Multivariate Analysis HW4

2020311198 Dongook Son

April 3, 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.

> Write a Python code to implement Hotelling's T2 test of a mean vector. (use p-value for testing).

I created a data class(`MultivariateData`) in order to implement various analysis methods for further homeworks.
Hotelling's test is implemented as a method for the object as `hotellings_t_test`.

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

    params
    ------
    src: input matrix of shape (n:int, p:int) where n is the sample size and p␣
↪is the number of dependent variables
        np.ndarray

    methods
    -------
    _get_mean_vector: gets the mean vector along the features
        np.array

    _get_cov_mat: gets the covariance matrix in (p by p).
        np.array

    _generalized_squared_distance: gets list of squared distance by dimension n.
```

```
        list

    _get_qq_tuples: gets the list of tuples of the qq pair for the chisquare␣
↪distribution with df=p
        list

    draw_qqplot: draws the plot by using matplotlib

    hotellings_t_test: performs a Hotelling's T^2 test with a given mu vector␣
↪and significance level
        float

    confidence_ellipsoid_info: Calculates the axis and the length of the␣
↪ellipsoide of the multivariate data given a significance level.
        dict

    simultaneous_confidence_interval: Calculates the simultaneous confidence␣
↪interval given a transformation vector
        tuple
    """

    def __init__(self, src) -> None:
        self.src = self._numpy_coersion(src)
        self.n, self.p = self.src.shape
        self.mean_vector = self._get_mean_vector()
        self.cov_matrix = self._get_cov_mat()

    @staticmethod
    def _numpy_coersion(data) -> np.array:
        # coerce pandas or other iterative data to numpy array.
        if not isinstance(data, np.ndarray):
            try:
                result = np.array(data)
            except Exception as e:
                print(
                    f"{data.__class__} cannot be coerced to Numpy array!\nERROR:
↪{e}")
            return result
        else:
            return data

    def _get_mean_vector(self) -> np.array:
        return (np.mean(self.src, axis=0))

    def _get_cov_mat(self) -> np.array:
        result = np.cov(self.src.transpose())
        assert result.shape == (self.p, self.p)
```

```
        return result

    def _generalized_squared_distance(self) -> list:
        result = []
        inv_cov = np.linalg.inv(self.cov_matrix)
        for row in self.src:
            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 draw_qqplot(self, terminal=False):
        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, significance=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
            significance (float, optional): Significance level. Defaults to 0.05.
            method (str, optional): Method of testing. Either 'p' or 'critical'.␣
↪Defaults to "p".
        """
        assert (isinstance(mu_vector_null, list)
```

```
                or isinstance(mu_vector_null, np.ndarray))
        assert (0 < significance < 1)
        inv_cov = np.linalg.inv(self.cov_matrix)
        diff = self.mean_vector - mu_vector_null
        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"T^2 statistic: {t_2_statistic}")
        print(f"F statistic: {f_statistic}")

        print(f"Significance(alpha): {significance}")

        if method == 'p':
            print(f"P-value: {p_value}")
        elif method == 'critical':
            print(
                f"Critical Value: {critical_value}")

        if p_value < significance:
            print(f"Conclusion: REJECT the null hypothesis")
        else:
            print(f"Conclusion: DO NOT reject the null hypothesis")
        print(f"---------------------------------------------------------------")

    def confidence_ellipsoid_info(self, significance=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 = {}
        eigenvalues, eigenvectors = np.linalg.eig(self.cov_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, significance=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)
        """
        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.cov_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.cov_matrix).dot(vec)/self.n)
            t_mean = vec.dot(self.mean_vector)
            return (t_mean - conf_width, t_mean + conf_width)
```

2.

a.

Testing `college.DAT` with the implemented function above.

[3]:
```
college_dat = pd.read_csv("college.DAT", delim_whitespace=True, header=None)
college_dat.columns = ["ssh", "vrbl", "sci"]
cd = MultivariateData(college_dat)
null_hypothesis_mean_vector = [500, 50, 30]
cd.hotellings_t_test(null_hypothesis_mean_vector, significance=0.05)
```

```
---------------------HOTELLING'S T^2 TEST----------------------
Null Hypothesis:
  Mean vector [526.5862069   54.68965517  25.12643678]
  is equal to [500  50  30]
T^2 statistic: 223.3101756848916
F statistic: 72.70563859508097
Significance(alpha): 0.05
P-value: 1.1102230246251565e-16
Conclusion: REJECT the null hypothesis
---------------------------------------------------------------
```

2.

b.

[4]:
```
from statsmodels.stats import multivariate as mv
```

[5]:
```
mv.test_mvmean(cd.src - null_hypothesis_mean_vector)
```

[5]:
```
<class 'statsmodels.stats.base.HolderTuple'>
statistic = 72.70563859508101
pvalue = 2.828097062464791e-23
df = (3, 84)
t2 = 223.3101756848917
distr = F
tuple = (72.70563859508101, 2.828097062464791e-23)
```

The result derived from the `statsmodel` package is consistent with my custom function. Note that even though the pvalue seems different, in reality it is not. This is due to the fact that computer software cannot accurately represent `float` this small.

2.

c.

[6]:
```
pprint.pprint(cd.confidence_ellipsoid_info(significance=0.05))
```

```
{0: {'axis': (array([-4.89424684, -0.51080087, -0.18371125]),
              array([4.89424684, 0.51080087, 0.18371125])),
     'length': 9.848516527874517},
 1: {'axis': (array([-0.0530798 ,  0.51035257, -0.00491465]),
              array([ 0.0530798 , -0.51035257,  0.00491465])),
     'length': 1.0262579964587029},
 2: {'axis': (array([-0.00934886,  0.00138893,  0.24520082]),
              array([ 0.00934886, -0.00138893, -0.24520082])),
     'length': 0.49076582646381417}}
```

2.

d.

Find the simultaneous confidence interval for $\mu_1 - 2\mu_2 + \mu_3$.

[7]:
```
print(cd.simultaneous_confidence_interval([1, -2, 1]))
```

```
(438.1245590084046, 446.5421076582621)
```

3.

a.

Find the simultaneous confidence interval for $\mu_1 + 2\mu_2 - \mu_3 - 2\mu_4$.

[8]:
```
# import data
stiff = pd.read_csv('stiff.DAT',
                    header=None, delim_whitespace=True)
stiff.columns = ['x1', 'x2', 'x3', 'x4', 'x5']

stf = MultivariateData(stiff)
print(stf.simultaneous_confidence_interval([1, 2, -1, -2, 0]))
```

```
(341.5131034172878, 550.6868965827111)
```

b.

Repeating the above step with large sample assumption.

[9]:
```
print(stf.simultaneous_confidence_interval(
    [1, 2, -1, -2, 0], large_sample=True))
```

```
(347.2745304989698, 544.925469501029)
```

We can observe that the confidence interval slightly shortened compared to the above result.