ML Day10

▼ **Standardization(5)**

```python
# Standardization(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_means = []

class_score_variances = []
class_score_stds = []

for _ in range(n_class):
    class_score_sums.append(0)
    class_score_square_sums.append(0)


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

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

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)
print("standard deviations:", class_score_stds)
```

```
for student_idx in range(n_student):
    for class_idx in range(n_class):
        score = scores[student_idx][class_idx]
        mean = class_score_means[class_idx]
        std = class_score_stds[class_idx]

        scores[student_idx][class_idx] = (score - mean)/std

print(scores)

class_score_sums = []
class_score_square_sums = []

class_score_means = []

class_score_variances = []
class_score_stds = []

for _ in range(n_class):
    class_score_sums.append(0)
    class_score_square_sums.append(0)

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


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)

print("standard deviations:", class_score_stds)
```

▼ [21]: student_scores 변수로 scores list를 for구문으로 돌면서, 다시 class_idx 변수로 n_class만큼 for구문을 돌며 class_score_sums list의 각 과목에 해당하는 인덱스 위치에 학생들의 과목 점수(student_scores[class_idx])를 합산하여 준다. 또한 각 학생의 과목 점수의 제곱을 한 값 계속 더해주어 class_score_square_sums에 대입

▼ [26]: 과목의 인덱스값(class_idx)으로 n_class만큼 for구문을 돌며 과목의 index에 해당하는 과목 점수의 평균을 class_score_means에 append

```
standard deviations: [11.180339887498949, 11.180339887498949, 11.180339887498949]
[[-1.3416407864998738, -1.3416407864998738, -1.3416407864998738], [-0.4472135954999579, -0.4472135954999579, -0.4472135954999579], [0.4472135954999579, 0.4472135954999579, 0.4472
standard deviations: [1.0, 1.0, 1.0]
```

▼ Hadamard Product(5)

```python
# Hadamard Product(5)

vectors = [[1, 11, 21],
           [2, 12, 22],
           [3, 13, 23],
           [4, 14, 24]]
h_prod = []
for dim_data in vectors:
    dim_prod = 1
    for dim_val in dim_data:
        dim_prod *= dim_val
    h_prod.append(dim_prod)

print(h_prod)
```

▼ [9]: dim_data = vectors[index], dim_val = vectors[index][index]

▼ dim_prod = 1은 아래 for구문에서 곱하여 주기 위하여 시작값을 1로 주었다.
dim_prod는 1*11*21, 2*12*22, 3*13*23, 4*14*24 값이 들어간다.

```
[231, 528, 897, 1344]
```

▼ Vector Norm(4)

```python
# Vector Norm(4)

vectors = [[1, 11, 21],
           [2, 12, 22],
           [3, 13, 23],
           [4, 14, 24]]
n_vector = len(vectors[0])
v_norms = []

for _ in range(n_vector):
    v_norms.append(0)
print(v_norms)

for dim_data in vectors:
    for dim_idx in range(n_vector):
        v_norms[dim_idx] += dim_data[dim_idx]**2
print(v_norms)

for vec_idx in range(n_vector):
    v_norms[vec_idx] **= 0.5
print(v_norms)
```

▼ [10]: n_vector(3)의 길이만큼 for구문을 돌며 v_norms에 0을 append

▼ [14]: vectors의 길이(4)만큼 for구문을 돌면서 다시 안에 for구문으로 n_vector만큼 각 dim_idx에 해당하는 값의 제곱을 합산하여 v_norms[dim_idx]에 대입.

▼ [19]: for 구문을 돌며 n_vector만큼 v_norms[vec_idx]**0.5한 값을 다시 대입해준다.

```
[0, 0, 0]
[30, 630, 2030]
[5.477225575051661, 25.099800796022265, 45.05552130427524]
```

▼ **Making Unit Vectors(4)**

```python
# Making Unit Vectors(4)

vectors = [[1, 11, 21],
           [2, 12, 22],
           [3, 13, 23],
           [4, 14, 24]]

n_vector = len(vectors[0])
v_norms = []

for _ in range(n_vector):
    v_norms.append(0)

for dim_data in vectors:
    for dim_idx in range(n_vector):
        v_norms[dim_idx] += dim_data[dim_idx]**2

for vec_idx in range(n_vector):
    v_norms[vec_idx] **= 0.5
print(v_norms)
n_dim = len(vectors)
n_vector = len(vectors[0])

for dim_idx in range(n_dim):
    for vec_idx in range(n_vector):
        vectors[dim_idx][vec_idx] /= v_norms[vec_idx]

n_vector = len(vectors[0])
v_norms = []

for _ in range(n_vector):
    v_norms.append(0)

for dim_data in vectors:
    for dim_idx in range(n_vector):
        v_norms[dim_idx] += dim_data[dim_idx]**2

for vec_idx in range(n_vector):
    v_norms[vec_idx] **= 0.5
print(v_norms)
```

▼ [11]: v_norms list에 for구문을 돌려 n_vector만큼의 0을 append해줌

▼ [14]: vectors의 길이(4)만큼 for문을 돌리고 그 안에 for문을 n_vector의 길이만큼 돌려 각 dim_inx에 해당하는 dim_data 들의 제곱을 합산하여 v_norms list에 해당하는 index위치에 값을 대입

▼ [18]: n_vector만큼 for문을 돌려 v_norms의 각 index위치의 값들을 0.5제곱(루트를 씌운 값)해준다.

▼ [24]: n_dim(=len(vectors))길이 만큼 for문 안에 n_vector만큼 for문을 돌려 vectors의 같은 dim_idx, vec_idx의 위치하는 값을 v_norms[vec_idx]로 나눠준다.

▼ [28~]: 다시 v_norms를 구해준다.

```
[5.477225575051661, 25.099800796022265, 45.05552130427524]
[0.9999999999999999, 1.0, 1.0]
```

▼ Dot Product(4)

```python
# Dot Product(4)

vectors = [[1, 11],
           [2, 12],
           [3, 13],
           [4, 14]]

d_prod = 0
for dim_data in vectors:
    d_prod += dim_data[0]*dim_data[1]
print("dot product:", d_prod)
```

▼ [8]: d_prod의 시작 값을 0으로 지정

▼ [9]: vectors의 길이만큼(4) dim_data[0], dim_data[1]를 곱한 결과를 d_prod에 합산하여 준다.

```
dot product: 130
```

▼ **Euclidean distance(4)**

```python
# Euclidean Distance(4)

vectors = [[1, 11],
           [2, 12],
           [3, 13],
           [4, 14]]
e_distance = 0

for dim_data in vectors:
    diff = dim_data[0] - dim_data[1]
    diff_square = diff**2

    e_distance += diff_square

e_distance **= 0.5
print("Euclidean distance:", e_distance)
```

▼ [9]: vectors의 길이만큼(4) for문을 돌며, dim_data[0] - dim_data[1] 값들을 vectors의 index순서대로 diff 변수에 대입. 또한 diff의 제곱을 diff_square에 대입.

▼ [13]: 위에서 나온 diff_square값들을 합산하여 e_distance에 대입한다.

▼ [15]: e_distance의 0.5 제곱(루트를 씌운 값) = e_distance

```
Euclidean distance: 20.0
```

▼ **과목별 최고점, 최우수 학생 구하기**

```python
# 과목별 최고점, 최우수 학생 구하기

scores = [[10, 40, 20],
          [50, 20, 60],
          [70, 40, 30],
          [30, 80, 40]]

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

M_classes = scores[0]
M_idx_classes = []

for _ in range(n_class):
    M_idx_classes.append(0)

for student_idx in range(n_student):
    student_scores = scores[student_idx]

    for class_idx in range(n_class):
        score = student_scores[class_idx]
        if score > M_classes[class_idx]:
            M_classes[class_idx] = score
            M_idx_classes[class_idx] = student_idx

print("Max scores:", M_classes)
print("Max score indices:", M_idx_classes)
```

▼ [14]: n_class의 길이만큼 for문으로 비어 있었던 M_idx_classes에 0을 append

▼ [17]: n_student만큼 student_scores = scores[student_idx]로 지정하고 for문을 돌린다. for문 안에 for문으로 n_class만큼 돌면서 score는 student_scores[class_idx]일 때 score가 M_classes[class_idx]보다 클 경우, M_classes[class)idx] = score가 되고 M_idx_classes[class_idx]는 student_idx가 된다. → 과목 별 최고 점(M_classes)과 최고 점을 가진 학생(M_idx_classes)을 구한다.

```
Max scores: [70, 80, 60]
Max score indices: [2, 3, 1]
```

▼ **One-hot Encoding**

```
1   # One-hot Encoding
2
3   labels = [0, 1, 2, 1, 0, 3]
4
5   n_label = len(labels)
6   n_class = 0
7   for label in labels:
8       if label > n_class:
9           n_class = label
10  n_class += 1
11
12  one_hot_mat = []
13  for label in labels:
14      one_hot_vec = []
15      for _ in range(n_class):
16          one_hot_vec.append(0)
17      one_hot_vec[label] = 1
18      one_hot_mat.append(one_hot_vec)
19
20  print(one_hot_mat)
```

▼ [7]: labels list 길이만큼을 for문으로 돌면서 label이 n_class(시작값=0)보다 크면 n_class = label이 된다. 1번의 for문이 끝날때마다 +1씩 해준다.

▼ [10]: for문을 돌고 나온 n_class = 3이므로 n_class +=1 은 n_class = 4가 된다.

▼ [12]: 비어있는 one_hot_mat list를 만든다.

▼ [13]: labels길이 만큼 for문을 돌며 one_hot_vec의 빈 list를 만들고 다시 그 안에서 n_class길이만큼의 for문을 돌며 one_hot_vec에 0값을 추가해준 후 one_hot_vec list의 labels의 label index 위치가 1로 변경된다. 즉 for문을 돌며 labels의 label index위치가 1로 변경되고 나머지는 0으로 바뀐 one_hot_vec list가 one_hot_mat list에 append된다.

```
[[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 1, 0, 0], [1, 0, 0, 0], [0, 0, 0, 1]]
```

▼ Accuracy 구하기(2)

```python
# Accuracy 구하기(2)

predictions = [[1, 0, 0, 0], [0, 0, 1, 0],
               [0, 0, 1, 0], [1, 0, 0, 0],
               [1, 0, 0, 0], [0, 0, 0, 1]]
labels = [0, 1, 2, 1, 0, 3]

n_label = len(labels)
n_class = 0
for label in labels:
    if label > n_class:
        n_class = label
n_class += 1

one_hot_mat = []
for label in labels:
    one_hot_vec = []
    for _ in range(n_class):
        one_hot_vec.append(0)
    one_hot_vec[label] = 1
    one_hot_mat.append(one_hot_vec)

n_pred = len(predictions)
n_class = len(predictions[0])
accuracy = 0
for pred_idx in range(n_pred):
    prediction = predictions[pred_idx]
    label = one_hot_mat[pred_idx]

    correct_cnt = 0
    for class_idx in range(n_class):
        if prediction[class_idx] == label[class_idx]:
            correct_cnt += 1

        if correct_cnt == n_class:
            accuracy += 1

accuracy /= n_pred
print("accuracy:", accuracy)
```

▼ [~21]: One-hot Encoding을 한 후

▼ [23]: n_pred = predictions의 길이, n_class = predictions[0]의 길이, accuracy 시작값을 0으로 지정

▼ [26]: n_pred 길이 만큼 for문을 돌며 prediction 은 predictions[pred_idx], label = one_hot_mat[pred_idx]로 대입된다.

```
accuracy: 0.6666666666666666
```