

```

import sys, math

#constants
DEBUG = True
INPUT_FILE_NAME_TRAIN = "ALL_AML_gr_no_one_folds.thr.train.csv"

def debug(logMsg):
    if DEBUG:
        print(logMsg)

def computeSignificantGenes(input_file_name):
    geneAMLAverage = {}
    geneALLAverage = {}
    geneALLStdDev = {}
    geneAMLStdDev = {}
    signalToNoiseRatiosALL = {}
    signalToNoiseRatiosAML = {}
    T_valuesALL = {}
    T_valuesAML = {}
    with open(input_file_name) as f:

        #27 ALL observations, 11 AML observations
        ALL_N = 27
        AML_N = 11

        #create a list of lines, stripped of the newline
        content = [line.rstrip('\n') for line in f]

        #open the file which will have the lines without one ratios; we don't want to append
        # out_file = open(output_file_name, "w")

        #just add the first line back to the result lines
        idLine = content.pop(0)
        # resultLines.append(idLine + "\n") #writelines requires newlines

        ALL_avg_sum = 0
        AML_avg_sum = 0

        #for every line (gene), compute the fold difference
        for line in content:
            #we need every integer
            intStrings = line.split(',')

            geneName = intStrings.pop(0)

            #get the
            lineExpressionValues = [int(exprVal) for exprVal in intStrings]

            #first slice the list into the ALL and AML values
            ALL_values = lineExpressionValues[0:27]
            AML_values = lineExpressionValues[27:]

            ALL_values_sum = sum(ALL_values)

```

```

AML_values_sum = sum(AML_values)

ALL_avg = sum(ALL_values) / ALL_N
AML_avg = sum(AML_values) / AML_N

ALL_avg_sum += ALL_avg
AML_avg_sum += AML_avg

geneAMLAverage[geneName] = AML_avg
geneALLAverage[geneName] = ALL_avg

ALL_sumOfSquares = 0
for val in ALL_values:
    ALL_sumOfSquares += (val)**2

AML_sumOfSquares = 0
for val in AML_values:
    AML_sumOfSquares += (val)**2

ALL_stdDev = math.sqrt((ALL_N * ALL_sumOfSquares - (ALL_values_sum**2)) /
(ALL_N*(ALL_N-1)))
AML_stdDev = math.sqrt((AML_N * AML_sumOfSquares - (AML_values_sum**2)) /
(AML_N*(AML_N-1)))

geneALLStdDev[geneName] = ALL_stdDev
geneAMLStdDev[geneName] = AML_stdDev

ALL_signalToNoise = (ALL_avg - AML_avg) / (ALL_stdDev + AML_stdDev)
ALL_T_value = (ALL_avg - AML_avg) / math.sqrt((ALL_stdDev*ALL_stdDev/ALL_N) +
(AML_stdDev*AML_stdDev/AML_N))

AML_signalToNoise = (AML_avg - ALL_avg) / (ALL_stdDev + AML_stdDev)
AML_T_value = (AML_avg - ALL_avg) / math.sqrt((ALL_stdDev*ALL_stdDev/ALL_N) +
(AML_stdDev*AML_stdDev/AML_N))

# if(geneName == 'X52005_at'):
#     print("ALL X52005_at T-value: " + str(ALL_T_value))
#     print("AML X52005_at T-value: " + str(AML_T_value))
#     print("ALL X52005_at S2N: " + str(ALL_signalToNoise))
#     print("AML X52005_at S2N: " + str(AML_signalToNoise))

signalToNoiseRatiosALL[geneName] = ALL_signalToNoise
T_valuesALL[geneName] = ALL_T_value
signalToNoiseRatiosAML[geneName] = AML_signalToNoise
T_valuesAML[geneName] = AML_T_value
#end for

ALL_T_values_lst = T_valuesALL.items()
ALL_signal2Noise_lst = signalToNoiseRatiosALL.items()
ALL_T_values_lst = sorted(ALL_T_values_lst, key=lambda x: x[1])
ALL_signal2Noise_lst = sorted(ALL_signal2Noise_lst, key=lambda x: x[1])

AML_T_values_lst = T_valuesAML.items()

```

```

AML_signal2Noise_lst = signalToNoiseRatiosAML.items()
AML_T_values_lst = sorted(AML_T_values_lst, key=lambda x: x[1])
AML_signal2Noise_lst = sorted(AML_signal2Noise_lst, key=lambda x: x[1])

#from ascending to descending
ALL_T_values_lst.reverse()
ALL_signal2Noise_lst.reverse()
AML_T_values_lst.reverse()
AML_signal2Noise_lst.reverse()

AML_top_50_T_values = AML_T_values_lst[0:50]
ALL_top_50_T_values = ALL_T_values_lst[0:50]

AML_top_50_S2N = AML_signal2Noise_lst[0:50]
ALL_top_50_S2N = ALL_signal2Noise_lst[0:50]

AML_top_3_T_values = AML_T_values_lst[0:3]
ALL_top_3_T_values = ALL_T_values_lst[0:3]

AML_top_3_S2N = AML_signal2Noise_lst[0:3]
ALL_top_3_S2N = ALL_signal2Noise_lst[0:3]

print("\nHighest signal to noise ratio for ALL: " + str(ALL_top_50_S2N[0]))
print("50th Highest signal to noise ratio for ALL: " + str(ALL_top_50_S2N[-1]))
print("\nHighest T-value for ALL: " + str(ALL_top_50_T_values[0]))
print("50th Highest T-value for ALL: " + str(ALL_top_50_T_values[-1]))
print("\nHighest signal to noise ratio for AML: " + str(AML_top_50_S2N[0]))
print("50th highest signal to noise ratio for AML: " + str(AML_top_50_S2N[-1]))
print("\nHighest T-value for AML: " + str(AML_top_50_T_values[0]))
print("50th highest T-value for AML: " + str(AML_top_50_T_values[-1]))

AML_top_50_T_values_genes_only = set([val[0] for val in AML_top_50_T_values])
ALL_top_50_T_values_genes_only = set([val[0] for val in ALL_top_50_T_values])

AML_top_50_S2N_genes_only = set([val[0] for val in AML_top_50_S2N])
ALL_top_50_S2N_genes_only = set([val[0] for val in ALL_top_50_S2N])

print("\n\nAML_top_50_S2N_genes_only: " + str(AML_top_50_S2N_genes_only))
print("\n\nALL_top_50_S2N_genes_only: " + str(ALL_top_50_S2N_genes_only))

common_AML_genes = AML_top_50_T_values_genes_only.intersection(AML_top_50_S2N_genes_only)
common_ALL_genes = ALL_top_50_T_values_genes_only.intersection(ALL_top_50_S2N_genes_only)

print("intersection of ALL top 50 gene sets selected by S2N ratio and T-Value: " +
str(common_ALL_genes))
print("intersection of AML top 50 gene sets selected by S2N ratio and T-Value: " +
str(common_AML_genes))

print("size of intersection of ALL top 50 gene sets selected by S2N ratio and T-Value:
" + str(len(common_ALL_genes)))
print("size of intersection of AML top 50 gene sets selected by S2N ratio and T-Value:
" + str(len(common_AML_genes)))

```

```
AML_top_3_T_values_genes_only = set([val[0] for val in AML_top_3_T_values])
ALL_top_3_T_values_genes_only = set([val[0] for val in ALL_top_3_T_values])

AML_top_3_S2N_genes_only = set([val[0] for val in AML_top_3_S2N])
ALL_top_3_S2N_genes_only = set([val[0] for val in ALL_top_3_S2N])

common_AML_genes_top_3 =
AML_top_3_T_values_genes_only.intersection(AML_top_3_S2N_genes_only)
common_ALL_genes_top_3 = ALL_top_3_S2N_genes_only.intersection(ALL_top_3_S2N_genes_only)

print("\n\nintersection of ALL top 3 gene sets selected by S2N ratio and T-Value: " +
str(common_ALL_genes_top_3))
print("\n\nintersection of AML top 3 gene sets selected by S2N ratio and T-Value: " +
str(common_AML_genes_top_3))
#end with open
#end compute

computeSignificantGenes(INPUT_FILE_NAME_TRAIN)
```