2주차 필수 과제 : 5늘의 스터디_5조

Poisson 분포의 모수 λ(람다)를 추정하는 Newton-Raphson 방법을 구현한 것

1. 다음의 데이터는 전등의 수명을 나타낸 자료이다.

   3, 5, 7, 18, 43, 85, 91, 98, 100, 130, 230, 487

   위의 시간이 지수분포 $Exp(\lambda)$를 따른다고 가정해보자.

   1.1 p45를 참고해 기울기가 0이 지점의 최적의 λ를 구하시오.

   1.2 뉴턴법을 이용하여 최적의 λ를 구하는 알고리즘을 작성해보시오.

Poisson 분포는 일정 시간 또는 공간 간격 내에서 사건이 발생하는 횟수를 모델링하는 데 사용됩니다. λ는 사건의 평균 발생률을 나타냅니다.

```
#1번문항
import numpy as np
data = np.array([3,5,7,18,43,85,91,98,100,130,230,487])
#초기 람다 값 =(1/표본평균)
lambda_init =1 / np.mean(data)
#뉴턴법 사용
for _ in range(100):
    gradient =-len(data) / lambda_init + sum(data)
    hessian =len(data) / lambda_init**2
    lambda_new = lambda_init - gradient / hessian
    if np.abs(lambda_new - lambda_init) < 1e-6:
        break
    lambda_init = lambda_new
print(f"Estimated lambda: {lambda_new}")
```

결과값: Estimated lambda: 0.009252120277563608

λ에 대한 우도 함수의 도함수가 λ = 1 / 평균에서 0이기에 교재에서 설명한 미분 없이, 람다 추정 값을 위처럼 둘 수 있습니다.

데이터셋은 사건의 개수를 나타내는 값들로 구성합니다.

λ초기화를 진행합니다. (데이터의 평균값의 역수로 초기화)

이후 newton-raphson 반복합니다.

```
data = np.array([3, 5, 7, 18, 43, 85, 91, 98, 100, 130, 230, 487])
# mean
mean = np.mean(data)
lambda_estimate =1 / mean
print(f"Estimated lambda: {lambda_estimate}")
```

결과값 : Estimated lambda: 0.009252120277563608

동일한 데이터셋을 사용하여 poission 분포의 모수(λ) 추정합니다.
데이터의 평균을 계산하고 그 평균의 역수를 취하여 λ 추정합니다.
추정된 λ 값을 출력합니다.

2. $A = \begin{pmatrix} 3 & 0 \\ 4 & 5 \end{pmatrix}$ 일때, U, Σ, V^T를 구해보시오.

```
#2
import numpy as np
A = np.array([[3, 0], [4, 5]])  #2x2 크기의 행렬 A를 정의
U, Sigma, VT = np.linalg.svd(A)
r =4
U_r = U[:, :r]
Sigma_r = np.diag(Sigma[:r])
VT_r = VT[:r, :]
A_r = U_r @ Sigma_r @ VT_r
#A_r이 처음에 정한 A와 같다면 문제에서 제시한 식이 참임을 보이게 됨
# A_r vs. A
print(np.allclose(A, A_r))
print(A)
print(A_r)
```

결과값:
True
[[3 0]
 [4 5]]
[[3.00000000e+00 1.33807967e-15]
 [4.00000000e+00 5.00000000e+00]]

linalg.svd 함수 이용: A를 특이값 U, 대각 행렬 Sigma, 전치된 특이값 VT로 분해

r을 주어진 순위로 설정하고, 특이값 분해 결과를 이용하여
순위가 r인 근사 행렬 A_r을 계산합니다.

np.allclose(A, A_r): 근사 행렬 A_r과 주어진 행렬 A의 동일 여부 출력
print(A): 원래 행렬 A를 출력
print(A_r): 근사 행렬 A_r을 출력
결론적으로, 주어진 행렬 A를 SVD를 이용하여 낮은 순위(r)의 근사 행렬 A_r로 분해하고, A와 A_r이 거의 동일한지를 검사하여 출력합니다.

3. p50의 표기를 참고해서, $\sigma_{r+1} = \ldots = \sigma_p = 0$이라고 가정해보자$(r < p)$. Σ_r을 대각성분이 $\sigma_1, \ldots, \sigma_r$ 인 대각행렬, U_r을 $u_1, .. u_r$의 행벡터를 가지는 행렬, V_r을 $v_1, .. v_r$의 행벡터를 가지는 행렬이라고 할 때, $A = U_r \Sigma_r V_r^T$임을 보이시오.

```
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso
X,y = make_regression(n_samples =300,
                      n_features =400,
                      n_informative =50,
                      n_targets =1,
                      bias =0.0,
                      noise =10.0,
                      random_state =42)
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =0.33,
random_state=42)
lr = LinearRegression()
lr = lr.fit(X_train, y_train)
x_pred = lr.predict(X_train)
y_pred = lr.predict(X_test)
from sklearn.metrics import mean_squared_error
print("MSE: ", mean_squared_error(y_test,y_pred))
```

결과값: MSE: 101402.29817674404

4. L2 규제를 포함한 로지스틱 모델을 구현해 p90에 예시와 비교해보시오

```
#4
## 해당 셀의 코드는 교재와 동일합니다
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
# 데이터 분리
X,y = load_breast_cancer(return_X_y=True, as_frame=False)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1234)
X_train, X_test = X_train[:,:3], X_test[:,:3]
y_train, y_test = y_train.reshape(-1,1), y_test.reshape(-1,1)
#스케일링, 절편 추가
train_mean, train_std = X_train.mean(axis=0), X_train.std(axis=0)
X_train, X_test = (X_train - train_mean)/train_std, (X_test - train_mean) / train_std
n,n_test = X_train.shape[0], X_test.shape[0]
X_train = np.append(np.ones((n,1)),X_train, axis=1)
X_test = np.append(np.ones((n_test,1)), X_test, axis=1)
# 이터레이션 값, 종료 조건
max_iter =10000
Tolerance =0.0001
# 초기 파라미터 설정
beta_old = np.ones((4,1))
# 모델 구축
for cnt in range(1, max_iter):
    W = np.zeros((n,n))
    p = np.zeros((n,1))
    for i in range(n):
        xb=np.exp((X_train[i].reshape(1,-1)@beta_old)[0][0])
        pi = xb / (1+xb)
        W[i][i] = pi*(1-pi)
        p[i]=pi
    left = np.linalg.inv(X_train.T @W@ X_train)
    right = X_train.T @ (y_train -p)
    update =0.0001 * (left@right)
    beta_new = beta_old + update
    if (np.linalg.norm(update) < Tolerance) : break
    if cnt % 1000 ==0:
        print(f'이터레이션: {cnt}, 업데이트 크기 : {np.linalg.norm(update)}')
    beta_old = beta_new
print(f'이터레이션: {cnt}, 업데이트 크기 : {np.linalg.norm(update)}')
print('₩n학습한 파라미터:₩n', beta_new)
```

```
이터레이션: 1000, 업데이트 크기 : 0.0014765820989698691
이터레이션: 2000, 업데이트 크기 : 0.0008288194947011425
이터레이션: 3000, 업데이트 크기 : 0.0005888637557052144
이터레이션: 4000, 업데이트 크기 : 0.0005016460961680814
이터레이션: 5000, 업데이트 크기 : 0.0004805887201861732
이터레이션: 6000, 업데이트 크기 : 0.0004921690754541085
이터레이션: 7000, 업데이트 크기 : 0.0005216582869966167
이터레이션: 8000, 업데이트 크기 : 0.0005614510237058703
이터레이션: 9000, 업데이트 크기 : 0.0006068663712747375
이터레이션: 9999, 업데이트 크기 : 0.0006545214496912331

학습한 파라미터:
 [[ 4.96797147e-01]
 [ 5.07895925e+00]
 [ 2.20184378e-03]
 [-5.65908436e+00]]
```

```
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
```

```
# 데이터 분리
X,y = load_breast_cancer(return_X_y=True, as_frame=False)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=1234)
X_train, X_test = X_train[:,:3], X_test[:,:3]
y_train, y_test = y_train.reshape(-1,1), y_test.reshape(-1,1)
#스케일링, 절편 추가
train_mean, train_std = X_train.mean(axis=0), X_train.std(axis=0)
X_train, X_test = (X_train - train_mean)/train_std, (X_test - train_mean) / train_std
n,n_test = X_train.shape[0], X_test.shape[0]
X_train = np.append(np.ones((n,1)),X_train, axis=1)
X_test = np.append(np.ones((n_test,1)), X_test, axis=1)
# 이터레이션 값, 종료 조건
max_iter =10000
Tolerance =0.0001
# 초기 파라미터 설정
beta_old = np.ones((4,1))
# 하이퍼파라미터 설정
lambda_ =0.1
# 모델 구축
for cnt in range(1, max_iter):
    W = np.zeros((n,n))
    p = np.zeros((n,1))
    for i in range(n):
        xb=np.exp((X_train[i].reshape(1,-1)@beta_old)[0][0])
        pi = xb / (1+xb)
        W[i][i] = pi*(1-pi)
        p[i]=pi
    left = np.linalg.inv(X_train.T @W@ X_train + lambda_ *
np.eye(X_train.shape[1]))
    right = X_train.T @ (y_train -p)
    update =0.0001 * (left@right)
    beta_new = beta_old + update
    if (np.linalg.norm(update) < Tolerance) : break
    if cnt % 1000 ==0:
        print(f'이터레이션: {cnt}, 업데이트 크기 : {np.linalg.norm(update)}')
    beta_old = beta_new
print(f'이터레이션: {cnt}, 업데이트 크기 : {np.linalg.norm(update)}')
print('₩n학습한 파라미터:₩n', beta_new)
```

```
이터레이션: 1000, 업데이트 크기 : 0.0010922117044632675
이터레이션: 2000, 업데이트 크기 : 0.0007226949938316695
이터레이션: 3000, 업데이트 크기 : 0.0005468750771264462
이터레이션: 4000, 업데이트 크기 : 0.0004646427026955055
이터레이션: 5000, 업데이트 크기 : 0.0004300030723686747
이터레이션: 6000, 업데이트 크기 : 0.0004209079859248419 7
이터레이션: 7000, 업데이트 크기 : 0.0004262250562825475 4
이터레이션: 8000, 업데이트 크기 : 0.0004397362419335891 4
이터레이션: 9000, 업데이트 크기 : 0.0004575737157295477
이터레이션: 9999, 업데이트 크기 : 0.0004771154663542016 4

학습한 파라미터:
 [[ 0.49730613]
 [ 3.71887338]
```

```
 [-0.01272472]
 [-4.28550355]]
```

```
## 모델 평가
right =0
for i in range(X_train.shape[0]):
    xb = np.exp((X_train[i].reshape(1,-1)@beta_old)[0][0])
    pi = xb/(1+xb)
    if (pi >=0.5 and y_train[i] ==1) or (pi <0.5 and y_train[i]==0):
        right +=1
print(f'학습 데이터셋 정확도 :{right / X_train.shape[0]*100: 2f}%')
right =0
for i in range(X_test.shape[0]):
    xb = np.exp((X_test[i].reshape(1,-1)@beta_old)[0][0])
    pi = xb/(1+xb)
    if (pi >=0.5 and y_test[i] ==1) or (pi <0.5 and y_test[i]==0):
        right +=1
print(f'테스트 데이터셋 정확도 :{right / X_test.shape[0]*100: 2f}%')
```

```
학습 데이터셋 정확도 : 83.989501%
테스트 데이터셋 정확도 : 78.191489%
```

(L2규제 내용을 제외한 나머지 부분은 교재와 동일)
모델 구축 과정에 L2 규제를 적용했습니다. 이때 하이퍼 파라미터는 0.1로 설정.
학습 데이터 셋 정확도는 교재가 높지만 테스트 데이터 셋의 정확도는 교재 76.06%, L2 규제 적용시 78.19%로 더 높은 것을 확인 할 수 있었습니다. 즉 과적합 방지의 효과가 나타난 것으로 볼 수 있겠습니다.

5. 3장 되새김 문제 2번을 풀어보시오

```
#5
from sklearn.linear_model import LogisticRegression
import numpy as np
import pandas as pd
n =300
np.random.seed(1234)
X1 = np.random.normal(0, 1, size=n)
X2 = np.random.normal(0, 1, size=n)
X3 = np.random.normal(0, 1, size=n)
X4 = np.random.normal(0, 1, size=n)
X5 = np.random.normal(0, 1, size=n)
X = pd.DataFrame({'X1': X1, 'X2': X2, 'X3': X3, 'X4': X4, 'X5': X5})
logistic_reg = LogisticRegression(fit_intercept =True, random_state =1234)
y = []
intercept =-1
beta = np.array([-1,-1,1,1,1]).reshape(-1,1)
for i in range(n):
```

```
    model = np.exp(((X.iloc[i].values.reshape(1,-1)@beta)[0][0]+ intercept)+
np.random.normal(0,3))
    soften = model / (1 +model)
    if soften <0.5:
        y.append(0)
    else:
        y.append(1)
y = pd.Series(y)
logistic_reg.fit(X,y)
prediction = logistic_reg.predict(X)
print(f"정확도:{(prediction == y).mean()*100}")
print("intercept", logistic_reg.intercept_)
print("coefficients:",logistic_reg.coef_)
```

```
정확도:75.66666666666667
intercept [-0.54586693]
coefficients: [[-0.5385009  -0.67087336  0.66779185  0.61243215  0.57936675]]
```

6. 선형 회귀모델 $Y = X\beta + \epsilon$ ($X \in \mathbb{R}^{n\times(p+1)}$는 disigned matrix)을 생각해보자. 부분집합 $M \in \{1, \dots, p\}$에 대해, $\beta_M = (\beta_0, (\beta_j)_{j\in M}), X_M = (1, (x_j)_{j\in M})$($x_j$는 X의 j번째 열벡터)라 하고, 다음과 같은 선형 모델 $Y \sim N(X_M\beta_M, \sigma^2 I)$을 모델 M이라고 하자. 충분히 작은 x에 대해 $log(1+x)$를 x로 근사할 수 있다는 사실을 이용하여, 모든 $j \in M^C$에 대해 $\beta_j = 0$일때, 모델 M의 AIC를 Mallows's C_p로 근사해 보이시오.

(참고. $C_p = \frac{SSE_p}{S^2} + 2(|M|+1) - n$, SSE_p는 모델 M의 SSE,
$AIC_M = -2log\hat{L}(M) + 2(|M|+1)$

```
#6
import numpy as np
from itertools import combinations
from sklearn.metrics import mean_squared_error
# 모델 M의 AIC를 Mallows's C_p로 근사하는 함수
def calculate_aic_mallows(X, y, model_features):
    n, p = X.shape
    X_M = np.hstack([np.ones((n, 1)), X[:, model_features]])
    beta_M = np.linalg.inv(X_M.T @ X_M) @ X_M.T @ y
    y_pred = X_M @ beta_M
    sse_p = mean_squared_error(y, y_pred) * n  # SSE_p
    s2 = sse_p / (n -len(model_features) -1)  # s^2
    aic_m =-2 * np.log(np.linalg.det(X_M.T @ X_M)) +2 * (len(model_features) +1)
# AIC_M
    cp = sse_p / s2 +2 * (len(model_features) +1) - n  # Mallows's C_p
    return aic_m, cp
# 예제 데이터 생성
n =1000
p =3
X = np.random.randn(n, p)
beta_true = np.random.randn(p +1, 1)
epsilon =0.1 * np.random.randn(n, 1)
y = X @ beta_true[1:] + beta_true[0] + epsilon
```

```
# 전체 변수 집합
full_feature_set = set(range(p +1))
# 변수 조합 생성
all_subsets = []
for r in range(1, p +1):
    subsets_r = list(combinations(range(p), r))  # 수정된 부분
    all_subsets.extend(subsets_r)
# 변수 조합에 대한 AIC_M, C_p 계산
aic_values = []
cp_values = []
for subset in all_subsets:
    aic_m, cp = calculate_aic_mallows(X, y, subset)
    aic_values.append(aic_m)
    cp_values.append(cp)
# 최적 모델 선택
best_model_index = np.argmin(aic_values)
best_model_features = list(all_subsets[best_model_index])
best_aic = aic_values[best_model_index]
best_cp = cp_values[best_model_index]
print(f"Best Model Features: {best_model_features}")
print(f"AIC for Best Model: {best_aic}")
print(f"Mallows's C_p for Best Model: {best_cp}")
```

```
Best Model Features: [0, 1, 2]
AIC for Best Model: -47.125862848619164
Mallows's C_p for Best Model: 4.000000000000114
```

Mallows의 Cp는 모형 내 예측 변수 개수의 균형을 맞추는 데 유용합니다.
일반적으로 Mallows의 Cp가 작으면서 모형의 예측 변수 개수에 상수(p)를 더한 값에 가까운 모형을 찾아야 하는데 이는, 모형이 실제 회귀 계수를 추정하고 미래 반응 값을 예측하는 데 있어서 비교적 정확하다(분산이 작다)는 것을 나타냅니다.

1. 함수 정의: calculate_aic_mallows
calculate_aic_mallows 함수는 주어진 설계 행렬 X, 응답 변수 y 및 변수 조합에 대해 AIC와 Mallows's C_p를 계산합니다.

2. 예제 데이터 생성

3. 전체 변수 집합과 변수 조합 생성
full_feature_set에는 전체 변수 집합이 포함됩니다.
all_subsets를 통해 모든 변수 조합을 생성하는 데 사용하고, combinations 함수를 사용하여 각 변수 개수에 대한 모든 조합을 생성합니다.

4. 변수 조합에 대한 AIC_M, C_p 계산
all_subsets에 대해 루프를 돌며 각 변수 조합에 대한 AIC와 Mallows's C_p를 계산합니다.

5. 최적 모델 선택

np.argmin(aic_values)를 사용하여 AIC가 최소인 모델의 인덱스를 찾습니다.

7. 교재 4장 되새김 문제 2번을 풀어보시오.

```
#7
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
x, y = make_regression(n_samples=300, n_features=400, n_informative=50,
n_targets=1, bias=0., noise=10.,
                       random_state=1234)
train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.33,
random_state=1234)
linear_reg = LinearRegression()
linear_reg.fit(train_x, train_y)
prediction = linear_reg.predict(test_x)
print(f"LinearRegression MSE: {mean_squared_error(test_y, prediction): .5f}")
lasso = Lasso()
lasso.fit(train_x, train_y)
prediction2 = lasso.predict(test_x)
print(f"Lasso MSE: {mean_squared_error(test_y, prediction2): .5f}")
```

```
LinearRegression MSE:  72001.11179
Lasso MSE:  457.39973
```

데이터 생성:
make_regression 함수를 사용하여 300개의 샘플과 400개의 특성으로 이루어진 회귀용 가상 데이터를 생성합니다.
n_informative, noise 매개변수를 통해 유용한 특성의 개수와 노이즈를 추가합니다.

데이터 분할: 테스트 데이터는 전체 데이터의 33%로 설정했습니다.

선형 회귀 모델 훈련과 예측:
LinearRegression 클래스를 사용하여 선형 회귀 모델을 생성하고 학습합니다.

라쏘 회귀 모델 훈련과 예측:
Lasso 클래스를 사용하여 라쏘 회귀 모델을 생성하고 학습합니다. 마찬가지로 학습 데이터를 사용합니다.
라쏘 회귀 모델을 이용하여 테스트 데이터에 대한 예측값을 계산 후 MSE를 출력합니다.