STAT/BIOST 571: Homework 4

Philip Pham

February 16, 2019

Problem 1: Iterative methods for solving GEE (10 points)

In this problem, you will compare variants on GEE (or general linear model) estimation procedures when applied to the example dental data set for the model on slide 2.71. Throughout, use a homoscedastic AR-1 working covariance matrix and estimate α and σ^2 using moment-based estimators. For each procedure, compute point estimates and robust sandwich-based standard errors that account for clustering. Where applicable, report how many interations are required to get convergence of all estimates ($\boldsymbol{\beta}$, α, and σ^2) in their 1st/2nd/3rd significant figures.

(a) *Use the non-iterative procedure we have employed previously; that is, first estimate $\boldsymbol{\beta}$ by OLS, then estimate the covariance parameters based on residuals from the OLS fit, and then restimate $\boldsymbol{\beta}$ using the updated working covariance matrix.*

Solution: The covariance structure for each cluster i is assumed to be the $m_i \times m_i$ matrix Σ_i, where $\Sigma_{ijj} = \sigma^2$ and $\Sigma_{ijk} = \sigma^2 \alpha^{|j-k|}$.

Given a working estimate of β, say $\hat{\beta}^{(t)}$, we compute working covariance matrices for each cluster

$$\hat{\Sigma}_i^{(t+1)} = \left(y_i - X_i\hat{\beta}^{(t)}\right)\left(y_i - X_i\hat{\beta}^{(t)}\right)^{\mathsf{T}}.$$

We get working estimates of the covariance parameters α and σ^2:

$$\left(\hat{\sigma}^2\right)^{(t+1)} = \frac{1}{\sum_{i=1}^n m_i} \sum_{i=1}^n \sum_{j=1}^{m_i} \hat{\Sigma}_{ijj}^{(t+1)}$$

$$\hat{\alpha}^{(t+1)} = \frac{1}{\left(\hat{\sigma}^2\right)^{(t+1)} \sum_{i=1}^n (m_i - 1)} \sum_{i=1}^n \sum_{j=1}^{m_i-1} \hat{\Sigma}_{i,j,j+1}^{(t+1)}.$$

Let $\hat{V}^{(t+1)}$ be the matrix where $\hat{V}_{ij}^{(t+1)} = \left(\hat{\sigma}^2\right)^{(t+1)} \left(\hat{\alpha}^{(t+1)}\right)^{|j-i|}$. Let $\hat{W}^{(t+1)} = \left(\hat{V}^{(t+1)}\right)^{-1}$.

Then, we can get an updated estimate of β:

$$\hat{\beta}^{(t+1)} = \left(\sum_{i=1}^n X_i^{\mathsf{T}} \hat{W}^{(t+1)} X_i\right)^{-1} \left(\sum_{i=1}^n X_i^{\mathsf{T}} \hat{W}^{(t+1)} y_i\right). \tag{1}$$

1

	Estimate	Standard error
β_0	22.615625	0.506629
β_1	0.784375	0.082753
β_2	-1.406534	0.793735
β_3	-0.304830	0.129649

Table 1: Estimate of β using OLS. Standard errors were estimated using Equation 2.

	Estimate	Standard error
β_0	22.749692	0.535465
β_1	0.769495	0.087378
β_2	-1.558229	0.838912
β_3	-0.285742	0.136895

Table 2: $\hat{\beta}^{(1)}$ is an updated estimates for β.

We can compute the covariance of this estimate with the sandwich estimate:

$$\text{côv}\left(\hat{\beta}^{(t+1)}\right) = \left(\sum_{i=1}^n X_i^{\mathsf{T}} \hat{W}^{(t+1)} X_i\right)^{-1} \left(\sum_{i=1}^n X_i^{\mathsf{T}} \hat{W}^{(t+1)} \hat{\Sigma}^{(t+1)} \hat{W}^{(t+1)} X_i\right) \left(\sum_{i=1}^n X_i^{\mathsf{T}} \hat{W}^{(t+1)} X_i\right)^{-1}, \quad (2)$$

where

$$\hat{\Sigma}^{(t+1)} = \frac{1}{n} \sum_{i=1}^n \left(y_i - X_i \hat{\beta}^{(t+1)}\right) \left(y_i - X_i \hat{\beta}^{(t+1)}\right)^{\mathsf{T}},$$

since we assume all clusters have the same covariance structure.

To obtain $\hat{\beta}^{(0)}$, we let $\hat{W}^{(0)} = I$. This is the OLS esimate. See Table 1 for the estimate and standard errors.

With this estimate, we obtain $\hat{\alpha}^{(1)} = 0.61136$ and $(\hat{\sigma}^2)^{(1)} = 4.90516$, which allows us to use $\hat{W}^{(1)}$ to get a new estimate of $\beta^{(1)}$. See Table 2.

(b) *Iterate the procedure in part (a), as suggested on slide 2.44.*

Solution: The final converged estimates can be found in Table 3.

For the covariance parameters, we found $\hat{\alpha} = 0.61353$ and $\hat{\sigma}^2 = 4.91065$.

It took only one iteration to find convergence in the first significant digit ($\hat{\beta}^{(1)}$). Already by the second iteration ($\hat{\beta}^{(2)}$), we have convergence in the second and third significant digits.

	Estimate	Standard error
β_0	22.750266	0.535584
β_1	0.769457	0.087397
β_2	-1.558861	0.839099
β_3	-0.285692	0.136925

Table 3: Final converged estimates for β.

Appendix

Code to produce tables is attached.

Iterative methods for solving GEE

```
In [1]: import numpy as np
        import pandas as pd
        from scipy import linalg
        from typing import List, NamedTuple

        class CovarianceParameters(NamedTuple('CovarianceParameters', [
            ('alpha', np.float64),
            ('sigma2', np.float64),
        ])):
            def make_correlation_matrix(self, size):
                correlation_matrix = np.eye(size)
                for i in range(size - 1):
                    for j in range(i + 1, size):
                        correlation_matrix[i, j] = correlation_matrix[j, i] = np.power(sel
        f.alpha, j - i)
                return correlation_matrix

            def make_covariance_matrix(self, size):
                return self.make_correlation_matrix(size)*self.sigma2

        class Cluster(NamedTuple('Cluster', [
            ('X', np.array),
            ('y', np.array),
            ('covariance', np.array),
        ])):
            """Cluster covariates, response, and covariance structure."""
```

```
In [2]: def estimate_covariance_parameters(clusters, beta):
    def _estimate_covariance_parameters(X, y, beta):
        epsilon = y - X.dot(beta)
        covariance_matrix = np.outer(epsilon, epsilon)
        sigma2 = np.diag(covariance_matrix)
        rho = [covariance_matrix[i, i + 1] for i in range(len(covariance_matrix) -
1)]
        return sigma2, rho

    sigma2 = []
    rho = []
    for cluster in clusters:
        cluster_sigma2, cluster_rho = _estimate_covariance_parameters(
            cluster.X, cluster.y, beta)
        sigma2.extend(cluster_sigma2)
        rho.extend(cluster_rho)
    sigma2 = np.mean(sigma2)
    return CovarianceParameters(alpha=np.mean(rho)/sigma2, sigma2=sigma2)

def estimate_beta(clusters: List[Cluster]):
    """Estimate beta under the assumption that clusters."""
    cluster_weights = [
        linalg.cho_solve(linalg.cho_factor(cluster.covariance), np.eye(len(cluster
.y))) for cluster in clusters
    ]
    projected_X = np.sum([
        cluster.X.T.dot(weights).dot(cluster.X)
        for cluster, weights in zip(clusters, cluster_weights)
    ], 0)
    projected_y = np.sum([
        cluster.X.T.dot(weights).dot(cluster.y)
        for cluster, weights in zip(clusters, cluster_weights)
    ], 0)
    beta = linalg.cho_solve(linalg.cho_factor(projected_X), projected_y)

    bread = linalg.cho_solve(linalg.cho_factor(projected_X), np.eye(len(projected_
X)))
    residuals = [cluster.y - cluster.X.dot(beta) for cluster in clusters]
    sigma = np.mean([np.outer(residual, residual) for residual in residuals], 0)
    meat = np.sum([
        cluster.X.T.dot(weights).dot(sigma).dot(weights).dot(cluster.X)
        for cluster, residual, weights in zip(clusters, residuals, cluster_weights
)
    ], 0)
    return beta, bread.dot(meat).dot(bread)

def update_clusters(clusters, covariance_parameters):
    return [
        Cluster(X=cluster.X, y=cluster.y,
                covariance=covariance_parameters.make_covariance_matrix(len(cluste
r.y)))
        for cluster in clusters]
```

```
In [3]: orthodont_data = pd.read_csv('orthodont.csv')
        orthodont_data = orthodont_data.set_index('Subject')
        orthodont_data.head(8)
```

Out[3]:

Subject	distance	age	Sex
M01	26.0	8	Male
M01	25.0	10	Male
M01	29.0	12	Male
M01	31.0	14	Male
M02	21.5	8	Male
M02	22.5	10	Male
M02	23.0	12	Male
M02	26.5	14	Male

```
In [4]: def make_covariates(data_frame):
            age = (data_frame['age'] - 8).values
            is_female = (data_frame['Sex'] == 'Female').values.astype(np.float64)
            return np.column_stack((
                np.ones(len(data_frame)),
                age,
                is_female,
                age*is_female,
            ))

        def make_response(data_frame):
            return data_frame['distance'].values
```

```
In [5]: def make_table(beta, beta_covariance):
            std_errors = np.sqrt(np.diag(beta_covariance))
            return pd.DataFrame(list(zip(beta, std_errors)),
                                columns=['Estimate', 'Standard error'],
                                index=['$\\beta_0$', '$\\beta_1$', '$\\beta_2$', '$\\beta_
        3$'])
```

Initially, we assume no covariance.

```
In [6]: clusters = [
            Cluster(
                X=make_covariates(orthodont_data.loc[i]),
                y=make_response(orthodont_data.loc[i]),
                covariance=np.eye(len(orthodont_data.loc[i])),
            )
            for i in orthodont_data.index
        ]

        beta, beta_covariance = estimate_beta(clusters)
        with open('beta0_estimate.tex', 'w') as f:
            f.write(make_table(beta, beta_covariance).to_latex(escape=False))
```

In each iteration, we estimate covariance parameters with $\hat{\beta}$. We compute a new correlation structure for each cluster, and use it to get a better estimate for β.

```python
In [7]: def _update_estimates(clusters, beta):
            covariance_parameters = estimate_covariance_parameters(clusters, beta)
            clusters = update_clusters(clusters, covariance_parameters)
            return covariance_parameters, clusters, estimate_beta(clusters)

        previous_beta, (covariance_parameters, clusters, (beta, beta_covariance)) = (
            beta, _update_estimates(clusters, beta))

        with open('beta1_estimate.tex', 'w') as f:
            f.write(make_table(beta, beta_covariance).to_latex(escape=False))
        covariance_parameters
```

Out[7]: CovarianceParameters(alpha=0.6113624114825353, sigma2=4.905158354377104)

```python
In [8]: while np.sum(np.abs(previous_beta - beta)) > 1e-12:
            previous_beta, (covariance_parameters, clusters, (beta, beta_covariance)) = (
                beta, _update_estimates(clusters, beta))
            print(beta, covariance_parameters)
            print(np.sqrt(np.diag(beta_covariance)))
```

```
[22.75026233  0.76945687 -1.55885764 -0.28569215] CovarianceParameters(alpha=0.61
3518833989525, sigma2=4.910599805251229)
[0.26779157 0.04369856 0.41954876 0.06846249]
[22.7502655   0.76945666 -1.55886113 -0.28569188] CovarianceParameters(alpha=0.61
35308146270477, sigma2=4.91065185077109)
[0.2677919  0.04369862 0.41954928 0.06846258]
[22.75026552  0.76945666 -1.55886115 -0.28569188] CovarianceParameters(alpha=0.61
35308813311675, sigma2=4.910652141021696)
[0.2677919  0.04369862 0.41954928 0.06846258]
[22.75026552  0.76945666 -1.55886115 -0.28569188] CovarianceParameters(alpha=0.61
35308817025588, sigma2=4.910652142637751)
[0.2677919  0.04369862 0.41954928 0.06846258]
[22.75026552  0.76945666 -1.55886115 -0.28569188] CovarianceParameters(alpha=0.61
35308817046278, sigma2=4.910652142646753)
[0.2677919  0.04369862 0.41954928 0.06846258]
[22.75026552  0.76945666 -1.55886115 -0.28569188] CovarianceParameters(alpha=0.61
35308817046404, sigma2=4.91065214264681)
[0.2677919  0.04369862 0.41954928 0.06846258]
```

The final, converged estimates are below.

```python
In [9]: with open('beta_final_estimate.tex', 'w') as f:
            f.write(make_table(beta, beta_covariance).to_latex(escape=False))
```