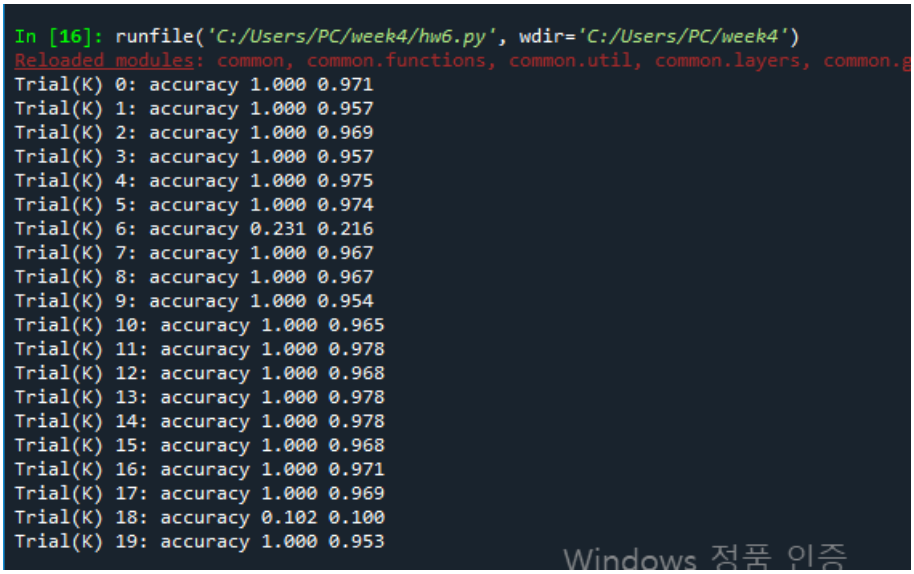
### Random Subsampling: $K$ Data Splits 실습해보기 for Digits

먼저 아래 그림과 같이 랜덤한 Spliting을 K번 해보고 가장 좋은 Trial을 보는 Random Subsampling: $K$ Data Splits 방법을 Digits에 적용해 보았습니다.




결과는 아래와 같으며 다음과 같이 분석할 수 있습니다.

- HoldOut과 유사한 정도의 정확도를 보입니다.
- K가  11, 13,14일 때 높은 정확도를 보였습니다.




### $K$-fold Cross Validation (KFCV)

코드가 없었기 때문에 적용해보지 못했습니다.

## 결론

- **Data Split Method는 제한된 데이터에서 딥러닝 모델의 성능을 분석할 수 있게 해준다.**
- **방식에는 HoldOut과 Cross Validation을 들 수 있다.**
- **Holdout 방식은 데이터를 train set과 test set으로 분류하며 train set 으로 모델을 훈련, test set으로 모델을 검증한다.**
- **Holdout 방식은 overfitting 문제가 발생할 수 있으며 교차 검증을 통해 이 문제를 해소할 수 있다.**
- **교차 검증(cross validation)은 train set을 train set + validation set으로 분리한 뒤, validation set을 사용해 검증하는 방식이다.**
- **K-Fold는 가장 일반적으로 사용되는 교차 검증 방법이다.**