EEE482 - Computational Neuroscience

Homework Assignment - 3

Ahmet Ali Nuhoğlu
21602149

Ahmet Ali NUHOĞLU
21602149

Homework 3

EEE482

Question 1

In this first question we are provided **Blood-oxygen level dependent** (BOLD) responses from a humans visual cortex. In the given data **Yn** holds 1000 responses to the inputs, **Xn**. Every **Xn** has 100 features.

a) We are expected to use the **ridge regression** method to fit our regularized linear model to predict the responses as a weighted sum of given features. Therefore, conduct a 10-fold cross-validation to tune our ridge parameter λ.

While conducting the ridge regression, our main purpose will be minimizing the loss function. Loss function of ridge regression is an extended version of the **Ordinary Least Squares** method. In ridge regressions loss function we have the extra term $\frac{\lambda}{2}||w||_2^2$. Loss function of ridge regression, as given in the lecture slides, is as follows.

$$\frac{1}{2}\|Xw - y\|_2^2 + \epsilon\|w\|_2^2$$

Since we are looking for the w values which minimizes this loss function while avoiding over-fitting. We will solve this function for w. Thus, we obtain:

$$(X^T X + \lambda I)w = X^T y$$

$$w = (X^T X + \lambda I)^{-1} X^T \cdot y$$

With respect to the obtained equation the following function is written to be used as ridge regression method to find optimal weight vector.

```python
def ridge_regression(X, y, lmbd):
    return np.linalg.inv(X.T.dot(X) + lmbd * np.identity(np.
 ↪shape(X)[1])).dot(X.T).dot(y)
```

Therefore, to tune the parameter λ we are going to implement a cross-validation formula with respect to the instructions provided in the assignment. We will take 100 contiguous samples for the validation set, 100 contiguous samples for the testing set, remaining 800 samples will be used as training set. Then, separate models will be fitted for each λ. Also, we are provided that $\lambda \in [0, 10^{12}$. We will take our λ values accordingly while conducting cross-validation. Corresponding codes are given below.

```python
def cross_validation(X, y, K, lmbd):

    part_len = int(np.size(y)/K)
    valid_means_d = dict()
    test_means_d = dict()

    for i in range(K):
        valid_data_start = i * part_len
        test_data_start = (i+1) * part_len
        train_data_start = (i+2) * part_len

        train_data_ind, test_data_ind, valid_data_ind = [], [], []

        for j in range(valid_data_start, test_data_start):
            valid_data_ind.append(j % np.size(y))

        for j in range(test_data_start, train_data_start):
            test_data_ind.append(j % np.size(y))

        for j in range(train_data_start, valid_data_start + np.
↪size(y)):
            train_data_ind.append(j % np.size(y))

        x_valid, x_test, x_train = X[valid_data_ind],␣
↪X[test_data_ind], X[train_data_ind]
        y_valid, y_test, y_train = y[valid_data_ind],␣
↪y[test_data_ind], y[train_data_ind]

        for l in lmbd:
            weight = ridge_regression(x_train, y_train, l)

            valid_means_d.setdefault(l, []).
↪append(r_squared(y_valid, x_valid.dot(weight)))
            test_means_d.setdefault(l, []).
↪append(r_squared(y_test, x_test.dot(weight)))

    valid_means_d = dict((lmbd, np.mean(val)) for lmbd, val in␣
↪valid_means_d.items())
    test_means_d = dict((lmbd, np.mean(val)) for lmbd, val in␣
↪test_means_d.items())

    return valid_means_d, test_means_d
```

```
lambda_values = np.logspace(0, 12, num=500, base=10)
dict_valid, dict_test = cross_validation(Xn, Yn, 10, lambda_values)
```

Eventually, it has been asked from us to compute the R^2 of each model of the validation sets and plot it versus cross validation folds. Thus, we will find optimal λ value which maximizes R^2.

```
def r_squared(Y, pred):
    return (np.corrcoef(Y, pred)[0, 1]) ** 2
```

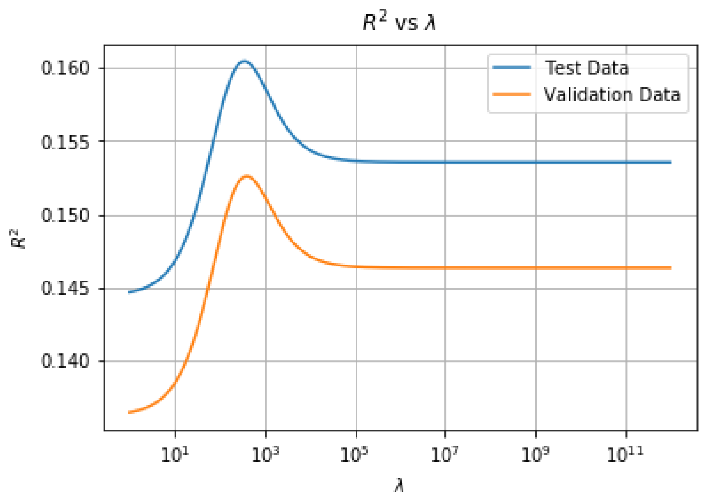
```
lambda_values = np.logspace(0, 12, num=500, base=10)
dict_valid, dict_test = cross_validation(Xn, Yn, 10, lambda_values)

lambda_opt = max(dict_valid, key=lambda k: dict_valid[k])

x_val, y_val = zip(*sorted(dict_valid.items()))
x_tst, y_tst = zip(*sorted(dict_test.items()))

plt.figure()
plt.plot(x_tst, y_tst)
plt.plot(x_val, y_val)
plt.legend(['Test Data', 'Validation Data',])
plt.ylabel(r'$R^2$')
plt.xlabel(r'$\lambda$')
plt.title(r'$R^2$'' vs ''$\lambda$')
plt.xscale('log')
plt.grid()
plt.show(block=False)

print("Optimal Lambda Value: ", lambda_opt)
```



```
Optimal Lambda Value:  395.5436244734702
```

b) At this point of the question we are asked to generate 1000 bootstrap samples. In total we will perform 500 iterations and we will be fitting a separate model in each iteration. For making these computations easier I have defined a bootstrap function with respect to the requirements.

```
np.random.seed(3)
def bootstrap(iter_num, x, y, lmbd):
    weight_new = []
    for i in range(iter_num):
        new_ind = np.random.choice(np.arange(np.size(y)), np.
 ↪size(y))
        x_new, y_new = Xn[new_ind], Yn[new_ind]
        weight_r = ridge_regression(x_new, y_new, lmbd)
        weight_new.append(weight_r)
    return weight_new
```

Also, it has been asked from us to identify the weights which are significantly different than 0. To do that operation, function given below is defined.

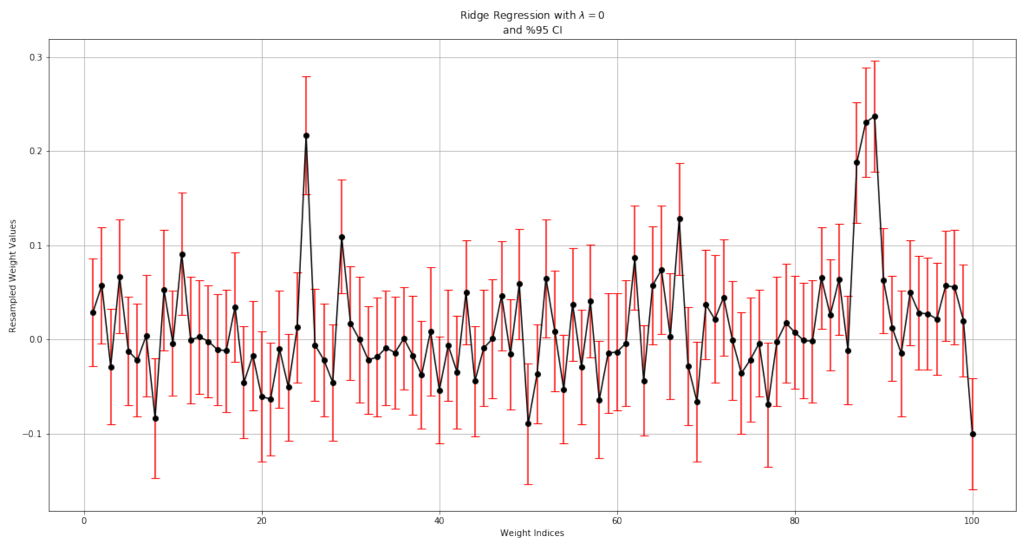
```
def find_significant_w(arr_mean, arr_std):
    p_values = 2 * (1 - norm.cdf(np.abs(arr_mean/arr_std)))
    significant_weights = np.where(p_values < 0.05)
    return significant_weights
```

Since it has been asked from us to generate samples with respect to ordinary least squares model, I have used bootstrap function with $\lambda = 0$. Therefore, found the means of the new weights and their %95 confidence intervals. Then, using the function **find_significant_w**, I found the weight indexes where mean of the generated weights is significantly higher than 0.

```
weight_new = []
weight_new = bootstrap(500, Xn, Yn, 0)

weight_new_mean = np.mean(weight_new, axis=0)
weight_new_std = np.std(weight_new, axis=0)
plt.figure(figsize=(20, 10))
plt.grid()
plt.errorbar(np.arange(1,101), weight_new_mean,␣
 ↪yerr=2*weight_new_std, ecolor='r', fmt='o-k', capsize=5)
plt.ylabel(r'Resampled Weight Values')
plt.xlabel(r'Weight Indices')
plt.title(r'Ridge Regression with ' r'$\lambda = 0$''\nand %95 CI')
plt.show(block=False)
print("Indices of the Resampled Weights which are significantly␣
 ↪different than zero:")
```

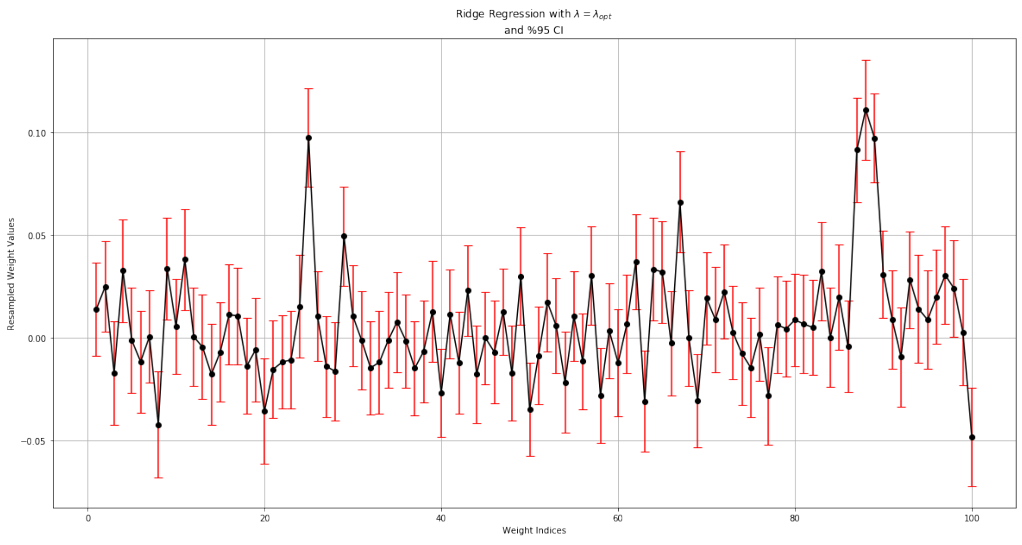
```
print(find_significant_w(weight_new_mean, weight_new_std)[0])
```



```
Indices of the Resampled Weights which are significantly different␣
↪than zero:
[ 3  7 10 20 24 28 48 49 51 57 61 64 66 68 76 82 84 86 87 88 89 96␣
↪99]
```

c) In this part we will be conducting the same operation as the previous part with only one difference. Instead of OLS method, we will be using ridge regression with the optimal lambda which we have found in **part a**.

```
weight_new_ridge = []
weight_new_ridge = bootstrap(500, Xn, Yn, lambda_opt)
weight_newR_mean = np.mean(weight_new_ridge, axis=0)
weight_newR_std = np.std(weight_new_ridge, axis=0)
plt.figure(figsize=(20, 10))
plt.grid()
plt.errorbar(np.arange(1,101), weight_newR_mean,␣
↪yerr=2*weight_newR_std, ecolor='r', fmt='o-k', capsize=5)
plt.ylabel(r'Resampled Weight Values')
plt.xlabel(r'Weight Indices')
plt.title(r'Ridge Regression with ' r'$\lambda =␣
↪\lambda_{opt}$''\nand %95 CI')
plt.show(block=False)
print("Indices of the Resampled Weights which are significantly␣
↪different than zero:")
print(find_significant_w(weight_newR_mean, weight_newR_std)[0])
```



```
Indices of the Resampled Weights which are significantly different␣
 ↪than zero:
[ 1  3  7  8 10 19 24 28 39 42 48 49 56 57 61 62 63 64 66 68 76 82␣
 ↪86 87
 88 89 92 96 97 99]
```

Normally, we would be expecting to have less number of indices, where their weights are significantly different than 0, since ridge regression forces the model to have smaller norms than the OLS model. This conflict may be caused by the variance. Significant, change in the variance might have caused us to have more values significantly different than 0.

Question 2

a) In this part we are examining two different populations of neurons. We would like to determine whether they are coming from the same distribution or not.

We will use bootstrap technique to generate 10000 samples and observe the two tailed p-value for out null hypothesis. Regarding to this hypothesis testing our null hypothesis is

$$H_0 = \mu_1 = \mu_2$$

As a result of our initial assumption our alternative hypothesis is that

$$H_A = \mu_1 \neq \mu_2$$

to determine in which conditions which hypothesis is true we need to know $P(x|H_0)$

We will need a bootstrap function to generate new samples. Bootstrap function is given below.

```python
def bootstrap(iter_num, x, seed=6):
    np.random.seed(seed)
    x_new = []
    for i in range(iter_num):
        new_ind = np.random.choice(np.arange(np.size(x)), np.
 ↪size(x))
        x_sample = x[new_ind]
        x_new.append(x_sample)
    return np.array(x_new)
```

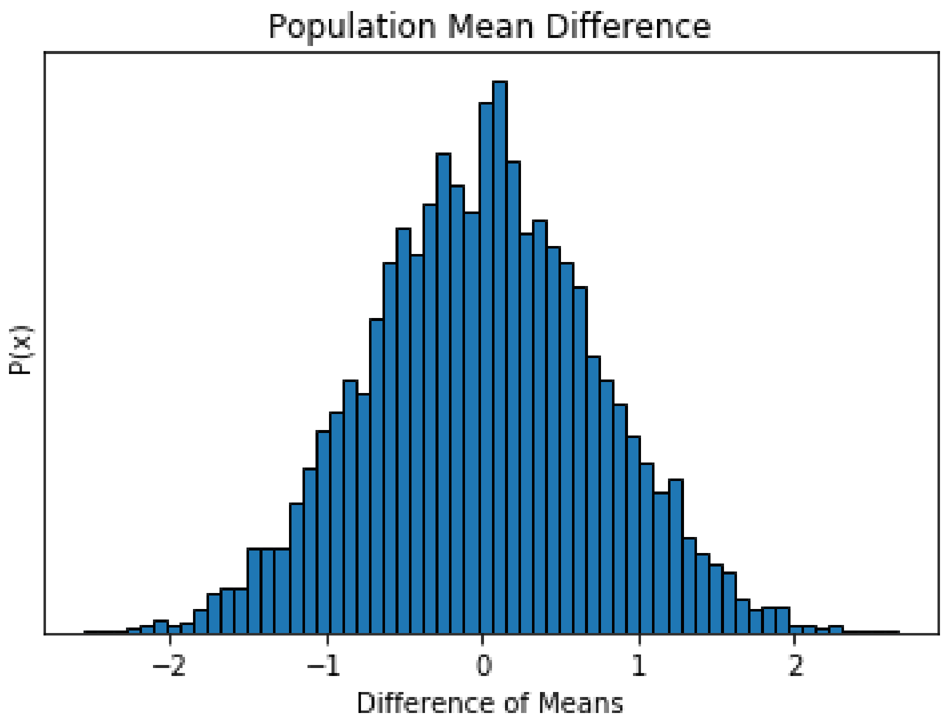
Since we are interested in whether they are coming from the same distribution or not, and our null hypothesis is that they are coming from the same population, I will treat them as one population. Therefore, using their combination new samples will be generated via bootstrapping. Then, I will separate that randomly generated samples with respect to the sizes that they were before and take a look at their means differences.

```python
def mean_difference(x, y, iterations):
    xy_concat = np.concatenate((x, y))
    xy_boot = bootstrap(iterations, xy_concat)
    x_boot = np.zeros((iterations, np.size(x)))
    y_boot = np.zeros((iterations, np.size(y)))
    for i in range(np.size(xy_concat)):
        if i < np.size(x):
            x_boot[:, i] = xy_boot[:, i]
        else:
            y_boot[:, i-np.size(x)] = xy_boot[:, i]
    x_means = np.mean(x_boot, axis=1)
    y_means = np.mean(y_boot, axis=1)
    mean_diff = x_means - y_means

    return mean_diff

mean_diff = mean_difference(pop1, pop2, 10000)
```

```
plt.figure()
plt.title('Population Mean Difference')
plt.xlabel('Difference of Means')
plt.ylabel('P(x)')
plt.yticks([])
plt.hist(mean_diff, bins=60, density=True, edgecolor='black')
plt.show(block=False)
```



Here we can see that, our new samples are approximately normally distributed. Hence, we can use z and p values to make comments about our initial hypothesis. Thus, we will be using the following function to compute the z and p values.

```
def find_z_and_p(x, mu):
    mu_0 = np.mean(x)
    sigma = np.std(x)
    z = np.abs((mu - mu_0) / sigma)
    p = (1 - norm.cdf(z))
    return z, p
```

```
z, p = find_z_and_p(mean_diff, np.mean(pop1)-np.mean(pop2))
print("z-score: ", z)
print("two sided p-value: ", 2*p)
```

```
z-score:  2.5416208430643454
two sided p-value:  0.011033979636079438
```

Since we have observed that we have a very small p value, we can conclude our hypothesis is only valid for small amount of samples and these two populations are coming from different populations.

b) Blood-oxygen level dependent responses from two voxels are provided in the data set. We will be examining their correlation in this part. Initially to do that we wil generate more samples using bootstrapping technique.Therefore, their correlations will be stored in an array for further observations.

```
with h5py.File('hw3_data3.mat', 'r') as file:
    vox1, vox2 = np.array(list(file['vox1'])).flatten(), np.
 ↪array(list(file['vox2'])).flatten()

vox1_boot = bootstrap(10000, vox1)
vox2_boot = bootstrap(10000, vox2)

corr_boot = np.zeros(10000)
for i in range(10000):
    corr_boot[i] = np.corrcoef(vox1_boot[i], vox2_boot[i])[0, 1]

corr_mean = np.mean(corr_boot)
sorted_corr = np.sort(corr_boot)
dif = np.size(sorted_corr)/40
corr_lower = sorted_corr[int(dif)]
corr_upper = sorted_corr[int(np.size(sorted_corr)-dif)]
print("Mean: ", corr_mean)
print("%95 CI: (", corr_lower, ", ", corr_upper, ")")

zero_corr = np.where(corr_boot < 10**(-2))
print("Number of elements with zero correlation: ", np.
 ↪size(zero_corr))
```

```
Mean:  0.5571045255666199
```

```
%95 CI: ( 0.3173570776868185 ,  0.7590485236833523 )
Number of elements with zero correlation:  0
```

While computing the 95% confidence interval, I have directly used the fact that we will be needing the 2.5%th and 97.5%th percentiles.

Also, this question asks us to find the percentile of the bootstrap distribution with zero correlation. Since, no such value is found as it can be seen above at the output, its percentile is also 0%.

c) In this part we have been expected to conduct another hypothesis testing. This time our null hypothesis will be

$$H_0 = Voxelresponseshavezeroornegativecorrelation$$

Relatively, our alternative hypothesis will be

$$H_A = Voxelshavepositivecorrelation$$

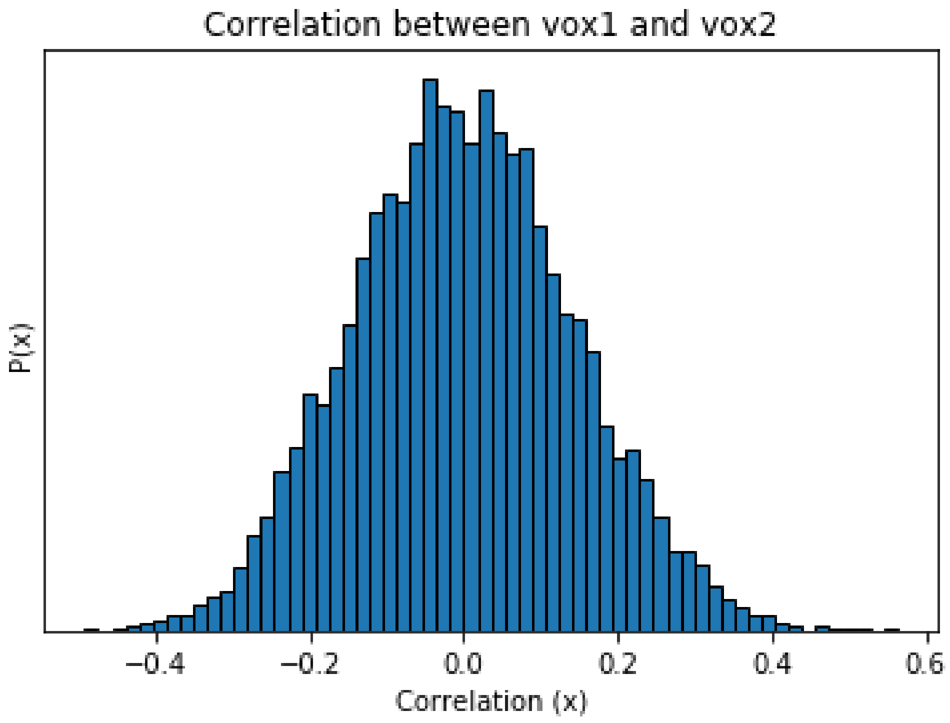
This time we will be sampling the responses of the voxels independently. Which means we will try to pick different values from **vox1** and **vox2** while sampling via bootstrapping.

```
vox1_indep = bootstrap(10000, vox1, 13)
vox2_indep = bootstrap(10000, vox2, 5)

corr_boot_indep = np.zeros(10000)
for i in range(10000):
    corr_boot_indep[i] = np.corrcoef(vox1_indep[i],␣
 ↪vox2_indep[i])[0, 1]
```

```
plt.figure()
plt.title('Correlation between vox1 and vox2')
plt.xlabel('Correlation (x)')
plt.ylabel('P(x)')
plt.yticks([])
plt.hist(corr_boot_indep, bins=60, density=True, edgecolor='black')
plt.show(block=False)

z, p = find_z_and_p(corr_boot_indep, np.corrcoef(vox1, vox2)[0, 1])
print("z-score: ", z)
print("one sided p value: ", p)
```



```
z-score:  3.9389626580609614
one sided p value:  4.091733224331762e-05
```

Our one sided p-value is small enought that we can clearly state that our null hypothesis is not correct. Then we can say that voxel responses actually have positive correlation.

d) We are also provided BOLD responses 20 different building and face images and we are going to conduct an hypothesis testing to these values assuming that we have same subject population. Our null hypothesis is

$$H_0 = Thereisnodifferencebetweenthebuildingandfaceresponses$$

On contrary we have

$$H_A = Thereisdifferencebetweenthebuildingandfaceresponses$$

Those hypothesis can be interpreted as

$$H_0 = \mu_1 - \mu_2 = 0$$

$$H_0 = \mu_1 - \mu_2 \neq 0$$

Our subject can be faced with four different cases which are (Face, Face), (Building, Building), (Face, Building) and (Building, Face). If subjects encounter with the same image obviously their difference in mean will be 0. Therefore, we generate a new sample by choosing from these 4 cases 10000 times. And we will be investigating its distribution and statistical values. Code which, I conducted these operations are provided below.

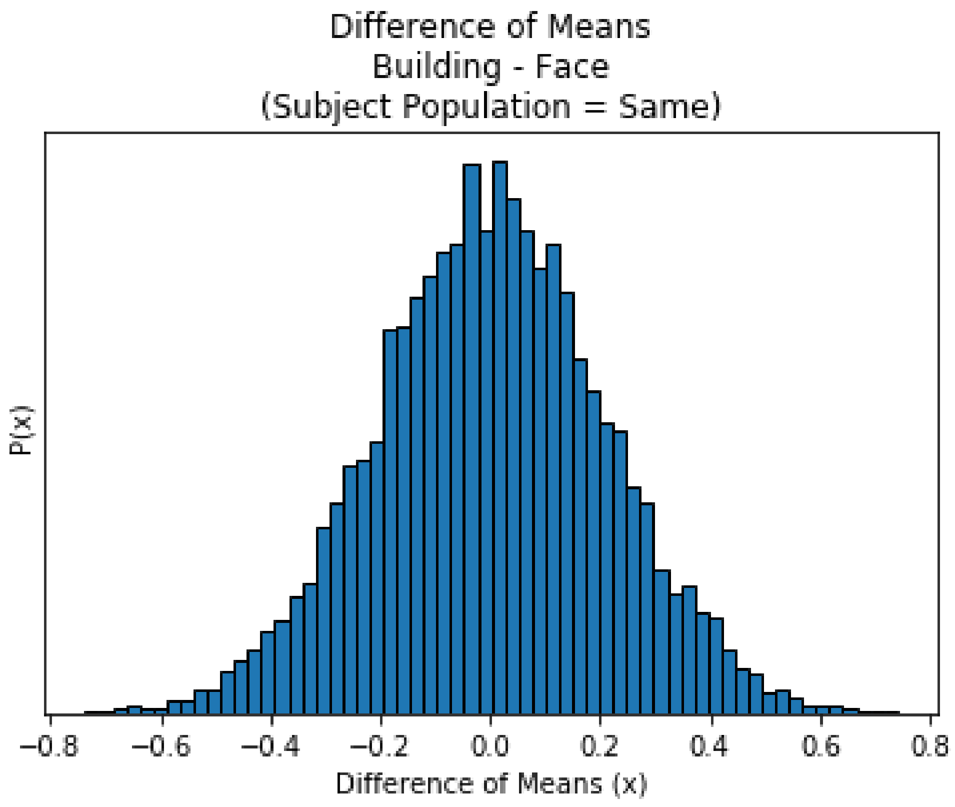
```
with h5py.File('hw3_data3.mat', 'r') as file:
    building, face = np.array(list(file['building'])).flatten(),␣
 ↪np.array(list(file['face'])).flatten()

mean_diff_d = np.zeros(10000)
diff_options = np.zeros(4)
choices = np.zeros(20)

for i in range(10000):
    for j in range(20):
        ind = np.random.choice(20)
        diff_options[0:1] = 0
        diff_options[2] = building[ind] - face[ind]
        diff_options[3] = -1 * diff_options[2]
        choices[j] = diff_options[np.random.choice(4)]
    mean_diff_d[i] = np.mean(choices)

plt.figure()
plt.title('Difference of Means\nBuilding - Face\n(Subject␣
 ↪Population = Same)')
plt.xlabel('Difference of Means (x)')
plt.ylabel('P(x)')
plt.yticks([])
plt.hist(mean_diff_d, bins=60, density=True, edgecolor='black')
plt.show(block=False)

z, p = find_z_and_p(mean_diff_d, np.mean(building) - np.mean(face))
print("z-score: ", z)
print("Two sided p value: ", 2*p)
```



```
z-score:  3.5733192689559212
Two sided p value:  0.00035248455559289127
```

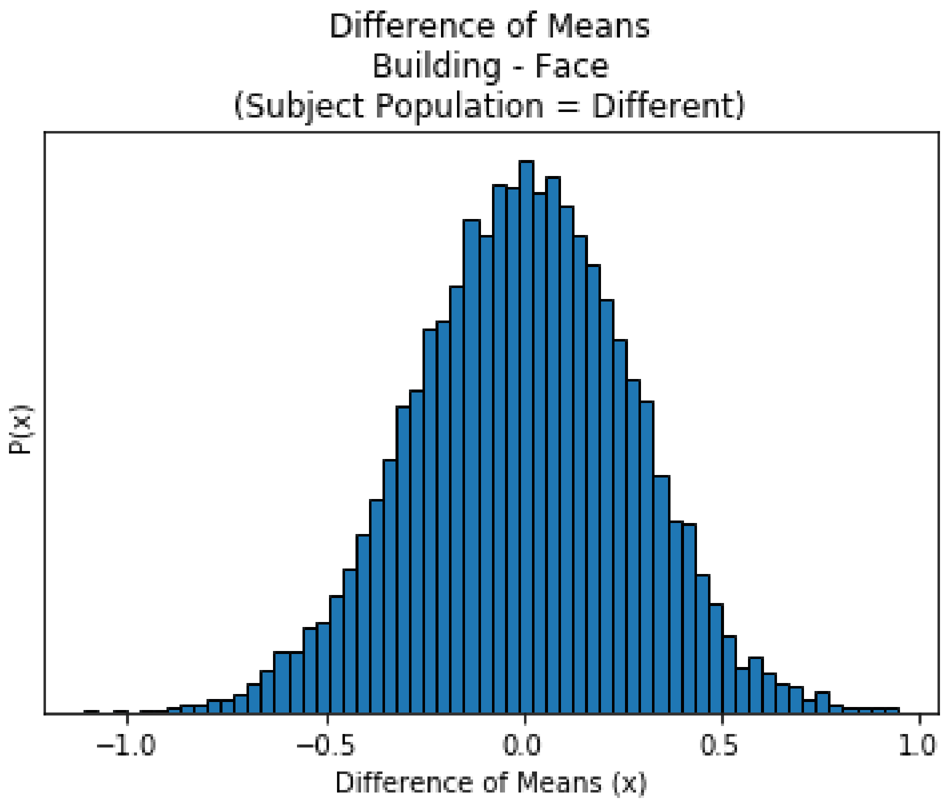
We have obtained a very small p value as a result. Thus, we can say that face and building images are not creating the same responses for the same subject population.

e) We will repeat **part d**, but this time we will consider subject groups are distinct. Therefore, we can directly apply the **mean_difference** function, which is used in **part a** to obtain its distribution. Then by computing its z-value and p-value, we will be deciding whether null hypothesis holds or not.

```
mean_diff_e = mean_difference(building, face, 10000)

plt.figure()
plt.title('Difference of Means\nBuilding - Face\n(Subject␣
 ↪Population = Different)')
plt.xlabel('Difference of Means (x)')
plt.ylabel('P(x)')
plt.yticks([])
plt.hist(mean_diff_e, bins=60, density=True, edgecolor='black')
plt.show(block=False)

z_e, p_e = find_z_and_p(mean_diff_e, np.mean(building)-np.
  ↪mean(face))
print("z-score: ", z_e)
print("Two sided p value: ", 2*p_e)
```



```
z-score:  2.693541201712695
Two sided p value:  0.007069740687789539
```

Again we have obtained a small p-value, which means null hypothesis will be rejected. So, there is obviously difference between building and face image responses regardless of the test subject population.