IFT 6758 Assignment-3

October 21, 2019

0.0.1 Classical Inference

2. [ISLR 3.7.3] Suppose we have a dataset with five predictors, $X_1 = $ GPA, $X_2 = IQ$, $X_3 = $ Gender (1 for Female and 0 for Male), $X_4 = $ Interaction between GPA and IQ, and $X_5 = $ Interaction between GPA and Gender. The response is starting salary after graduation (in thousands of dollars). Suppose we use least squares to fit the model, and get $\hat{\beta}_0 = 50, \hat{\beta}_1 = 20, \hat{\beta}_2 = 0.07, \hat{\beta}_3 = 35, \hat{\beta}_4 = 0.01, \hat{\beta}_5 = -10$.

a. Which is correct, and why? i. For a fixed value of IQ and GPA, males earn more on average than females. ii. For a fixed value of IQ and GPA, females earn more on average than males. iii. For a fixed value of IQ and GPA, males earn more on average than females provided that the GPA is high enough. iv. For a fixed value of IQ and GPA, females earn more on average than males provided that the GPA is high enough.

b. Predict the salary of a female with IQ of 110 and a GPA of 4.0.

c. True or false: Since the coefficient for the GPA / IQ interaction term is very small, there is very little evidence of an interaction effect. Justify your answer.

Answer 2a:
Using linear regression:
Males:
= 50 + 20(GPA) + 0.07(IQ) + 35(0) + 0.01(GPA x IQ) -10(GPA x 0)
= 50 + **20.01(GPA)** + 0.07(IQ) + 0.01(GPA x IQ) Females:
= 50 + 20(GPA) + 0.07(IQ) + 35(1) + 0.01(GPA x IQ) -10(GPA x 1)
= 50 + **10.01(GPA)** + 0.07(IQ) + 0.01(GPA x IQ) + 35
If GPA > 3.5, the salary of Males will be higher.
Therefore answer is **3** For a fixed value of IQ and GPA, males earn more on average than females provided that the GPA is high enough.'

Answer 2b:
Salary of Female with GPA = 4 & IQ = 110:
= 50 + 20(4) + 0.07(110) + 35(1) + 0.01(440) - 10(4)
= 130 + 7.7 + 35 + 4.4 - 40 = 137.1 thousands dollar

Answer 2c:
False, even though the interaction terms is very small, we dont know the significance without doing a t test and checking the p-value for the hypothesis that B4 will be zero

0.0.2 The Bootstrap

6. Suppose that $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$ are two independent samples. As a measure of the difference in location of the two samples, the difference of the 20% trimmed means is used (each trimmed mean is the mean after discarding the 10% smallest and 10% largest values in the group). Explain how the bootstrap could be used to estimate the standard error of this difference.

Answer 6:
Although there is method/formula for calculating the standard error of difference in mean, but I'm not aware of any formula for standard error of difference in trimmed means. Therefore, bootstrap is a convenient way to estimate this error without the need to plug in formulaes.

Algorithm to calculate standard error for difference in trimmed mean using bootstrap:

Step 1:
Take a bootstrap sample with replacement for both X and Y.

Step 2:
Calculate the trimmed mean for both the samples and store the difference between those means.

Step 3:
Repeat this process 10000 times and collect all the trimmed means difference

Step 4:
Find the Standard Deviation for all the mean differences and calculate the standard deviation.

Step 5:
This standard deviation calculated above will give the estimate of the standard error of difference in trimmed means.

I have written a code below to illustrate all these steps

[1]:
```
# from sklearn.utils import resample
# from scipy import stats
# trimmed_mean = []
# for i in range(1000):
#     boot_X = resample(X, replace=True, n_samples=len(X))
#     boot_Y = resample(Y, replace=True, n_samples=len(Y))

#     trimmed_mean_boot_X = stats.trim_mean(boot_X, 0.2)
#     trimmed_mean_boot_Y = stats.trim_mean(boot_Y, 0.2)
#     diff_trimmed_mean = trimmed_mean_boot_X - trimmed_mean_boot_Y
#     trimmed_mean.append(trimmed_mean_boot_X - trimmed_mean_boot_X)

# import numpy as np
# np.std(np.array(trimmed_mean))
```

7. [ISLR 4.5.9] Consider the Boston housing dataset.

a. Based on this dataset, provide an estimate for the population mean of `medv`. Call this estimate $\hat{\mu}$.

b. Provide an estimate of the standard error of $\hat{\mu}$. Interpret this result. *Hint: We can compute the standard error of the sample mean by dividing the sample standard deviation by the square root of the number of observations..*

c. Now estimate the standard error $\hat{\mu}$ using the bootstrap. How does this compare to your answer from (b)?

d. Based on your bootstrap estimate from (c), provide a 95% confidence interval for the mean of `medv`. *Hint: You can approximate a 95% confidence interval using the formula* $[\hat{\mu} - 2s.e.(\hat{\mu}), \hat{\mu} + 2s.e.(\hat{\mu})]$.

e. Based on this dataset, provide an estimate $\hat{\mu}_{med}$ for the median value of `medv` in the population.

f. We now would like to estimate the standard error of $\hat{\mu}_{med}$. Unfortunately, there is no simple formula for computing the standard error of the median. Instead, estimate the standard error of the median using the bootstrap. Comment on your findings.

g. Based on this dataset, provide an estimate for the 10th percentile of `medv` in Boston suburbs. Call this quantity $\hat{\mu}_{0.1}$.

h. Use the bootstrap to estiamte the standard error of $\hat{\mu}_{0.1}$. Comment on your findings.

[20]:
```
import pandas as pd
import numpy as np
df = pd.read_csv("https://gist.githubusercontent.com/krisrs1128/
↪2c1ce8d004b1efc18b2d6e03e84a27c6/raw/
↪9e95c9782b46f1ede7c288466390c8f937aecc08/boston.csv")
```

Answer 7a

[3]:
```
df['medv'].mean()
```

[3]:
```
22.532806324110698
```

Answer 7b

As explained in the question, the formula for standard error is the standard deviation divided by square root of the total samples

[6]:
```
length_of_samples = len(df['medv'])
std_deviation = df['medv'].std()
sem = std_deviation/np.sqrt(length_of_samples)
sem
```

[6]:
```
0.4088611474975351
```

Answer 7c

Taking a bootstrap sample over 10000 times, and calculating the mean everytime. The standard deviation of this list of means will give us the standard error of the mean. And in this case it turns out to be very similar to the one we calculated above using a formula i.e. 0.40

[7]:
```
from sklearn.utils import resample
mean = []
for i in range(10000):
    boot = resample(df['medv'], replace=True, n_samples=len(df['medv']))
    mean.append(boot.mean())

import numpy as np
np.std(np.array(mean))
```

[7]:
```
0.4064960176946766
```

Answer 7d
Standard Error caluclated above = 0.40
95% confidence interval is (mean-2(s.e.m), mean+2(s.e.m))
(22.53 - 2 x 0.40, 22.53 + 2 x 0.40)
(21.73, 23.33)
Answer 7e

[8]:
```
df['medv'].median()
```

[8]:
```
21.2
```

Answer 7f
Since we dont have any formula for calculating standard error in median, we can do the same thing as above using a bootstrap and calculating the standard deviation of all medians.

[9]:
```
from sklearn.utils import resample
median = []
for i in range(10000):
    boot = resample(df['medv'], replace=True, n_samples=len(df['medv']))
    median.append(boot.median())

import numpy as np
np.std(np.array(median))
```

[9]:
```
0.37840361507258324
```

Answer 7g

[10]:
```
df['medv'].quantile(.1)
```

[10]:
```
12.75
```

Answer 7h
We can calculate the standard error of quantile similarily by bootstrapping. Quantile standard error = 0.50

[11]:
```
from sklearn.utils import resample
quantiles = []
for i in range(10000):
    boot = resample(df['medv'], replace=True, n_samples=len(df['medv']))
    quantiles.append(boot.quantile(.1))

import numpy as np
np.std(np.array(quantiles))
```

[11]:
```
0.5018735494125985
```

0.0.3 Large-Scale Inference

13. The data here are a simulation of 10,000 experiments, each seeking to detecting a difference between treatment and control. Only the last 10% of hypotheses are actually nonnull (ids 9001 to 10000).

a. For experiment, run a t-test comparing treatment and control, using an $\alpha = 0.05$ significance level. How many false positives (rejected hypotheses among the null IDs) do you find? What is the FDR in this instance (the fraction $\frac{V}{R}$)?

b. Apply a Bonferroni correction to all p-values. How many false positives do you find? What is the false discovery rate?

c. Apply the Benjamini-Hochberg procedure. What is the false discovery rate?

Answer 13

[12]:
```python
import pandas as pd
df_13 = pd.read_csv("https://gist.githubusercontent.com/krisrs1128/
↪ff7b6498c89316b9dd526a0f44d92d31/raw/
↪2683592cd4ed08d00e17d12adbc90a6bafac434c/experiments.csv")
```

Answer 13a

Calculating t-test for each experiment while comparing treatment and control using $\alpha = 0.05$. Assuming that the samples are independent as no such information is provided. Has they been dependent samples, we would have conducted paired/dependent t-test. In that case we would get 436 False Positives and the FDR would have been 0.65

Continuing with the assumption that these are independent samples..

False Positive: 411

R = V + S = 411 + 292 (Also called as discoveries)

False Discovery Rate: $\frac{FalsePositive}{FalsePositive+TruePositive}$

FDR = $\frac{411}{411+292}$

FDR = 0.58

[21]:
```python
def calculate_p(control_values,treatment_values):
    from scipy.stats import t
    mean1, mean2 = control_values.mean(), treatment_values.mean()
    se1, se2 = control_values.sem(), treatment_values.sem()
    sed = np.sqrt(se1**2 + se2**2)
    t_stat = (mean1-mean2)/sed
    df = len(control_values) + len(treatment_values) - 2
    p = (1 - t.cdf(abs(t_stat), df))*2

def calculate_p_values(data):
    from scipy.stats import ttest_ind, ttest_rel
    p_values = []
    for i in range(1,10001):
        experiment_rows = data[data['experiment_id']==i]
        control_values =␣
↪experiment_rows[(experiment_rows['type']=="control")]['value']
        treatment_values =␣
↪experiment_rows[(experiment_rows['type']=="treatment")]['value']
#         p = calculate_p(control_values, treatment_values)
        s, p = ttest_rel(control_values, treatment_values)
        p_values.append(p)
    return p_values
```

```
def calculate_errors(df, alpha = 0.05):
    V = sum(df[df.h_null==1]['pval'] <= 0.05) # False Positives - Type 1 Errors␣
 ↪- (Rejected among Null)
    U = sum(df[df.h_null==1]['pval'] > 0.05) # True Negatives
    S = sum(df[df.h_null==0]['pval'] <= 0.05) # True Positives
    T = sum(df[df.h_null==0]['pval'] > 0.05) # False Negatives - Type 2 Errors␣
 ↪-
    return V,U,S,T

p_values = calculate_p_values(df_13)
df_pval = pd.DataFrame({"pval":p_values})
df_pval['h_null'] = np.where(df_pval.index < 9000, 1, 0)

V, U, S, T = calculate_errors(df_pval)
FDR = V/(V+S)
print("False Positives: {}".format(V))
print("True Positives: {}".format(S))
print("False Negatives: {}".format(T))
print("True Negatives: {}".format(U))
print("")
print("FDR is {}".format(FDR))
```

```
False Positives: 436
True Positives: 227
False Negatives: 773
True Negatives: 8564

FDR is 0.6576168929110106
```

Answer 13b

Bonferroni correction is used to decrease the Type 1 Errors by increasing the significant level. So significant level changes from α to $\frac{\alpha}{m}$. We can also make changes to the p-values by multiplying them with m and keeping the same significant level as α

The new adjusted p-values are increased to a level that there are no more false positives and the FDR is 0

I tried implementing the algorithm and compared my results with a library and the results seems to be the same.

[16]:
```
## Implementation

m = 10000 # Since we have 10,000 hypothesis tests
adjusted_p_values = [i*m for i in p_values]

df_pval_adj = pd.DataFrame({"pval":adjusted_p_values})
df_pval_adj['h_null'] = np.where(df_pval_adj.index < 9000, 1, 0)
V_c, U_c, S_c, T_c = calculate_errors(df_pval_adj)
FDR_c = V_c/(V_c+S_c)
print("False Positives after correction: {}".format(V_c))
print("True Positives after correction: {}".format(S_c))
print("False Negatives after correction: {}".format(T_c))
print("True Negatives after correction: {}".format(U_c))
print("")
print("FDR after correction is {}".format(FDR_c))
```

```
False Positives after correction: 0
True Positives after correction: 1
False Negatives after correction: 999
True Negatives after correction: 9000

FDR after correction is 0.0
```

[17]:
```
# Using Library

from statsmodels.stats.multitest import multipletests
results,adjusted_p_val,c,d = multipletests(p_values, method="bonferroni", alpha␣
 ↪= 0.05)

S, V = 0, 0
for i,result in enumerate(results):
    if result:
        if i < 9000:
            V += 1
        else:
            S += 1

FDR = V/(V+S)
print("False Positives after correction: {}".format(V))
print("True Positives after correction: {}".format(S))
print("")
print("FDR after correction is {}".format(FDR))
```

```
False Positives after correction: 0
True Positives after correction: 1

FDR after correction is 0.0
```

Answer 13c

Benjamini-Hochberg procedure is used to reduce the type 1 errors by controlling the FDR. I've tried the implementation and the library and came to the same conclusion that there are 0 False Positives.

Procedure

Sorted all the p_values in ascending order and starting from the largest p_value. Multiplied it by the (total number of p values) / (rank of the p value) and inserted the minimum value between the previous value and the calculated Bh

[18]:
```
# Implementation

def adjust_p_values_bh(p_values):
    p_values_copy = list(p_values)
    p_values_copy.sort()
    total_p_values = len(p_values_copy)
    for i in reversed(range(total_p_values-1)):
        rank = i + 1
        new_p_value = p_values_copy[i]*(total_p_values/rank)
        p_values_copy[i] = min(new_p_value, p_values_copy[i+1])
    return p_values_copy


adjusted_p_values_bh = adjust_p_values_bh(p_values)
df_pval_adj_bh = pd.DataFrame({"pval_old":p_values})

df_pval_adj_bh['h_null'] = np.where(df_pval_adj_bh.index < 9000, 1, 0)
df_pval_adj_bh = df_pval_adj_bh.sort_values(by='pval_old', ascending=True)

df_pval_adj_bh['pval'] = adjusted_p_values_bh
V_c_bh, U_c_bh, S_c_bh, T_c_bh = calculate_errors(df_pval_adj_bh)
FDR_c_bh = V_c_bh/(V_c_bh+S_c_bh)

print("False Positives after correction: {}".format(V_c_bh))
print("True Positives after correction: {}".format(S_c_bh))
print("False Negatives after correction: {}".format(T_c_bh))
print("True Negatives after correction: {}".format(U_c_bh))
print("")
print("FDR after correction is {}".format(FDR_c_bh))
```

```
False Positives after correction: 0
True Positives after correction: 1
False Negatives after correction: 999
True Negatives after correction: 9000

FDR after correction is 0.0
```

[19]:
```
# Using Library

from statsmodels.stats.multitest import multipletests
results,adjusted_p_val,c,d = multipletests(p_values, method="fdr_bh", alpha = 0.
 ↪05)

S, V = 0, 0
for i,result in enumerate(results):
    if result:
        if i < 9000:
            V += 1
        else:
            S += 1

FDR = V/(V+S)
print("False Positives after correction: {}".format(V))
print("True Positives after correction: {}".format(S))
print("")
print("FDR after correction is {}".format(FDR))
```

```
False Positives after correction: 0
True Positives after correction: 1

FDR after correction is 0.0
```

Clustering

2. [ISLR 10.7.4] Suppose that for a particular data set, we perform hierarchical clustering using single inkage and using complete linkage. We obtain two dendrograms.

a. At a certain point on the single linkage dendrogram, the clusters $\{1,2,3\}$ and $\{4,5\}$ fuse. On the complete linkage dendrogram, the clusters $\{1,2,3\}$ and $\{4,5\}$ also fuse at a certain point. Which fusion will occur higher on the tree, or will they fuse at the same height, or is there not enough information to tell?

b. At a certain point on the single linkage dendrogram, the clusters $\{5\}$ and $\{6\}$ fuse. On the complete linkage dendrogram, the clusters $\{5\}$ and $\{6\}$ also fuse at a certain point. Which fusion will occur higher on the tree, or will they fuse at the same height, or is there not enough information to tell?

Answer 2a

Single Linkage (Minimal intercluster dissimilarity):
Computes all pair-wise dissimilarities between points in cluster A and cluster B, and records the smallest dissimilarity.

Complete Linkage (Minimal intercluster dissimilarity):
Computes all pair-wise dissimilarities between points in cluster A and cluster B, and records the largest dissimilarity.

So if two clusters were to fuse togther using these two linkages, there can be two cases:
Min_distance_btw_points_cluster < Max_distance_btw_points_cluster – Case 1
Min_distance_btw_points_cluster = Max_distance_btw_points_cluster – Case 2

Case 2 can be removed as if the min and max distances are same, which means all the points were equidistant and they would have all been under one cluster.

So based on Case 1: The fusion point of Complete linkage will be higher than that of Single linkage as max>min.

Answer 2b

Even in this case we can have those cases, but assuming we have only one point each in both the cluster, we can ignore Case 1 above which leaves us with: min_distance = max_distance.

Therefore, they will fuse at the same height.

Dimensionality Reduction

4. Consider the numbers $\lambda_1 = \|z_1\|_2, \ldots, \lambda_p = \|z_p\|_2$, giving the proportion of variance explained, as in equation ISLR (10.7). Define the statistics $\sum_{k=1}^{p} \left(\lambda_k - \bar{\lambda}\right)^2$, where $\bar{\lambda}$ is the average of all the $\lambda_1, \ldots, \lambda_p$. Discuss the relative usefulness of dimensionality reduction when this statistic is large vs. small.

Answer 4:

As stated λ_1, λ_2, .. λ_p are the proportion of variance explained (PVE) which is the ratio of variance of principal component and the total variance.

The statistics $\sum_{k=1}^{p} \left(\lambda_k - \bar{\lambda}\right)^2$ is like a variance of all the PVEs. If is more varied, it means that few principal component have more variance and are sufficient to explain the data.

If this statistics is low, it means that the PVEs are located near the mean and are less varied which makes our dimensionality reduction less useful as there is no one principal component which can significantly explain our variance. Whereas if this is high, which makes our PVEs more varied and thus more variation can be explained by first or second principal component indicating that the dimensionality reduction was useful in capturing most of data variation in few PCs.