Data Mining Assignment #5

Jeremy Keys

11/02/2017

Part I
Examining gene variation

A) I completed this section with a Python script. This script also accomplishes sections B, C, D, and F.

compute-fold-values.py

```
import sys
#constants DEBUG = True
INPUT_FILE_NAME_TRAIN = "ALL_AML_gr.thr.train.csv"
OUTPUT_FILE_NAME_FOLD_VALUES = "ALL_AML_gr_no_one_folds.thr.
    train.csv"
OUTPUT_FILE_NAME_GENE_DISTRIBUTION = "ALL_AML_gr.distribution.
    train.txt"

FOLD_DIFF_LT_2 = 'LT2'   #Val <= 2
FOLD_DIFF_2_4 = '2_4'    #2 < Val <= 4
FOLD_DIFF_4_8 = '4-8'    #4< Val <= 8
FOLD_DIFF_8_16 = '8-16'  #4 < Val <= 8
FOLD_DIFF_16_32 = '16-32'     #...
FOLD_DIFF_32_64 = '32-64'
FOLD_DIFF_64_128 = '64-128'
FOLD_DIFF_128_256 = '128-256'
FOLD_DIFF_256_512 = '256-512'
FOLD_DIFF_GT_512 = 'GT512'

#globals
countFoldDiffPerRange = {FOLD_DIFF_LT_2 : 0, FOLD_DIFF_2_4 :
    0, FOLD_DIFF_4_8 : 0, FOLD_DIFF_8_16 : 0, FOLD_DIFF_16_32
    : 0, FOLD_DIFF_32_64 : 0, FOLD_DIFF_64_128 : 0,
    FOLD_DIFF_128_256 : 0, FOLD_DIFF_256_512 : 0,
    FOLD_DIFF_GT_512 : 0}
```

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

def computeFoldValues(input_file_name, output_file_name,
    log_file_name):
    with open(input_file_name) as f:
        resultLines = []# will need to remove all genes with
            fold values of 1, so store a list of all genes
            which do not have that value
        gnuplotLines = []
        genesWithOneFoldRatio = {}
        geneFoldValues = {}
        #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")
        #open the file which will have the lines without one
            ratios; we don't want to append
        out_file_gnuplot = open(log_file_name, "w")

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

        #for every line, compute the fold difference
        for line in content:
                if len(line) <= 2:
                continue
            #we need every integer
            int_strings = line.split(',')
            #the first value is the name of the gene
            geneName = int_strings.pop(0)
            #set the max and min to the opposite end of the
                range
            maxVal = 20
            minVal = 16000
            #compute the maxima and minima of each gene
            for val in int_strings:
                val_int = int(val)
                if(val_int >= maxVal):
                    maxVal = val_int
                if(val_int <= minVal):
                    minVal = val_int
            #compute the fold difference
            currFoldDiff = maxVal / minVal
            #if maxVal eq minVal, then it has a ratio of one
    if maxVal == minVal:
        genesWithOneFoldRatio[geneName] = currFoldDiff
    else:
        resultLines.append(line + "\n")

    #add the computed fold difference to its range
    if currFoldDiff <= 2:
        countFoldDiffPerRange[FOLD_DIFF_LT_2] += 1
    elif currFoldDiff > 2 and currFoldDiff <= 4:
        countFoldDiffPerRange[FOLD_DIFF_2_4] += 1
    elif currFoldDiff > 4 and currFoldDiff <= 8:
        countFoldDiffPerRange[FOLD_DIFF_4_8] += 1
    elif currFoldDiff > 8 and currFoldDiff <= 16:
        countFoldDiffPerRange[FOLD_DIFF_8_16] += 1
    elif currFoldDiff > 16 and currFoldDiff <= 32:
        countFoldDiffPerRange[FOLD_DIFF_16_32] += 1
    elif currFoldDiff > 32 and currFoldDiff <= 64:
        countFoldDiffPerRange[FOLD_DIFF_32_64] += 1
    elif currFoldDiff > 64 and currFoldDiff <= 128:
        countFoldDiffPerRange[FOLD_DIFF_64_128] += 1
    elif currFoldDiff > 128 and currFoldDiff <= 256:
        countFoldDiffPerRange[FOLD_DIFF_128_256] += 1
    elif currFoldDiff > 256 and currFoldDiff <= 512:
        countFoldDiffPerRange[FOLD_DIFF_256_512] += 1
    else:
        countFoldDiffPerRange[FOLD_DIFF_GT_512] += 1

    #store the fold difference in the dictionary
    geneFoldValues[geneName] = currFoldDiff
#end for line in content

#find the largest and smallest fold diffs; also
    compute the range
largestFoldDiff = -1
smallestFoldDiff = 16000000
for key, value in geneFoldValues.items():
    geneName = key
    if(value >= largestFoldDiff):
        largestFoldDiff = value
    if(value <= smallestFoldDiff):
        smallestFoldDiff = value
#now find the number of genes which have these values
    and record them
numGenesWithLargestFoldDiff = 0
numGenesWithSmallestFoldDiff = 0
for key, value in geneFoldValues.items():
    if(value == largestFoldDiff):
        numGenesWithLargestFoldDiff += 1
    if(value == smallestFoldDiff):
        numGenesWithSmallestFoldDiff += 1
        for key, value in countFoldDiffPerRange.items():
            gnuplotLines += key + "\t" + str(value) + "\n"
    #end with open

    debug("largestFoldDiff: " + str(largestFoldDiff))
    debug("smallestFoldDiff: " + str(smallestFoldDiff))
    debug("numGenesWithLargestFoldDiff" + str(
        numGenesWithLargestFoldDiff))
    debug("numGenesWithSmallestFoldDiff" + str(
        numGenesWithSmallestFoldDiff))
    # debug("geneFoldValues (dictionary):\n" + str(
        geneFoldValues))
    # debug("genesWithOneFoldRatio (dictionary):\n" + str(
        genesWithOneFoldRatio))
    # debug("countFoldDiffPerRange (dictionary):\n" + str(
        countFoldDiffPerRange))

    out_file.writelines(resultLines)
    out_file_gnuplot.writelines(gnuplotLines)

    return geneFoldValues #end computeFoldVals

geneFoldValuesMain = computeFoldValues(INPUT_FILE_NAME_TRAIN,
    OUTPUT_FILE_NAME_FOLD_VALUES,
    OUTPUT_FILE_NAME_GENE_DISTRIBUTION)
```

B) The largest fold difference is 800.0, and 17 genes have it.

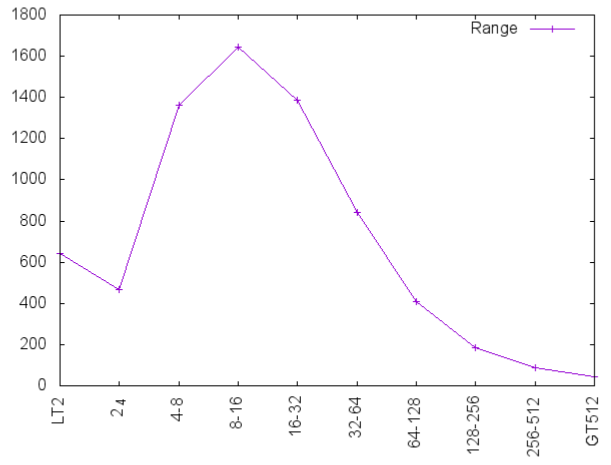
C) The smallest fold difference is 1.0, and 476 genes have it.

D) The distribution is shown below, as a tab-delimited file. (Useful for gnuplot)

ALL_AML_gr.distribution.train.txt

```
LT2      644
2_4      469
4-8      1363
8-16     1643
16-32    1387
32-64    840
64-128   407
128-256  183
```

Figure 1: A graph showing the distribution of genes with fold ratios occurring



in pre-specified ratios

```
256-512  88
GT512    46
```

E) I completed this section with gnuplot

```
set output "distribution.png"
set terminal png
set xtics
set xtics rotate 90
plot "ALL_AML_gr.distribution.train.txt" using 2:
    xticlabels(1) title 'Range' with linespoints
```

Part II
Finding most significant genes

A) I completed this section with a Python script. This script also accomplishes sections B, C, D, and E.

```
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]

        #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))
        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("\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)
```

B)

```
>python3.6m.exe .\calculate-significant-genes.py
> [...]
>Highest signal to noise ratio for ALL: ('
   U22376_cds2_s_at', 1.3393078806073437)
>50th Highest signal to noise ratio for ALL: ('M11722_at
   ', 0.8197361384191656)
>Highest signal to noise ratio for AML: ('M55150_at',
   1.4676411648891123)
>50th highest signal to noise ratio for AML: ('
   M75715_s_at', 0.8119045118112564)
```

The gene with the highest S2N for ALL is 'U22376_cds2_s_at' with a S2N ratio of 1.3393078806073437, and the 50th highest is 'M11722_at' with a ratio of 0.8197361384191656.

The gene with the highest S2N for AML is 'M55150_at' with a ratio of 1.4676411648891123, and the 50th highest is 'M75715_s_at' with a ratio of 0.8119045118112564.

The relationship between Signal-to-Noise ratios is inverse; the gene with the highest S2N ratio for ALL will have the lowest S2N ratio for AML, and vice versa. This is because the numerator of the S2N equation is Avg1-Avg2 or Avg2-Avg1 depending, while the denominator stays the same. This has the effect of reversing the sign of the ratio.

C)

```
>python3.6m.exe .\calculate-significant-genes.py
 >[...]
>Highest T-value for ALL: ('U22376_cds2_s_at', 7.904300374235952)
>50th Highest T-value for ALL: ('U88666_at', 4.8677784184869495)
>Highest T-value for AML: ('M55150_at', 8.091951182855736)
>50th highest T-value for AML: ('D14874_at', 3.8450616652235006)
```

The gene with the highest T-Value for ALL is 'U22376_cds2_s_at' with a T-Value of 7.904300374235952, and the 50th highest is 'U88666_at' with a T-Value of 4.8677784184869495.

The gene with the highest T-Value for AML is 'M55150_at' with a T-Value of 8.091951182855736, and the 50th highest is 'D14874_at' with a T-Value of 3.8450616652235006.

The relationship between T Values is inverse; the gene with the highest T-Value for ALL will have the lowest T-Value for AML, and vice versa. This is because the numerator of the T-Value equation is Avg1-Avg2 or Avg2-Avg1 depending, while the denominator stays the same. This has the effect of reversing the sign of the ratio.

D)

For the ALL calculations, 46 of the top genes are contained in the intersection of the top 50 genes ordered by T-Value, and the top 50 genes ordered by Signal-to-Noise ratio.

```
>intersection of ALL top 3 gene sets selected by S2N
   ratio and T-Value: { 'X52142_at', 'X59417_at', '
   U22376_cds2_s_at' }
```

All three of the top genes are in the intersection of the two sets of genes (top 50 by T-Value, top 50 by S2N). They are 'X52142_at', 'X59417_at', and 'U22376_cds2_s_at'.

E)

For the AML calculations, 38 of the top genes are contained in the intersection of the top 50 genes ordered by T-Value, and the top 50 genes ordered by Signal-to-Noise ratio.

```
>intersection of AML top 3 gene sets selected by S2N
   ratio and T-Value: { 'U50136_rna1_at', 'M55150_at' }
```

Two of the top three genes are in the intersection of the two sets of genes (top 50 by T-Value, top 50 by S2N). They are 'U50136_rna1_at' and 'M55150_at'.

Part III
Lessons Learned

First and foremost, I have learned that data preparation is a time- and labor-intensive effort. Maybe it takes longer for me to accomplish certain goals because I am weak with Linux command line tools, so I tend to use Python scripts for everything, but the data prep still seems to take forever. Even something as basic as normalizing the expression values and printing it out to a new file takes a hand-written script (for me). In terms of feature selection, I have definitely learned that one needs to carefully screen features before selecting them for use by classification algorithms. For instance, "Source" turns out to be an awful feature for classifiers to use when trying to mine data about cancer, because a patient develops cancer before they arrive at the "source hospital".