Practical Computing for Bioinformatics - HW2

Adnan Kaan Ekiz(r0776549) - Olympia Gennadi(r0829391)

December 2020

Contents

1 Question 1

Based on our initial observations, we came to a conclusion that these genes are all of **Protein coding** type and they belong to **Homo Sapiens** (Human) as a species. All genes belong to the same organism since they are retrieved from a specific genome in order to have them expressed with a treatment of interest. Therefore, we do not need to investigate the whole list. Furthermore, all genes encode the synthesis of proteins. We have used the **Ensembl Database** to take the necessary information and reach to this conclusion.

Here are the first 20 results we obtained from the research that we made about the listed gene IDs:

- ENSG00000122971: acyl-CoA dehydrogenase short chain - Protein coding
- ENSG00000047457: ceruloplasmin - Protein coding
- ENSG00000182197: exostosin glycosyltransferase 1 - Protein coding
- ENSG00000114480: 1,4-alpha-glucan branching enzyme 1 - Protein coding
- ENSG00000111057: keratin 18 - Protein coding
- ENSG00000108733: peroxisomal biogenesis factor 12 - Protein coding
- ENSG00000102144: phosphoglycerate kinase 1 - Protein coding
- ENSG00000151552: quinoid dihydropteridine reductase - Protein coding
- ENSG00000083720: 3-oxoacid CoA-transferase 1 - Protein coding
- ENSG00000165140: fructose-bisphosphatase 1 - Protein coding
- ENSG00000115461: insulin like growth factor binding protein 5 - Protein binding
- ENSG00000156136: deoxycytidine kinase - Protein coding
- ENSG00000134202: glutathione S-transferase mu 3 - Protein coding
- ENSG00000198034: ribosomal protein S4 X-linked - Protein coding
- ENSG00000148926: adrenomedullin - Protein coding
- ENSG00000089685: baculoviral IAP repeat containing 5 - Protein coding
- ENSG00000107159: carbonic anhydrase 9 - Protein coding
- ENSG00000099622: cold inducible RNA binding protein - Protein coding
- ENSG00000006125: adaptor related protein complex 2 subunit beta 1 - Protein coding
- ENSG00000136943: cathepsin V - Protein coding

2 Question 2,3 and 4

We have divided the overall task to several different functions in order to explain our solution clearly. Below, you can see the list of the functions created to accomplish the task:

1. **Processing of the Text Files**
2. **Extract Selected and Not Selected Genes**
3. **Determine Chi-Square Value with a given matrix**
4. **Process GO Terms**
5. **Create the Unique List**
6. **Main Code**

In each part, we are planning to explain the logic of our functions. In addition to this, we have defined several assumptions that needs mentioning since it might change the result of the solution that we have.

2.1 Processing of the Text Files

In this part, we try to explain the processing of our text files. Overall, we tried to create different dictionaries to count the instance of every gene in the text file so that it would be more practical to find the number of times that a certain gene ID has been seen; although our code only considers unique gene IDs (keys of the dictionary). However, in the future it would be easier to convert the code such that it will also consider multiple instances of the same ID.

2.1.1 Reading the GO Annotations

In addition to the dictionary described above, we have created 2 more additional dictionaries while reading the "GOannotations.txt" file since it also contains the different GO terms and the explanations of it.

The first dictionary that we have created holds the GO terms as a key and explanations corresponding to it as value; while another dictionary is holding the GO terms as a key and list of corresponding gene IDs as value. One of the biggest reasons to do this was the ease of access to data when using the dictionary. Based on the trials we had, we have concluded that using the dictionaries instead of lists increases the speed of the program dramatically.

– Beginning of the function –

```
def processGO(fileReader):

    goExpD = {}
    geneCount = {}
    goToID = {}
    next(fileReader)

    for line in fileReader:
        lineArr = line.split("\t")
        '''
        print(lineArr[0])
        print(lineArr[1])
        print(lineArr[2])
        '''
        goExp = lineArr[2][:-1]
        geneID = lineArr[0]
        goTerm = lineArr[1]

        #go explanation dictionary
        if goTerm not in goExpD.keys():
            goExpD[goTerm] = goExp

        #gene count dictionary
        if geneID in geneCount.keys():
            geneCount[geneID] += 1
        else:
            geneCount[geneID] = 1


        if goTerm not in goToID.keys():
            goToID[goTerm] = [geneID]
        else:
            goToID[goTerm].append(geneID)

    return geneCount, goToID, goExpD
```

– End of the function –

Explanation of different lines of the code is as follows:

- Function starts by initializing 3 different dictionaries and then we skip the first line of the read since this part consists of headers that we don't need
- **for line in fileReader:** Start reading the "GOannotations.txt" file line by line
- **geneID = lineArr[0], goTerm = lineArr[1], goExp = lineArr[2][:-1]**: Dividing the line to 3 pieces in order to create the dictionaries. These terms are the gene IDs, GO terms and GO explanations
- **if goTerm not in goExpD.keys():** If GO term described in the line has not been used as a key before in the dictionary
- **goExpD[goTerm] = goExp**: Then add the GO term as key and the explanation part as value corresponding to it
- **if geneID in geneCount.keys():** If gene ID can be found in the dictionary as a key
- **geneCount[geneID] += 1**: Then increase the count corresponding to the gene ID by 1
- **geneCount[geneID] = 1**: If the gene ID can't be found in the dictionary as a key, add an instance of it and set the counter to 1
- **if goTerm not in goToID.keys():** If the GO term is not found in the dictionary as a key
- **goToID[goTerm] = [geneID]** Then put a list with the first gene ID that corresponds to the GO term as value and set the GO term as key

- **goToID[goTerm].append(geneID)** If GO term already exists as a key, then add the new found gene ID into the list that corresponds to the key

2.1.2 Process Genes

This function is created to read the text files that consists of only gene IDs. Hence, it is used with "allGenes.txt" and "IDS2.txt" files and generates a dictionary that counts how many times a certain gene ID has been seen by using the gene ID itself as a key.

-Beginning of the function-

```
def processGenes(reader):

    geneCount = {}
    for line in reader:
        lineArr = line.split("\t")
        geneID = lineArr[0][:-1]

        if geneID in geneCount:
            geneCount[geneID] += 1
        else:
            geneCount[geneID] = 1

    return geneCount
```

-End of the function-

Explanation of the code as follows:

- **for line in reader:** For every line in the text file
- **geneID = lineArr[0][:-1]** Splitting the line string and taking the part corresponding to the gene ID
- **if geneID in geneCount:** If gene ID is used as a key in the dictionary
- **geneCount[geneID] += 1**: Then increase the count for that gene ID by 1
- **geneCount[geneID] = 1**: If the gene ID can't be found in the dictionary as a key, put a new instance of it and set the counter for that gene ID to 1

2.2 Extract Selected and Not Selected Genes

In this part we create a function named extractSelOrNot in which we distinguish the IDs of Selected genes that are expressed differently after the treatment than the rest of the genes in the organism. In order to achieve this, we create two dictionaries (sel, notSel) in which we are going to place these genes and afterwards we compare the lists containing All genes (allGenes) and Selected genes (IDS2) between them. We retrieve from All genes list the IDs of selected genes included in the IDS2 list and the other genes from All genes list that are not included in IDS2 list.

The formation of these dictionaries aimed to the split of the given lists in smaller lists to make the calculation of the variables in the next functions easier and faster.

-Beginning of the function-

```
def extractSelOrNot(selected , all):

    sel = {}
    notSel = {}

    #genes that are not selected
    for key in all:
        if key not in selected:
            if key not in notSel:
                notSel[key] = all[key]

    #genes that are selected
    for key in selected:
        if key in all:
            if key not in sel:
                sel[key] = selected[key]

    return sel , notSel
```

-End of the function-

The explanation of each line is given below :

- **sel = {}**: Create an empty dictionary to insert the IDs as key and counter corresponding to it for selected genes
- **notSel = {}**: Create an empty dictionary to insert the gene IDs as key and counter value corresponding to it. These gene IDs are not among the selected genes
- **for key in all:** For every key in dictionary 'all' do the following process
- **if key not in selected:** If the key does not exist in dictionary 'selected' follow the below steps
- **if key not in notSel:** After searching the 'selected' dictionary, check if the key value exists in the constructed dictionary 'notSel'. If it does not exist
- **notSel[key] = all[key]**: Then add the key and the corresponding value in 'notSel' dictionary
- **for key in selected:** Begin a new loop. For every key in dictionary 'selected'
- **if key in all:** If the key value exists in 'all' dictionary move to the next step
- **if key not in sel:** If the key value does exists in the constructed dictionary 'sel' then follow the next step
- **sel[key] = selected[key]**: Add the specific key and the corresponding value to dictionary 'sel'

2.3 Determine Chi-Square Value with a given matrix

In this function we calculate Chi square statistic based on the observed and expected values. Using the function "processGOterms" we have designed one matrix with the observed values A, B, C and D and another one with the expected values calculated by the given equation.

-Beginning of the function-

```
def determineChiSq(expected, observed):

    result = 0
    for row in range(0,2):
      for col in range(0,2):
        if expected[row][col] != 0:
          result = result + ((observed[row][col]-expected[row][col])**2)
            /expected[row][col] #continuing

    return result
```

-End of the function-

In the following lines we analyze code:

- **result = 0**: Initialize a new variable 'result' with the zero value
- **for row in range(0,2):** Each matrix contains 2 rows. For every row in observed matrix and expected matrix
- **for col in range(0,2):** Select the column in each matrix. This loop gives us the possibility to select each cell and element of the desired matrix
- **if expected[row][col] != 0:** If the element in the expected matrix is not 0 then calculate the variable 'result' as follows. This allow us to avoid the division with the value 0 and get non defined results
- **result = result + ((observed[row][col] - expected[row][col])**2) / expected[row][col]**: In this line we finally calculate the chi square by using the necessary equation. We take each element of the expected matrix and we subtract it from each element of the observed matrix. We calculate the square of this subtraction and in the end we divide it with the element of the expected matrix. We follow this procedure for every cell and finally the chi square is the sum of all of the calculations made during this procedure

2.4 Calculating the Result Dictionary for each of the GO Terms

In this function, by using the extracted selected and not selected dictionaries we create another dictionary that calculates the chi square value for every corresponding GO term key. For every gene ID list that corresponds to the GO term key, we check the selected and not selected dictionaries to determine variables A and B and also the other variables needed for the chi square calculation. Using dictionary allowed us to check whether a certain gene ID is available in selected and not selected in constant time. Hence, decreased the total running time of our code dramatically.

-Beginning of the function-

```
def processGOterms(sel, notSel, goToID):

    goValue = {}
    for GO in goToID:

        A = 0
        B = 0

        for ID in goToID[GO]:
            if ID in sel:
                A += 1

            if ID in notSel:
                B += 1

        C = len(sel) - A
        D = len(notSel) - B

        Row1 = A + B
        Row2 = C + D
        Col1 = A + C
        Col2 = B + D
        N = A + B + C + D

        exp_A = (Row1 * Col1) / N
        exp_B = (Row1 * Col2) / N
        exp_C = (Row2 * Col1) / N
        exp_D = (Row2 * Col2) / N

        expected = np.array([[exp_A, exp_B], [exp_C, exp_D]])
        observed = np.array([[A, B], [C, D]])

        chiVal = determineChiSq(expected, observed)

        goValue[GO] = chiVal

    return goValue
```

-End of the function-

The explanation of each line is given below:

- **for GO in goToID:** For every GO term key in the dictionary
- **for ID in goToID[GO]:** For each gene ID in the list that is corresponding to the GO term
- **if ID in sel:** If the gene ID is inside of the selected dictionary as key
- **A += 1**: Then increasing the A counter by 1 each time it is confirmed that gene ID is inside of the selection dictionary
- **if ID in notSel**: If the gene ID is inside of the not selected dictionary as key

- **B += 1**: Then increasing the B counter by 1 each time it is confirmed that gene ID is inside of the not selected dictionary
- **C = len(sel) - A**: Substracting the A amount from the total element number in selected dictionary would be enough to find the C value
- **D = len(notSel) - B**: Substracting the B amount from the total element number in not selected dictionary would be enough to find the D value
- Then we have determined the marginal values for each column and row. Hence, calculating the expected **A, B, C, D** value
- **expected = np.array([[exp_A, exp_B], [exp_C, exp_D]])**: Creating the expected matrix to give it as parameter in order to calculate Chi Square value for the specific GO term
- **observed = np.array([[A, B], [C, D]])**: Creating the observed matrix to give it as parameter in order to calculate Chi Square value for the specific GO term
- **chiVal = determineChiSq(expected, observed)**: Get the Chi square value for the GO term by calling the **"determineChiSq"** function
- **goValue[GO] = chiVal**: Add the calculated Chi square value into dictionary for the specific GO term

2.5 Create the Unique List

In this section we design a function in which we create a new list with the unique values of gene IDs included. This will help us to retrieve only the unique gene IDs of GO annotations list and remove the replicates. Therefore, we use this list in the function "processGOterms" by examining each GO term separately without the need of checking out a GO term for a second time.

-Beginning of the function-

```
def unique(list):
    unique_list = []

    for x in list:
        if x not in unique_list:
            unique_list.append(x)

    return unique_list
```

-End of the function-

The explanation of each line is given below:

- **unique_list = []**: Initialize an empty list 'unique_list'
- **for x in list:** For each element in the list follow the next steps
- **if x not in unique_list:** If the element is not contained into the list then
- **unique_list.append(x)**: Add the element in the list

2.6 Main Code

In the Main code we open the text files given for the accomplishment of the assignment and we also call all the functions of the script to execute the corresponding commands with the appropriate parameters. We finally retrieve the chi square value for each GO term and we put the results in a new dictionary. In order to gain the top five GO categories with the highest chi square statistic, we sort the dictionary in a descending order and create a new text file "GO_categories.txt" to insert the results.

-Beggining of the Main Code-

```
allGenes = open("allGenes.txt",'r')
go = open("GOannotations.txt",'r')
selected = open("IDS2.txt",'r')

idCount, goToID, goExp = processGO(go)
selectedCount = processGenes(selected)
allCount = processGenes(allGenes)

for GO in goToID:
    goToID[GO] = unique(goToID[GO])

print("Number of genes in IDS2.txt:", len(selectedCount) )
print("Number of genes in GOannonations.txt:", len(idCount) )
print("Number of total genes in allGenes.txt:", len(allCount) )

sel, notSel = extractSelOrNot(selectedCount, allCount)

result = processGOterms(sel, notSel, goToID)

sorted_dict = dict(sorted( result.items(), key=operator.itemgetter(1), reverse=True))

final = []
for key in sorted_dict:
    element = str(key) + "-" + str(sorted_dict[key]) + "-" + goExp[key]
    final.append(element)

print("Printing the TOP 5 Results:")
for i in final[:5]:
    print(i)

print("Writing all the results to a file...")

with open("GO_categories.txt", 'w') as output:
    output.write("Go term" + '\t\t' + "Chi square" + '\t\t\t' + "GO Term Explanation" + '\n')

    for key in sorted_dict:
        output.write(str(key) + '\t' + str(sorted_dict[key]) + '\t' + (goExp[key]) + '\n')
```

-End of the Main Code-

The explanation of the Main Code is given below:

- **allGenes = open("allGenes.txt",'r')**: Open the file allGenes.txt as 'allGenes' and make it available for reading

- **go = open("GOannotations.txt",'r')**: Open the file GOannotations.txt as 'go' and make it available for reading
- **selected = open("IDS2.txt",'r')**: Open the file IDS2.txt as 'selected' and make it available for reading
- **idCount, goToID, goExp = processGO(go)**: Retrieve the three dictionaries created by the function "processGO"
- **selectedCount = processGenes(selected)**: Return the dictionary which contains the selected genes of IDS2 list
- **allCount = processGenes(allGenes)**: Return the dictionary which contains all genes of the organism
- **for GO in goToID:** For every key in dictionary 'goToID'
- **goToID[GO] = unique(goToID[GO])**: Keep only the unique values of GO term accessions with the contribution of function "unique"
- In the next 3 lines we print in the command window the number of genes included in GOannotations, allGenes and IDS2 files
- **sel, notSel = extractSelOrNot(selectedCount, allCount)**: Distinguish the selected genes from the other genes by using the function "extractSelOrNot"
- **result = processGOterms(sel, notSel, goToID)** Run the function "processGOterms" for the 3 dictionaries and take for each GO term the corresponding Chi square statistic. The results are placed in a new dictionary 'result' which keeps as a key the GO term and as a value for each key the chi square statistic
- **sorted_dict = dict(sorted(result.items(), key=operator.itemgetter(1), reverse=True))**: Sort the previous dictionary in a descending order based on the Chi square statistic
- **final = []**: Initialize an empty list 'final'
- **for key in sorted_dict:** For every key in the sorted dictionary
- **print(key,"-",sorted_dict[key])**: Print in the command window the key and the corresponding value of the dictionary. That means that we actually print every GO term (from sorted dictionary) and the related chi square statistic
- **element = str(key) + " - " + str(sorted_dict[key]) + " - " + goExp[key]**: Create a new variable 'element' which includes the GO term, the chi square statistic and the GO explanation
- **final.append(element)**: Insert the variable in the list 'final' for each GO term
- **print("Printing the TOP 5 Results:")**: Print in the screen the top 5 results
- **for i in final[:5]:** for the first 5 lines in the list 'final'
- **print(i)**: Print in the command window each element of the line containing the GO term, chi square and GO explanation
- **with open("GO_categories.txt", 'w') as output:** Open a new text file named GO_categories.txt and make it available for writing

- **output.write("Go term" + '\t \t' + "Chi square" + '\t \t \t' + "GO Term Explanation" + '\n'):** Write on the first line of the new text file. Create 3 separate segments : GO term, Chi square and GO Term Explanation. Then move to a new line
- **for key in sorted_dict:** For every key in the sorted dictionary
- **output.write(str(key) + '\t' + str(sorted_dict[key]) + '\t' + (goExp[key]) + '\n')** Write in the text file the GO term -1st column- , the chi square statistic -2nd column- and the corresponding GO term explanation -3rd column-. Then move to the next line in order to write the next GO term and values.

2.7 Assumptions

As we have mentioned couple times during the report, we are aware that we have used couple of assumptions in the solution in order to come up with a code that solves the given problem. Assumptions used can be seen below:

- One of these assumptions is the fact that we have only used the unique gene IDs in order to calculate the Chi Square value of a given GO term. This might have an effect on the solution of the question. However, if we decide not to use the unique gene IDs, this can be achieved by changing the code slightly due to the efficiency and the counters stored by the dictionaries.
- Another assumption that is used was ignoring the genes that are not existed in "allGenes.txt". For this reason, we have generated extractSelOrNot function to get rid of the gene IDs that are not mentioned in the text file "allGenes.txt".
- In addition, an important fact is the differentiation of the genes in "allGenes.txt" file. Using the function extractSelOrNot we create dictionaries with the selected genes that are differentially expressed after the treatment and the genes that are not selected. In that way we can eliminate the time of our program and make comparisons between smaller dictionaries.
- When calculating the Chi square value, we have decided to ignore the values where the expected square value is 0 since divider will be 0 and it will cause us to get a nan value.

2.8 Full Code

-Full script is listed below-

```
import numpy as np
import operator


def processGO(fileReader):


    goExpD = {}
    geneCount = {}
    goToID = {}
    next(fileReader)


    for line in fileReader:
        lineArr = line.split("\t")
        '''
        print(lineArr[0])
        print(lineArr[1])
        print(lineArr[2])
        '''
        goExp = lineArr[2][:-1]
        geneID = lineArr[0]
        goTerm = lineArr[1]

        #go explanation dictionary
        if goTerm not in goExpD.keys():
            goExpD[goTerm] = goExp

        #gene count dictionary
        if geneID in geneCount.keys():
            geneCount[geneID] += 1
        else:
            geneCount[geneID] = 1


        if goTerm not in goToID.keys():
            goToID[goTerm] = [geneID]
        else:
            goToID[goTerm].append(geneID)

    return geneCount, goToID, goExpD



def processGenes(reader):

    geneCount = {}

    for line in reader:
        lineArr = line.split("\t")
        geneID = lineArr[0][:-1]

        if geneID in geneCount:
            geneCount[geneID] += 1
        else:
            geneCount[geneID] = 1

    return geneCount



def extractSelOrNot(selected, all):

    sel = {}
    notSel = {}

    for key in all:
        if key not in selected:
            if key not in notSel:
                notSel[key] = all[key]

    for key in selected:
        if key in all:
            if key not in sel:
                sel[key] = selected[key]

    return sel, notSel



def determineChiSq(expected, observed):

    result = 0
    for row in range(0,2):
        for col in range(0,2):
            if expected[row][col] != 0:
                result = result + ((observed[row][col] - expected[row][col])**2) / expected[row][col]

    return result



def processGOterms(sel, notSel, goToID):

    goValue = {}
    for GO in goToID:

        A = 0
        B = 0

        for ID in goToID[GO]:
            if ID in sel:
                A += 1

            if ID in notSel:
                B += 1

        C = len(sel) - A
        D = len(notSel) - B

        Row1 = A + B
        Row2 = C + D
        Col1 = A + C
        Col2 = B + D
        N = A + B + C + D

        exp_A = (Row1 * Col1) / N
        exp_B = (Row1 * Col2) / N
        exp_C = (Row2 * Col1) / N
        exp_D = (Row2 * Col2) / N

        expected = np.array([[exp_A, exp_B], [exp_C, exp_D]])
        observed = np.array([[A, B], [C, D]])

        chiVal = determineChiSq(expected, observed)

        goValue[GO] = chiVal

    return goValue



def unique(list):

    unique_list = []

    for x in list:

        if x not in unique_list:
            unique_list.append(x)

    return unique_list



#———Main Code——————————


allGenes = open(”allGenes.txt”,’r’)
```

```
go = open("GOannotations.txt",'r')
selected = open("IDS2.txt",'r')

idCount, goToID, goExp = processGO(go)
selectedCount = processGenes(selected)
allCount = processGenes(allGenes)



for GO in goToID:
    goToID[GO] = unique(goToID[GO])

print("Number of genes in IDS2.txt:", len(selectedCount) )
print("Number of genes in GOannonations.txt:", len(idCount) )
print("Number of total genes in allGenes.txt:", len(allCount) )

sel, notSel = extractSelOrNot(selectedCount, allCount)


result = processGOterms(sel, notSel, goToID)

sorted_dict = dict(sorted( result.items(), key=operator.itemgetter(1), reverse=True))

final = []
for key in sorted_dict:
    element = str(key) + "-" + str(sorted_dict[key]) + "-" + goExp[key]
    final.append(element)

print("Printing the TOP 5 Results:")
for i in final[:5]:
    print(i)

print("Writing all the results to a file...")


with open("GO_categories.txt", 'w') as output:
    output.write("Go term" + '\t\t' + "Chi square" + '\t\t\t' + "GO Term Explanation" + '\n')

    for key in sorted_dict:
        output.write(str(key) + '\t' + str(sorted_dict[key]) + '\t' + (goExp[key]) + '\n')
```

-End of the script-