Multivariate Analysis HW5

2020311198 Dongook Son

April 11, 2021

[1]:
```
import numpy as np
import pandas as pd
from scipy.stats import chi2, f
import matplotlib.pyplot as plt
import pprint
try:
    import termplotlib as tpl
except Exception as e:
    print(f"termploblib is not installed.\nUsing matplotlib as default.")
```

1.

Hotelling's test is implemented as a method for the `MultivariateData` object as `profile_analysis`.

[2]:
```
class MultivariateData:
    """
        Object for computing multivariate data

        Attributes:
            data (np.array): input data
            n, (int):
            p (int):
            mean_vector (np.array):
            covariance_matrix (np.array):

        Args:
            inputdata (np.array, list, tuple, ...): any iterable object that␣
↪numpy supports
    """

    def __init__(self, inputdata) -> None:
        self.data = np.array(inputdata)
        self.n, self.p = self.data.shape
        self.mean_vector = np.mean(self.data, axis=0)
        self.covariance_matrix = np.cov(self.data.transpose())
```

```python
    def __sub__(self, other):
        if isinstance(other, np.ndarray):
            return MultivariateData(self.data - other)
        elif isinstance(other, MultivariateData):
            return MultivariateData(self.data - other.data)
        else:
            raise ValueError("Object must be np.array or MultivariateData")

    def __add__(self, other):
        if isinstance(other, np.ndarray):
            return MultivariateData(self.data + other)
        elif isinstance(other, MultivariateData):
            return MultivariateData(self.data + other.data)
        else:
            raise ValueError("Object must be np.array or MultivariateData")

    def __mul__(self, other):
        if isinstance(other, int) or isinstance(other, float):
            return MultivariateData(self.data * other)
        elif isinstance(other, MultivariateData):
            if self.p == other.n:
                return MultivariateData(np.matmul(self.data, other.data))
            else:
                raise ValueError("Dimension does not match.")
        elif isinstance(other, np.ndarray):
            return MultivariateData(np.matmul(self.data, other))
        else:
            raise TypeError("Unsupported operation between types")

    def __repr__(self) -> str:
        return f"MultivariateData(SampleSize:{self.n}, Features:{self.p})"

    def append(self, other, orientation: str = 'h'):
        """Appends MultivariateData in given orientation

        Args:
            other (MultivariateData): Other multivariate object
            orientation (str): 'h' for horizontal, 'v' for vertical
        """
        assert isinstance(other, MultivariateData)
        axis = 1 if orientation == 'v' else 0
        return MultivariateData(np.concatenate((self.data, other.data),␣
↪axis=axis))

    def generalized_squared_distance(self) -> list:
        result = []
        inv_cov = np.linalg.inv(self.covariance_matrix)
```

```
        for row in self.data:
            diff = row - self.mean_vector
            # numpy broadcasting
            result.append(np.matmul(np.matmul(diff, inv_cov), diff))
        assert len(result) == self.n
        return result

    def __get_qq_tuples(self) -> list:
        result = []
        sorted_general_distance = sorted(self.generalized_squared_distance())
        for i, x in enumerate(sorted_general_distance):
            x_probability_value = (i+1 - 0.5) / self.n
            q_value = chi2.ppf(x_probability_value, self.p)
            result.append(
                (q_value, x)
            )
        return result

    def qqplot(self, terminal=False):
        """Draws qqplot for Multivariate Data

        Args:
            terminal (bool, optional): [Option for drawing the qqplot in␣
↪terminal].
            If False -> draws via matplotlib
        """
        qq_tuples = self.__get_qq_tuples()
        x = [x for x, _ in qq_tuples]
        y = [y for _, y in qq_tuples]
        if terminal:
            fig = tpl.figure()
            fig.plot(x, y, width=60, height=20)
            fig.show()
        else:
            plt.scatter(x, y)
            plt.show()

    def hotellings_t_test(self, mu_vector_null, alpha=0.05, method="p"):
        """Performs Hotellings test for mean comparison, via adjusted F␣
↪distribution

        Args:
            mu_vector_null ([int, float]): vector of mean under the null␣
↪hypothesis
            alpha (float, optional): 1-alpha = Significance level. Defaults to 0.
↪05.
```

```
            method (str, optional): Method of testing. Either 'p' or 'critical'.␣
↪Defaults to "p".
        """
        significance = 1-alpha
        assert (isinstance(mu_vector_null, list)
                or isinstance(mu_vector_null, np.ndarray))
        diff = self.mean_vector - mu_vector_null
        if self.p > 1:
            inv_cov = np.linalg.inv(self.covariance_matrix)
            t_2_statistic = self.n * np.matmul(np.matmul(diff, inv_cov), diff)
            critical_value = ((self.n - 1) * self.p)/(self.n-self.p) * \
                f.ppf(significance, self.p, self.n - self.p)
            f_statistic = ((self.n - self.p) * t_2_statistic) / \
                ((self.n-1) * self.p)
            p_value = 1 - f.cdf(f_statistic, self.p, self.n - self.p)
            print(f"---------------------HOTELLING'S T^2␣
↪TEST----------------------")
            print(
                f"Null Hypothesis:\n  Mean vector {self.mean_vector}\n  is equal␣
↪to {np.array(mu_vector_null)}")
            print(f"Distribution: F{(self.p, self.n-self.p)}")
            print(f"F statistic: {f_statistic}")
            print(f"t^2 statistic: {t_2_statistic}")
        else:
            print(f"---------------------      F TEST       ␣
↪----------------------")
            cov = self.covariance_matrix.max()
            x_bar = diff.max()
            mu = mu_vector_null[0]
            n = self.n
            print(
                f"Null Hypothesis:\n  Mean {x_bar}\n  is equal to {mu}")
            t_statistic = (x_bar - mu) / (np.sqrt(cov / n))
            f_statistic = t_statistic ** 2
            p_value = 1 - f.cdf(f_statistic, 1, self.n - 1)
            print(f"Distribution: F({(1, self.n - 1)})")
            print(f"F statistic: {f_statistic}")
        print(f"Significance: {significance*100}%")
        if method == 'p':
            print(f"P-value: {p_value}")
        elif method == 'critical':
            print(
                f"Critical Value: {critical_value}")

        if p_value < alpha:
            print(f"Conclusion: REJECT the null hypothesis")
        else:
```

```
            print(f"Conclusion: DO NOT reject the null hypothesis")
        print(f"---------------------------------------------------------------")

    def confidence_ellipsoid_info(self, alpha=0.05) -> dict:
        """Calculates the axis and the length of the ellipsoide of the␣
↪multivariate data.

        Args:
            significance (float, optional): [Level of significance]. Defaults to␣
↪0.05.

        Returns:
            dict: integer keys will be the axes in the descending order. Each␣
↪key has two keys("axis", "length")
                  axis denotes the direction of the ellipsoide
                  length denotes the length of the axis.
        """
        result = {}
        significance = 1-alpha
        eigenvalues, eigenvectors = np.linalg.eig(self.covariance_matrix)
        for i, v in enumerate(eigenvalues):
            conf_half_len = np.sqrt(v) * np.sqrt((self.n - 1) * self.p * f.ppf(
                significance, self.p, self.n - self.p) / (self.n * (self.n -␣
↪self.p)))
            conf_axe_abs = conf_half_len * eigenvectors[i]
            result[i] = {
                "axis": (conf_axe_abs, -conf_axe_abs),
                "length": conf_half_len * 2
            }
        return result

    def simultaneous_confidence_interval(self, vector, alpha=0.05,␣
↪large_sample=False) -> tuple:
        """Calculates the simultaneous confidence interval given a␣
↪transformation vector and a significance level.
             The default method would be not assuming the data as a large sample.

        Args:
           vector (list or ndarray): [The transformation vector].
           significance (float, optional): [Level of significance]. Defaults to␣
↪0.05.
           large_sample (bool, optional): [Use large sample assumptions].␣
↪Defaults to False.

        Returns:
           tuple: (lowerbound: float, upperbound: float)
```

```
        """
        significance = 1-alpha
        assert len(vector) == self.p
        if not isinstance(vector, np.ndarray):
            vec = np.array(vector)
        else:
            vec = vector
        if not large_sample:
            conf_width = np.sqrt(
                self.p * (self.n - 1) * f.ppf(significance, self.p, self.n -␣
↪self.p) * vec.dot(self.covariance_matrix).dot(vec) / (self.n * (self.n - self.
↪p)))
            t_mean = vec.dot(self.mean_vector)
            return (t_mean - conf_width, t_mean + conf_width)
        else:
            conf_width = np.sqrt(chi2.ppf(significance, self.p) *
                                 vec.dot(self.covariance_matrix).dot(vec)/self.n)
            t_mean = vec.dot(self.mean_vector)
            return (t_mean - conf_width, t_mean + conf_width)

    def profile_analysis(self, flat=True, c_matrix=None, alpha=0.05, method="p"):
        if flat:
            c_matrix = self.__flat_c_matrix()
            transformed_data = MultivariateData(
                np.matmul(c_matrix, self.data.T).T)
            transformed_data.hotellings_t_test(
                np.zeros(transformed_data.p), alpha, method)
        else:
            assert c_matrix is not None, "If not flat, c_matrix is required."
            c_mat = np.array(c_matrix)
            try:
                _, c_mat_n_col = c_mat.shape
            except Exception as e:
                if isinstance(e, ValueError) & (e.args[0] == 'not enough values␣
↪to unpack (expected 2, got 1)'):
                    _, c_mat_n_col = (len(c_mat), 1)
            transformed_array = np.matmul(c_mat, self.data.T)
            # transformed_data = np.reshape(transformed_array ,␣
↪(len(transformed_array),c_mat_n_col))
            transformed_data = transformed_array.T
            transformed_multivar_data = MultivariateData(transformed_data)
            transformed_multivar_data.hotellings_t_test(
                [0]*len(c_mat), alpha, method)
        return

    def __flat_c_matrix(self):
        minus_identity_matrix = -np.identity(self.p)
```

```
        col_ones = np.ones((self.p, 1))
        return np.hstack((col_ones, minus_identity_matrix))[:self.p-1, :self.p]
```

2.

a.

[4]:
```
stiff_df = pd.read_csv(
    'stiff.DAT',
    header=None,
    index_col=False,
    delim_whitespace=True)
stiff_df.columns = ['x1', 'x2', 'x3', 'x4', 'd2']
stiff = MultivariateData(stiff_df.iloc[:, 0:4])
```

[5]:
```
stiff.profile_analysis(flat=True, alpha=0.05)
```

```
---------------------HOTELLING'S T^2 TEST----------------------
Null Hypothesis:
  Mean vector [156.56666667 396.96666667 181.13333333]
  is equal to [0. 0. 0.]
Distribution: F(3, 27)
F statistic: 79.0514087513837
t^2 statistic: 254.72120597668084
Significance: 95.0%
P-value: 1.7219559111936178e-13
Conclusion: REJECT the null hypothesis
---------------------------------------------------------------
```

b.

[6]:
```
c_mat = np.array(
    [
        [1, -2, 1, 0],
        [0, 1, -2, 1],
    ]
)
stiff.profile_analysis(flat=False, c_matrix=c_mat)
```

```
---------------------HOTELLING'S T^2 TEST----------------------
Null Hypothesis:
  Mean vector [-83.83333333 456.23333333]
  is equal to [0 0]
Distribution: F(2, 28)
F statistic: 87.08000779315766
t^2 statistic: 180.38001614296945
Significance: 95.0%
P-value: 9.560130465047223e-13
```

```
Conclusion: REJECT the null hypothesis
----------------------------------------------------------------
```

c.

[7]:
```python
from statsmodels.stats import multivariate as mv
print(mv.test_mvmean(
    pd.concat([
        stiff_df['x1'] - stiff_df['x2'],
        stiff_df['x2'] - stiff_df['x3'],
        stiff_df['x3'] - stiff_df['x4'],
    ], axis=1)
))
```

```
statistic = 79.05140875138369
pvalue = 1.721915017054007e-13
df = (3, 27)
t2 = 254.72120597668078
distr = F
tuple = (79.05140875138369, 1.721915017054007e-13)
```

[8]:
```python
print(mv.test_mvmean(
    pd.concat([
        stiff_df['x1'] - 2*stiff_df['x2'] + stiff_df['x3'],
        stiff_df['x2'] - 2*stiff_df['x3'] + stiff_df['x4'],
    ], axis=1)
))
```

```
statistic = 87.08000779315768
pvalue = 9.560459481929198e-13
df = (2, 28)
t2 = 180.38001614296948
distr = F
tuple = (87.08000779315768, 9.560459481929198e-13)
```

3.

a.

[9]:
```python
lumber_df = pd.read_csv(
    'lumber.dat',
    header=None,
    index_col=False,
    delim_whitespace=True)
lumber_df.columns = ['x1', 'x2']
lumber_sample_first = lumber_df[:15]
lumber_sample_second = lumber_df[15:]
lumber = MultivariateData(lumber_df)
```

```
lumber_s1 = MultivariateData(lumber_sample_first)
lumber_s2 = MultivariateData(lumber_sample_second)
lumber_subtracted = lumber_s1 - lumber_s2
lumber_subtracted.hotellings_t_test([0,0])
```

```
---------------------HOTELLING'S T^2 TEST----------------------
Null Hypothesis:
  Mean vector [-193.4         -48.26666667]
  is equal to [0 0]
Distribution: F(2, 13)
F statistic: 1.269778799536756
t^2 statistic: 2.7349081836176286
Significance: 95.0%
P-value: 0.3135310532015405
Conclusion: DO NOT reject the null hypothesis
---------------------------------------------------------------
```

b.

[10]:
```
pprint.pprint(lumber_subtracted.simultaneous_confidence_interval([1, 0]))
```

```
(-546.0898558358173, 159.2898558358173)
```

c.

[11]:
```
c_mat = np.array([
    [1, 0, -1, 0],
    [0, 1, 0, -1],
])
lumber_concat = lumber_s1.append(lumber_s2, 'v')
lumber_concat.profile_analysis(flat=False, c_matrix=c_mat)
```

```
---------------------HOTELLING'S T^2 TEST----------------------
Null Hypothesis:
  Mean vector [-193.4         -48.26666667]
  is equal to [0 0]
Distribution: F(2, 13)
F statistic: 1.269778799536756
t^2 statistic: 2.7349081836176286
Significance: 95.0%
P-value: 0.3135310532015405
Conclusion: DO NOT reject the null hypothesis
---------------------------------------------------------------
```