ML Day9

▼ 홀수/짝수 구하기(2)

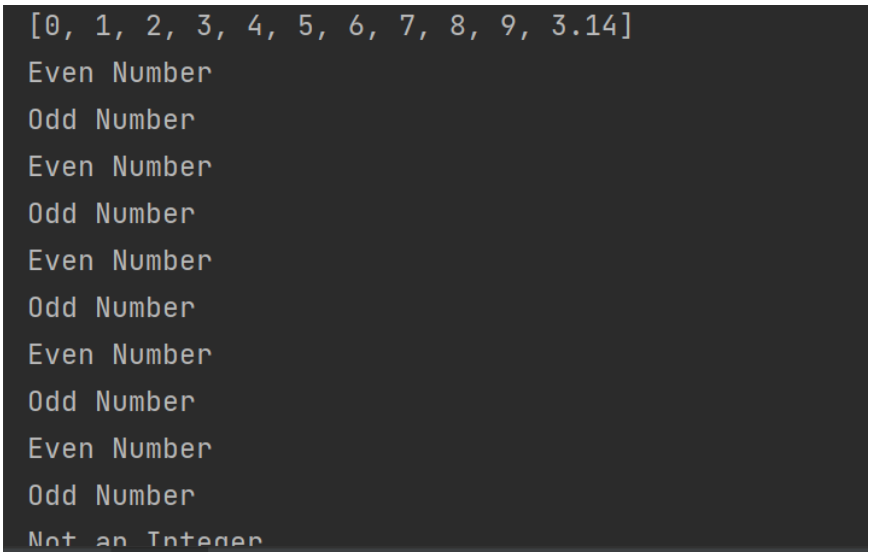
```python
# 홀수/짝수 구하기(2)
numbers = []
for num in range(10):
    numbers.append(num)

numbers.append(3.14)
print(numbers)

for num in numbers:
    if num % 2 == 0:
        print("Even Number")
    elif num % 2 == 1:
        print("Odd Number")
    else:
        print("Not an Integer")
```

▼ [3]: numbers 빈 list에 for구문으로 0~9까지 숫자를 append해준 후 실수 3.14를 append

▼ [9]: for구문으로 짝수와 홀수를 판별, 정수가 아닌



▼ n배수 들의 합 구하기(2, 3의 배)

```python
# n배수들의 합 구하기(2, 3의 배수)

multiple_of = 3

numbers = []
for num in range(100):
    numbers.append(num)

sum_multiple_of_n = 0
for num in numbers:
    if num % multiple_of == 0:
        sum_multiple_of_n += num
print(sum_multiple_of_n)

multiple_of = 2

numbers = []
for num in range(101):
    numbers.append(num)

sum_multiple_of_n = 0
for num in numbers:
    if num % multiple_of == 0:
        sum_multiple_of_n += num
print(sum_multiple_of_n)
```

▼ [5]: 비어있는 numbers list 생성한 후 for구문으로 0~99까지 numbers list에 append

```
1683
2550
```

▼ **최댓값, 최솟값 구하기**

```
1   # 최댓값, 최솟값 구하기
2
3   scores = [60, 40, 70, 20, 30]
4   M, m = 0, 100
5
6   for score in scores:
7       if score > M:
8           M = score
9       if score < m:
10          m = score
11
12  print("Max value:", M)
13  print("min value:", m)
14
15
16  scores = [20, 40, 70, 60, 30]
17  M, m = 0, 100
18
19  for score in scores:
20      if score > M:
21          M = score
22      elif score < m:
23          m = score
24
25  print("Max value:", M)
26  print("min value:", m)
```

▼ [4]: M, m = 0, 100 → 비교 대상을 지정

▼ [6]: for문을 돌면서 순서대로 M과 m값이 이전 값들과 비교되면서 결과가 바뀐다.

▼ [16]: list 원소 배열의 순서를 바꿈(20<>60)

▼ [19]: if조건에 만족하면 elif는 pass하므로 위 코드와 결과가 달라짐

```
Max value: 70
min value: 20
Max value: 70
min value: 30
```

▼ 최댓값, 최솟값 구하기(2)

```python
# 최댓값, 최솟값 구하기(2)

scores = [-20, 60, 40, 70, 120]

# method.1
M, m = scores[0], scores[0]
for score in scores:
    if score > M:
        M = score
    if score < m:
        m = score

print("Max value:", M)
print("Min value:", m)


# method.2
M, m = None, None
for score in scores:
    if M == None or score > M:
        M = score
    if m == None or score < m:
        m = score

print("Max value:", M)
print("Min value:", m)
```

```
Max value: 120
Min value: -20
Max value: 120
Min value: -20
```

▼ Min-Max Normalization

```python
# Min-Max Normalization

scores = [-20, 60, 40, 70, 120]

M, m = 0, 100
for score in scores:
    if score > M:
        M = score
    if score < m:
        m = score

print("Max value:", M)
print("Min value:", m)

for score_idx in range(len(scores)):
    scores[score_idx] = (scores[score_idx] - m) / (M - m)

print("scores after normalization:\n", scores)

M, m = 0, 1
for score in scores:
    if score > M:
        M = score
    if score < m:
        m = score
print("Max value:", M)
print("Min value:", m)
```

▼ [6]: scores list의 최대, 최소 값을 구한 후

▼ [15]: score list의 각각의 index에 위치하는 값들을 (scores[score_idx] - m) / (M - m)로 치환

▼ [21]: 바뀐 list로 다시 최대, 최소 값을 구함

```
Max value: 120
Min value: -20
scores after normalization:
 [0.0, 0.5714285714285714, 0.42857142857142855, 0.6428571428571429, 1.0]
Max value: 1.0
Min value: 0.0
```

▼ **최댓값, 최솟값의 위치 구하기**

```python
# 최댓값, 최솟값의 위치 구하기
scores = [60, -20, 40, 120, 70]
M, m = None, None
M_idx, m_idx = 0, 0

for score_idx in range(len(scores)):
    score = scores[score_idx]

    if M == None or score > M:
        M = score
        M_idx = score_idx

    if m == None or score < m:
        m = score
        m_idx = score_idx

print("M/M_idx:", M, M_idx)
print("m/m_idx:", m, m_idx)
```

▼ [4]: 최대, 최소 index의 시작점을 0으로 지정

▼ [6]: for구문을 돌며 최댓값, 최솟값을 구하는 동시에 각 원소가 위치하는 index도 함께 update됨

```
M/M_idx: 120 3
m/m_idx: -20 1
```

▼ **Sorting**

```python
# Sorting

scores = [40, 20, 30, 10, 50]
sorted_scores = []

for _ in range(len(scores)):
    M, M_idx = scores[0], 0

    for score_idx in range(len(scores)):
        if scores[score_idx] > M:
            M = scores[score_idx]
            M_idx = score_idx

    sorted_scores.append(M)
    del scores[M_idx]

    print("remaining scores:", scores)
    print("sorted scores:", sorted_scores, '\n')
```

```
remaining scores: [40, 20, 30, 10]
sorted scores: [50]

remaining scores: [20, 30, 10]
sorted scores: [50, 40]

remaining scores: [20, 10]
sorted scores: [50, 40, 30]

remaining scores: [10]
sorted scores: [50, 40, 30, 20]

remaining scores: []
sorted scores: [50, 40, 30, 20, 10]
```

▼ **Accuracy 구하기**

```python
# Accuracy 구하기

predictions = [0, 1, 0, 2, 1, 2, 0]
labels = [1, 1, 0, 0, 1, 2, 1]
n_correct = 0

for pred_idx in range(len(predictions)):
    if predictions[pred_idx] == labels[pred_idx]:
        n_correct += 1

accuracy = n_correct / len(predictions)
print('accuracy[%]:', accuracy*100, '%')
```

▼ [7]: predictions list의 길이만큼 for문으로 만약 predictions[pred_idx]값이 labels[pred_idx]와 같다면 n_correct에 1을 더한 후 다시 n_correct에 계속 대입

```
accuracy[%]: 57.14285714285714 %
```

▼ **Confusion Vector 구하기 (Class 별 accuracy를 확인하고 싶을 때 자주 쓰임)**

```python
# Confusion Vector 구하기 (Class 별로 accuracy를 확인하려 할 때 자주 쓰임)

predictions = [0, 1, 0, 2, 1, 2, 0]
labels = [1, 1, 0, 0, 1, 2, 1]

M_class = None
for label in labels:
    if M_class == None or label > M_class:
        M_class = label
M_class += 1

class_cnts, correct_cnts, confusion_vec = list(), list(), list()
for _ in range(M_class):
    class_cnts.append(0)
    correct_cnts.append(0)
    confusion_vec.append(None)

for pred_idx in range(len(predictions)):
    pred = predictions[pred_idx]
    label = labels[pred_idx]

    class_cnts[label] += 1
    if pred == label:
        correct_cnts[label] += 1
print("class_cnt:", class_cnts)
print("correct_cnts:", correct_cnts)

for class_idx in range(M_class):
    confusion_vec[class_idx] = correct_cnts[class_idx]/class_cnts[class_idx]
print("confusion vector:", confusion_vec)
```

▼ [7]: M_class = None 시작점을 지정해주고 labels list의 순서대로 for구문을 돌린다. 1, 2를 순서대로 만족하여 결론적으로 M_class = 2가 되고 2 += 1 —> 3이 된다. class 종류의 수를 구하기 위한 code.

▼ [12]: 3개의 빈 list를 만들고, M_class의 수만큼 각각 3개의 list에 append해준다. ([0, 0, 0], [0, 0, 0], [None, None, None]) → cofusion_vec list는 뒤 코드에서 결과를 구해서 바로 대입하기 위해 None값을 시작점으로 주었다.

▼ [18]: 각 클래스의 수를 구하기 위한 code, pred_idx로 for구문을 돌면서 class_cnts list에 labels list의 원소들이 가리키는(index) 자리에 1씩 추가

▼ [19]: pred_index(predictions list의 위치값)로 for구문을 돌면서 labels list에 같은 값이 있으면 correct_cnts list의 해당하는 위치에 +1을 함

▼

```
class_cnt: [2, 4, 1]
correct_cnts: [1, 2, 1]
confusion vector: [0.5, 0.5, 1.0]
```

▼ **Histogram 구하기**

```python
# Histogram 구하기

scores = [50, 20, 30, 40, 10, 50, 70, 80, 90, 20, 30]
cutoffs = [0, 20, 40, 60, 80]
histogram = [0, 0, 0, 0, 0]

for score in scores:
    if score > cutoffs[4]:
        histogram[4] += 1
    elif score > cutoffs[3]:
        histogram[3] += 1
    elif score > cutoffs[2]:
        histogram[2] += 1
    elif score > cutoffs[1]:
        histogram[1] += 1
    elif score > cutoffs[0]:
        histogram[0] += 1
    else:
        pass
print("histogram of the scores:", histogram)
```

▼ [7]: for구문으로 80보다 큰 점수는 histogram[4]에 1씩 추가,

▼ 60보다 큰 점수는 histogram[3]에 1씩 추가,

▼ 40보다 큰 점수는 histogram[2]에 1씩 추가,

▼ 20보다 큰 점수는 histogram[1]에 1씩 추가,

▼ 마지막으로 0보다 큰 점수는 histogram[0]에 추가,

▼ 위 순서대로 내려오며 조건에 만족하면 해당 index에 1씩 추

```
histogram of the scores: [3, 3, 2, 2, 1]
```

▼ 절댓값 구하기

```python
# 절댓값 구하기

numbers = [-2, 2, -1, 3, -4, 9]
abs_numbers = []

for num in numbers:
    if num < 0:
        abs_numbers.append(-num)
    else:
        abs_numbers.append(num)

print(abs_numbers)
```

```
[2, 2, 1, 3, 4, 9]
```

▼ Mahattan Distance

```python
# Manhattan Distance

v1 = [1, 3, 5, 2, 1, 5, 2]
v2 = [2, 3, 1, 5, 2, 1, 3]

m_distance = 0
for dim_idx in range(len(v1)):
    sub = v1[dim_idx] - v2[dim_idx]
    if sub < 0:
        m_distance += -sub
    else:
        m_distance += sub
print("Manhattan distance:", m_distance)
```

▼ 각 list에 같은 index값을 가진 값들을 v1 - v2 해주고, 음수인 경우 -(-sub)형태를 통해 양수로 바꿔준 후 모두 합산하여 준다.

```
Manhattan distance: 14
```

▼ Nested List(list 안 list) 만들기 & 원소 접근하기(Indexing)

```python
# Nested List 만들기 & 원소 접근하기

scores = [[10, 20, 30], [50, 60, 70]]
print(scores)
print(scores[0])
print(scores[1])
print(scores[0][0], scores[0][1], scores[0][2])
print(scores[1][0], scores[1][1], scores[1][2])
```

```
[[10, 20, 30], [50, 60, 70]]
[10, 20, 30]
[50, 60, 70]
10 20 30
50 60 70
```

▼ Nested List 원소 접근하기(2) - for구문

```python
# Nested List 원소 접근하기(2)- for구문

scores = [[10, 20, 30], [50, 60, 70]]

for student_scores in scores:
    print(student_scores)
    for score in student_scores:
        print(score)
```

```
[10, 20, 30]
10
20
30
```

▼ 학생별 평균 점수 구하기

```python
# 학생별 평균점수 구하기

scores = [[10, 15, 20], [20, 25, 30], [30, 35, 40], [40, 45, 50]]

n_class = len(scores[0])
student_score_means = []

for student_scores in scores:
    student_score_sum = 0
    for score in student_scores:
        student_score_sum += score
    student_score_means.append(student_score_sum/n_class)

print("mean of students' scores:", student_score_means)
```

▼ [6]: 점수의 평균을 구해서 집어 넣을 빈 list생성

▼ [8]: for문 안 for문으로 list내에 있는 list의 원소들을 합산하여준다.

▼ [sum, sum, sum, sum]형태로 나옴

```
mean of students' scores: [15.0, 25.0, 35.0, 45.0]
```

▼ 과목별 평균 점수 구하기

```python
# 과목별 평균점수 구하기

scores = [[10, 15, 20], [20, 25, 30], [30, 35, 40], [40, 45, 50]]

n_student = len(scores)
n_class = len(scores[0])

class_score_sums = list()
class_score_means = list()

# set the sum of class score as 0
for _ in range(n_class):
    class_score_sums.append(0)

# calculate the sum of class scores
for student_scores in scores:
    for class_idx in range(n_class):
        class_score_sums[class_idx] += student_scores[class_idx]

print("sum of classes' scores:", class_score_sums)

# calculate the mean of class scores
for class_idx in range(n_class):
    class_score_means.append(class_score_sums[class_idx] / n_student)

print("mean of classes' scores:", class_score_means)
```

▼ [5]: 과목별 점수 합산/평균을 넣을 빈 list생성

▼ [12]: n_class(len(scores[0])) = 3 만큼 for구문을 돌려 class_score_sums에 append → [0, 0, 0]

▼ [16]: 학생의 점수를 구하기 위해 scores list의 원소갯수만큼(4) for구문을 돌리고 다시 그 안에서 class_index(과목별 index)를 n_class만큼 for구문을 돌려 각 과목의 index에 위치한 값들을 더하여 class_score_sums list에 대입하여 준다.

▼ [23]: 과목 별 index로 n_class만큼 for 구문을 돌려 각 과목의 합산을 학생의 수로 나눠주어 과목 별 평균을 내고 그 값을 class_score_means list에 대입

```
sum of classes' scores: [100, 120, 140]
mean of classes' scores: [25.0, 30.0, 35.0]
```

▼ **Mean Subtraction(5)**

```python
# Mean Subtraction(5)

scores = [[10, 15, 20], [20, 25, 30], [30, 35, 40], [40, 45, 50]]

n_student = len(scores)
n_class = len(scores[0])

class_score_sums = []
class_score_means = []
for _ in range(n_class):
    class_score_sums.append(0)

for student_scores in scores:
    for class_idx in range(n_class):
        class_score_sums[class_idx] += student_scores[class_idx]

for class_idx in range(n_class):
    class_score_means.append(class_score_sums[class_idx] / n_student)

print("mean of classes' scores:", class_score_means)

for student_idx in range(n_student):
    for class_idx in range(n_class):
        scores[student_idx][class_idx] -= class_score_means[class_idx]

class_score_sums = []
class_score_means = []
for _ in range(n_class):
    class_score_sums.append(0)

for student_scores in scores:
    for class_idx in range(n_class):
        class_score_sums[class_idx] += student_scores[class_idx]

for class_idx in range(n_class):
    class_score_means.append(class_score_sums[class_idx] / n_student)

print("mean of classes' scores:", class_score_means)
```

▼ [22]: 위에서 구한 과목 별 평균을 가지고 각 학생별 index로 n_student만큼 for구문을 돌리고 다시 그 안에서 과목 별 index로 n_class만큼 for구문을 돌려 학생의 과목 별 점수에서 과목 별 평균을 뺀 값을 scores list로 대입하여 준다.

▼ [26]: 업데이트 된 scores list로 다시 평균을 구한다.

```
mean of classes' scores: [25.0, 30.0, 35.0]
mean of classes' scores: [0.0, 0.0, 0.0]
```

▼ 분산과 표준편차(5)

```python
# 분산과 표준편차(5)

scores = [[10, 15, 20], [20, 25, 30], [30, 35, 40], [40, 45, 50]]

n_student = len(scores)
n_class = len(scores[0])

class_score_sums = []
class_score_square_sums = []

class_score_variances = []
class_score_stds = []

# set the sum of class scores as 0
for _ in range(n_class):
    class_score_sums.append(0)
    class_score_square_sums.append(0)

# classes' sums, squared sums
for student_scores in scores:
    for class_idx in range(n_class):
        class_score_sums[class_idx] += student_scores[class_idx]
        class_score_square_sums[class_idx] += student_scores[class_idx]**2

# classes' variances
for class_idx in range(n_class):
    mos = class_score_square_sums[class_idx] / n_student
    som = (class_score_sums[class_idx] / n_student)**2

    variance = mos - som
    std = variance**0.5

    class_score_variances.append(variance)
    class_score_stds.append(std)

# classes' standard deviations
print("variances:", class_score_variances)
print("standard deviations:", class_score_stds)
```

▼ [8]: 순서대로 과목 별 점수의 합, 과목 별 점수의 제곱의 합, 과목 별 점수의 분산, 과목 별 점수들의 표준 편차를 집어넣을 비어 있는 list생성

▼ [15]: 과목점수들의 합과 제곱의 합을 넣어 줄 list에 for구문을 통해 미리 0을 대입해 줌 → [0, 0, 0], [0, 0, 0]

▼ [20]: scores list의 길이로 for 구문을 돌고 그 안에서 다시 n_class만큼for구문을 돌면서 학생 별 각 과목에 해당하는 값들의 합산을 class_score_sums에 대입하고, 학생 별 각 과목 점수의 제곱이 되는 값들을 class_score_square_sums에 대입

▼ [26]: mean of square(mos), square of mean(som) 변수에 값을 대입

```
variances: [125.0, 125.0, 125.0]
standard deviations: [11.180339887498949, 11.180339887498949, 11.180339887498949]
```