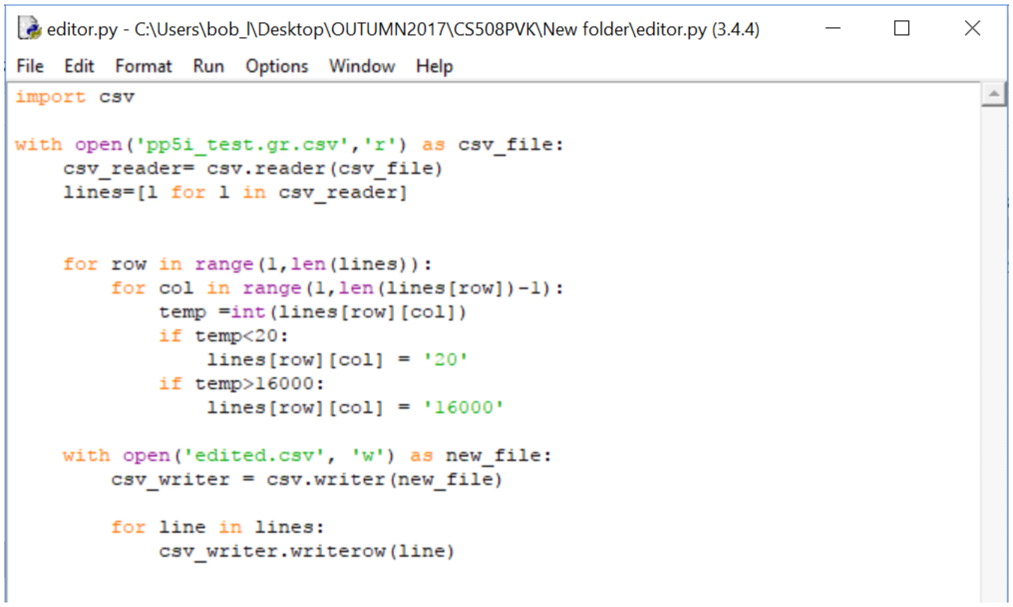
Sean Reeves

Jose Jarquin

Jeremy Keys

(1)

We completed section 1 by re-using the normalize-data script we each had to write for HW5.


editor.py - C:\Users\bob_l\Desktop\OUTUMN2017\CS508PVK\New folder\editor.py (3.4.4)
File Edit Format Run Options Window Help

```python
import csv

with open('pp5i_test.gr.csv','r') as csv_file:
    csv_reader= csv.reader(csv_file)
    lines=[l for l in csv_reader]


    for row in range(1,len(lines)):
        for col in range(1,len(lines[row])-1):
            temp =int(lines[row][col])
            if temp<20:
                lines[row][col] = '20'
            if temp>16000:
                lines[row][col] = '16000'

    with open('edited.csv', 'w') as new_file:
        csv_writer = csv.writer(new_file)

        for line in lines:
            csv_writer.writerow(line)
```

Figure 1 A simple normalize script

(2)

For the first section of step (2), we re-purposed the fold calculating scripts we had to write. (For this part, we used Sean's script, which is written in R.)

```r
value <- read.csv(file =
"C:/Users/Sreav/Desktop/FinalData/pp5i_train.gr.normalized.removeless2.csv", head = TRUE,
sep=",")

str(value)
sumsq <-
 function(X){
  # sum of squares of input object

  if(is.list(X)){
   xss <- sum(sapply(X,sumsq))
  } else {
   xss <- sum(X^2)
  }

  xss
 }

for(i in 1:nrow(value)){
   avMED <- rowMeans(value[i,2:40])
   sumMED <- rowSums(value[i,2:40])
   sumSQMED <- sumsq(value[i,2:40])
   stdevMED <- sqrt((39*sumSQMED - sumMED*sumMED)/(39*(39-1)))

   avMGL <- rowMeans(value[i,41:47])
   sumMGL <- rowSums(value[i,41:47])
   sumSQMGL <- sumsq(value[i,41:47])
   stdevMGL <- sqrt((7*sumSQMGL - sumMGL *sumMGL)/(7*(7-1)))

   avRHB <- rowMeans(value[i,48:54])
   sumRHB <- rowSums(value[i,48:54])
   sumSQRHB <- sumsq(value[i,48:54])
   stdevRHB <- sqrt((7*sumSQRHB - sumRHB *sumRHB )/(7*(7-1)))

   avEPD <- rowMeans(value[i,55:64])
   sumEPD <- rowSums(value[i,55:64])
   sumSQEPD <- sumsq(value[i,55:64])
   stdevEPD <- sqrt((10*sumSQEPD - sumEPD *sumEPD )/(10*(10-1)))

   avJPA <- rowMeans(value[i,65:70])
   sumJPA <- rowSums(value[i,65:70])
   sumSQJPA <- sumsq(value[i,65:70])
   stdevJPA <- sqrt((6*sumSQJPA - sumJPA *sumJPA )/(6*(6-1)))

   avOtherMED <- (avMGL + avRHB + avEPD + avJPA)/4
   avOtherMGL <- (avMED + avRHB + avEPD + avJPA)/4
   avOtherRHB <- (avMED + avMGL + avEPD + avJPA)/4
   avOtherEPD <- (avMED + avMGL + avRHB + avJPA)/4
   avOtherJPA <- (avMED + avMGL + avRHB + avEPD)/4

   sumOtherMED <- sumMGL + sumRHB + sumEPD + sumJPA
   sumOtherMGL <- sumMED + sumRHB + sumEPD + sumJPA
   sumOtherRHB <- sumMGL + sumMED + sumEPD + sumJPA
   sumOtherEPD <- sumMGL + sumMED + sumRHB + sumJPA
   sumOtherJPA <- sumMGL + sumMED + sumRHB + sumEPD

   sumSQOtherMED <- sumSQMGL + sumSQRHB + sumSQEPD + sumSQJPA
   sumSQOtherMGL <- sumSQMED + sumSQRHB + sumSQEPD + sumSQJPA
   sumSQOtherRHB <- sumSQMGL + sumSQMED + sumSQEPD + sumSQJPA
   sumSQOtherEPD <- sumSQMGL + sumSQMED + sumSQRHB + sumSQJPA
   sumSQOtherJPA <- sumSQMGL + sumSQMED + sumSQRHB + sumSQEPD

   stdevOtherMED <- sqrt((30*sumSQOtherMED - sumOtherMED * sumOtherMED)/(30*(30-1)))
```

```
    stdevOtherMGL <- sqrt((62*sumSQOtherMGL - sumOtherMGL * sumOtherMGL)/(62*(62-1)))
    stdevOtherRHB <- sqrt((62*sumSQOtherRHB - sumOtherRHB * sumOtherRHB)/(62*(62-1)))
    stdevOtherEPD <- sqrt((59*sumSQOtherEPD - sumOtherEPD * sumOtherEPD)/(59*(59-1)))
    stdevOtherJPA <- sqrt((63*sumSQOtherJPA - sumOtherJPA * sumOtherJPA)/(63*(63-1)))

     #TvalueALL <- (avALL - avAML)/sqrt(stdevALL*stdevALL / 27 + stdevAML*stdevAML / 11)
     #TvalueAML <- (avAML - avALL)/sqrt(stdevAML*stdevAML / 11 + stdevALL*stdevALL / 27)
     TvalueMED <- (avMED - avOtherMED)/sqrt(stdevMED*stdevMED / 39 +
  stdevOtherMED*stdevOtherMED / 30)
     TvalueMED <- abs(TvalueMED)
     TvalueMGL <- (avMGL - avOtherMGL)/sqrt(stdevMGL*stdevMGL / 7 +
  stdevOtherMGL*stdevOtherMGL / 62)
     TvalueMGL <- abs(TvalueMGL)
     TvalueRHB <- (avRHB - avOtherRHB)/sqrt(stdevRHB*stdevRHB / 7 +
  stdevOtherRHB*stdevOtherRHB / 62)
     TvalueRHB <- abs(TvalueRHB)
     TvalueEPD <- (avEPD - avOtherEPD)/sqrt(stdevEPD*stdevEPD / 10 +
  stdevOtherEPD*stdevOtherEPD / 59)
     TvalueEPD <- abs(TvalueEPD)
     TvalueJPA <- (avJPA - avOtherJPA)/sqrt(stdevJPA*stdevJPA / 6 +
  stdevOtherJPA*stdevOtherJPA / 63)
     TvalueJPA <- abs(TvalueJPA)

     write(TvalueMED,file="C:/Users/Sreav/Desktop/FinalData/TvalueMED.txt",append=TRUE)
     write(TvalueMGL,file="C:/Users/Sreav/Desktop/FinalData/TvalueMGL.txt",append=TRUE)
     write(TvalueRHB,file="C:/Users/Sreav/Desktop/FinalData/TvalueRHB.txt",append=TRUE)
     write(TvalueEPD,file="C:/Users/Sreav/Desktop/FinalData/TvalueEPD.txt",append=TRUE)
     write(TvalueJPA,file="C:/Users/Sreav/Desktop/FinalData/TvalueJPA.txt",append=TRUE)


  }
```

After generating the top 30 genes for each class sorted in descending order, we generated the subsets by using the `head` command on Linux to retain the first `N` lines (genes) for each subset (2,4,6,8,10,12,15,20,25) and each class, `cat` to merge the resulting subset files for each N, and 'gawk' to keep only the unique lines for each subset in their original order. E.g.:

```
    head -25 Top30MED.csv > Top25MED.csv
    cat Top25MED Top25JPA.csv Top25MGL.csv Top25RHB.csv Top25EPD.csv > Top25-not-
unique.csv
    gawk '!seen[$0]++' Top25-not-unique.csv
    cat Top25-not-unique.csv | awk '!seen[$0]++' > Top25-unique.csv
```

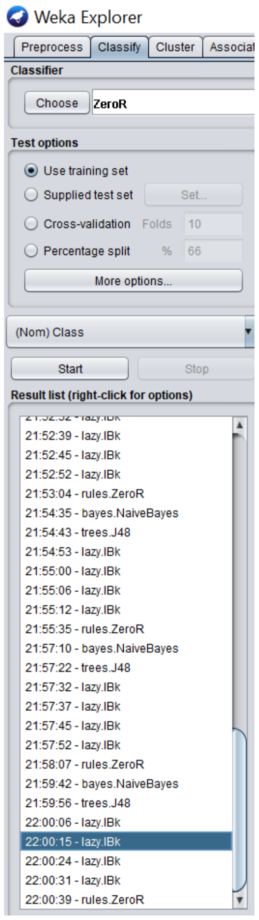
Finally, we transposed the class file we were given, and manually added it to the last line of each top-N-unique files, then transposed each file again to get it in genes-in-columns format.

```
#https://unix.stackexchange.com/questions/60590/is-there-a-command-
line-utility-to-transpose-a-csv-file

import csv, sys

#constants
DEBUG = True
INPUT_FILE_NAME_TRAIN_30 = "pp5i_train.top30.gr.unique.csv"
OUTPUT_FILE_NAME_TRAIN_30 = "pp5i_train.top30.gr.unique.transposed.csv"

def transposeRows(input_file_name, output_file_name):
    with open(input_file_name) as file:
        out_file = open(output_file_name, "w")
        rows = list(csv.reader(file))
        writer = csv.writer(out_file)
        for col in range(0, len(rows[0])):
            writer.writerow([row[col] for row in rows])

transposeRows(INPUT_FILE_NAME_TRAIN_30, OUTPUT_FILE_NAME_TRAIN_30)
```

(3)

For part 3 we selected the classifier ZeroR at the beginning, but after measuring the classifier's accuracy, we found out that with this classifier, we would get the same value for all the previously generated files and we decided to change it to a more accurate classifier.

The one we ended up selecting is the Multilayer Perceptron classifier.



Classifier: ZeroR

Test options:
- [✓ Use training set]
- Supplied test set
- Cross-validation Folds 10
- Percentage split % 66

(Nom) Class

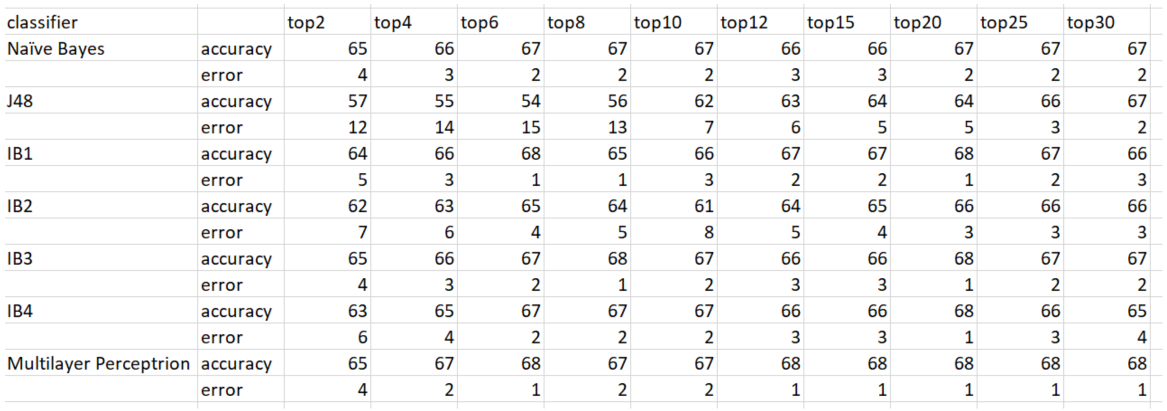
Result list (right-click for options)

```
21:52:32 - lazy.IBk
21:52:39 - lazy.IBk
21:52:45 - lazy.IBk
21:52:52 - lazy.IBk
21:53:04 - rules.ZeroR
21:54:35 - bayes.NaiveBayes
21:54:43 - trees.J48
21:54:53 - lazy.IBk
21:55:00 - lazy.IBk
21:55:06 - lazy.IBk
21:55:12 - lazy.IBk
21:55:35 - rules.ZeroR
21:57:10 - bayes.NaiveBayes
21:57:22 - trees.J48
21:57:32 - lazy.IBk
21:57:37 - lazy.IBk
21:57:45 - lazy.IBk
21:57:52 - lazy.IBk
21:58:07 - rules.ZeroR
21:59:42 - bayes.NaiveBayes
21:59:56 - trees.J48
22:00:06 - lazy.IBk
22:00:15 - lazy.IBk
22:00:24 - lazy.IBk
22:00:31 - lazy.IBk
22:00:39 - rules.ZeroR
```

Figure 2 All the models we got from calculating the accuracy

3b)

After this we calculated the cross-validation error rate the same way as before, but this time instead of using the training set, we used cross-validation with 10 folds. (See Figure 2.)



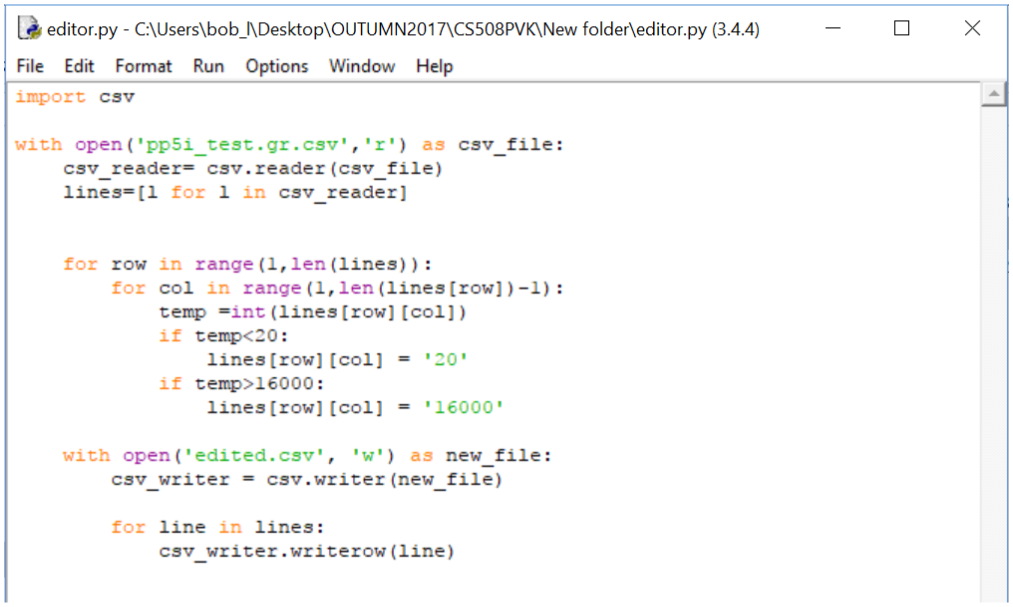
classifier		top2	top4	top6	top8	top10	top12	top15	top20	top25	top30
Naïve Bayes	accuracy	65	66	67	67	67	66	66	67	67	67
	error	4	3	2	2	2	3	3	2	2	2
J48	accuracy	57	55	54	56	62	63	64	64	66	67
	error	12	14	15	13	7	6	5	5	3	2
IB1	accuracy	64	66	68	65	66	67	67	68	67	66
	error	5	3	1	1	3	2	2	1	2	3
IB2	accuracy	62	63	65	64	61	64	65	66	66	66
	error	7	6	4	5	8	5	4	3	3	3
IB3	accuracy	65	66	67	68	67	66	66	68	67	67
	error	4	3	2	1	2	3	3	1	2	2
IB4	accuracy	63	65	67	67	67	66	66	68	66	65
	error	6	4	2	2	2	3	3	1	3	4
Multilayer Perceptrion	accuracy	65	67	68	67	67	68	68	68	68	68
	error	4	2	1	2	2	1	1	1	1	1

Figure 3 Results obtained by running 10-fold cross-validation on each classifier.

With this information, we decided we would select Multilayer Perceptrion because this it tied for the most accurate classifier for all of the files (6 out of 10 have error rate 1/69) and gene set of top 20 for the same reason (4 out of 7 have error rate 1/69).

3c)

Then, with the test set, we used a python program that converted all the values smaller than 20 to 20 and the values larger that 1600 to 1600.

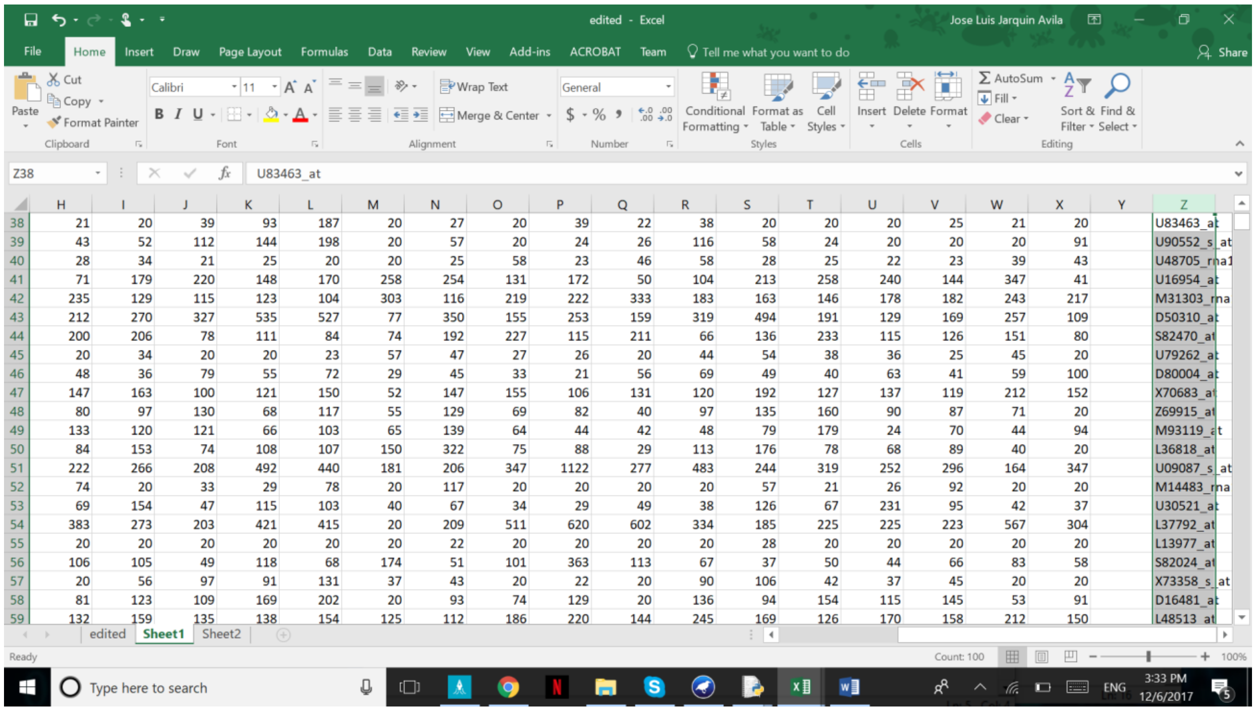

editor.py - C:\Users\bob_l\Desktop\OUTUMN2017\CS508PVK\New folder\editor.py (3.4.4)
File Edit Format Run Options Window Help

```python
import csv

with open('pp5i_test.gr.csv','r') as csv_file:
    csv_reader= csv.reader(csv_file)
    lines=[l for l in csv_reader]


    for row in range(1,len(lines)):
        for col in range(1,len(lines[row])-1):
            temp =int(lines[row][col])
            if temp<20:
                lines[row][col] = '20'
            if temp>16000:
                lines[row][col] = '16000'

    with open('edited.csv', 'w') as new_file:
        csv_writer = csv.writer(new_file)

        for line in lines:
            csv_writer.writerow(line)
```

Figure 4 A simpler normalize script

3d)

To get the "top 20 train set" names so we could extract the data from the test set, we selected the "top 20 train set" data and transposed it. After that, we cut the names of the top 20 and pasted them next to the original test set file (see Figure 6).


Z38 | U83463_at

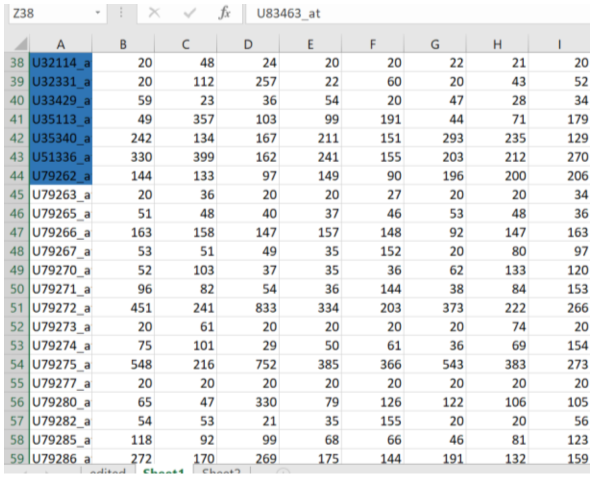
	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
38	21	20	39	93	187	20	27	20	39	22	38	20	20	20	25	21	20		U83463_at
39	43	52	112	144	198	20	57	20	24	26	116	58	24	20	20	20	91		U90552_s_at
40	28	34	21	25	20	20	25	58	23	46	58	28	25	22	23	39	43		U48705_rna1
41	71	179	220	148	170	258	254	131	172	50	104	213	258	240	144	347	41		U16954_at
42	235	129	115	123	104	303	116	219	222	333	183	163	146	178	182	243	217		M31303_rna
43	212	270	327	535	527	77	350	155	253	159	319	494	191	129	169	257	109		D50310_at
44	200	206	78	111	84	74	192	227	115	211	66	136	233	115	126	151	80		S82470_at
45	20	34	20	20	23	57	47	27	26	20	44	54	38	36	25	45	20		U79262_at
46	48	36	79	55	72	29	45	33	21	56	69	49	40	63	41	59	100		D80004_at
47	147	163	100	121	150	52	147	155	106	131	120	192	127	137	119	212	152		X70683_at
48	80	97	130	68	117	55	129	69	82	40	97	135	160	90	87	71	20		Z69915_at
49	133	120	121	66	103	65	139	64	44	42	48	79	179	24	70	44	94		M93119_at
50	84	153	74	108	107	150	322	75	88	29	113	176	78	68	89	40	20		L36818_at
51	222	266	208	492	440	181	206	347	1122	277	483	244	319	252	296	164	347		U09087_s_at
52	74	20	33	29	78	20	117	20	20	20	20	57	21	26	92	20	20		M14483_rna
53	69	154	47	115	103	40	67	34	29	49	38	126	67	231	95	42	37		U30521_at
54	383	273	203	421	415	20	209	511	620	602	334	185	225	225	223	567	304		L37792_at
55	20	20	20	20	20	20	22	20	20	20	20	28	20	20	20	20	20		L13977_at
56	106	105	49	118	68	174	51	101	363	113	67	37	50	44	66	83	58		S82024_at
57	20	56	97	91	131	37	43	20	22	20	90	106	42	37	45	20	20		X73358_s_at
58	81	123	109	169	202	20	93	74	129	20	136	94	154	115	145	53	91		D16481_at
59	132	159	135	138	154	125	112	186	220	144	245	169	126	170	158	212	150		L48513_at

Figure 5 Selecting unique genes

We used an Excel formula to compare two arbitrary columns and highlight the names they have in common (the intersection of the two). First, we selected the A column as our target and the recently pasted names as out search and used the formula

=countif($Z:$Z,$A1)

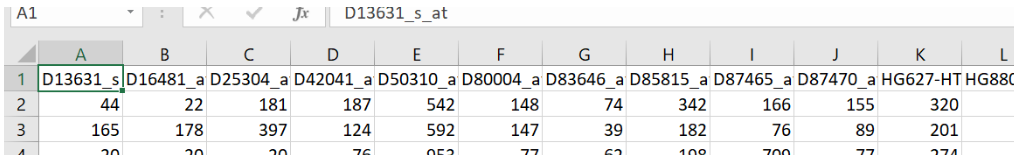
$Z:$Z represents the entire Z (gene name) column, which is where our train set data is located, and $A represents the row where we will search. (A1 tells the function to begin at row 1.) The reason why 1 does not have a $ before it is that without it, the formula will continue to A2 and so on.


Z38 | U83463_at

	A	B	C	D	E	F	G	H	I
38	U32114_a	20	48	24	20	20	22	21	20
39	U32331_a	20	112	257	22	60	20	43	52
40	U33429_a	59	23	36	54	20	47	28	34
41	U35113_a	49	357	103	99	191	44	71	179
42	U35340_a	242	134	167	211	151	293	235	129
43	U51336_a	330	399	162	241	155	203	212	270
44	U79262_a	144	133	97	149	90	196	200	206
45	U79263_a	20	36	20	20	27	20	20	34
46	U79265_a	51	48	40	37	46	53	48	36
47	U79266_a	163	158	147	157	148	92	147	163
48	U79267_a	53	51	49	35	152	20	80	97
49	U79270_a	52	103	37	35	36	62	133	120
50	U79271_a	96	82	54	36	144	38	84	153
51	U79272_a	451	241	833	334	203	373	222	266
52	U79273_a	20	61	20	20	20	20	74	20
53	U79274_a	75	101	29	50	61	36	69	154
54	U79275_a	548	216	752	385	366	543	383	273
55	U79277_a	20	20	20	20	20	20	20	20
56	U79280_a	65	47	330	79	126	122	106	105
57	U79282_a	54	53	21	35	155	20	20	56
58	U79285_a	118	92	99	68	66	46	81	123
59	U79286_a	272	170	269	175	144	191	132	159

Figure 6 Removing extraneous genes with Excel

After the similar values were selected, we removed the ones that were not underlined. Finally, we transposed the data to have the names in columns by copying it and using “paste special” in Excel.

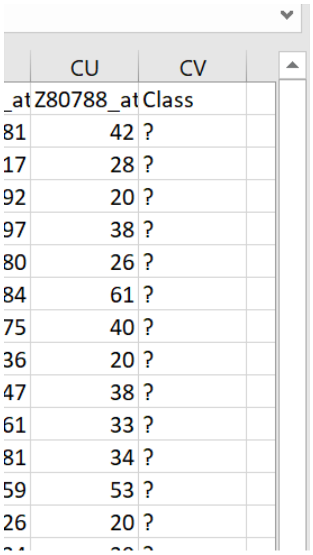


A1 | *fx* D13631_s_at

	A	B	C	D	E	F	G	H	I	J	K	L
1	D13631_s	D16481_a	D25304_a	D42041_a	D50310_a	D80004_a	D83646_a	D85815_a	D87465_a	D87470_a	HG627-HT	HG88
2	44	22	181	187	542	148	74	342	166	155	320	
3	165	178	397	124	592	147	39	182	76	89	201	

Figure 7 transposing the genes to genes-in-columns format

3e) We manually added the class with “?” values in Excel.



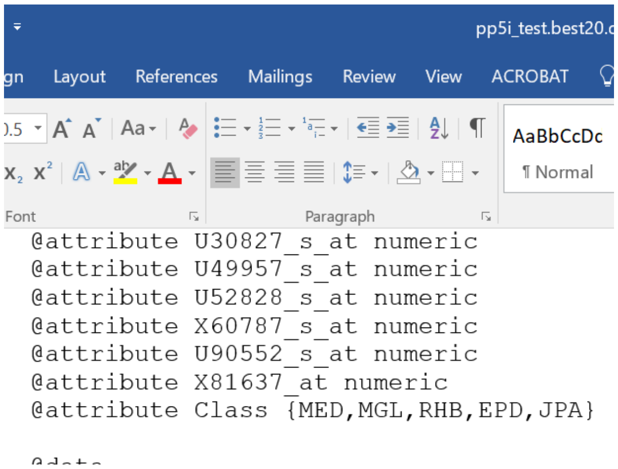
	CU	CV
_at	Z80788_at	Class
81	42	?
17	28	?
92	20	?
97	38	?
80	26	?
84	61	?
75	40	?
36	20	?
47	38	?
61	33	?
81	34	?
59	53	?
26	20	?

Figure 8 Inserting the ? column

(4) Generate Predictions For the Test Set

a) The test file was converted to ARFF format by opening it in weka and then saving it.

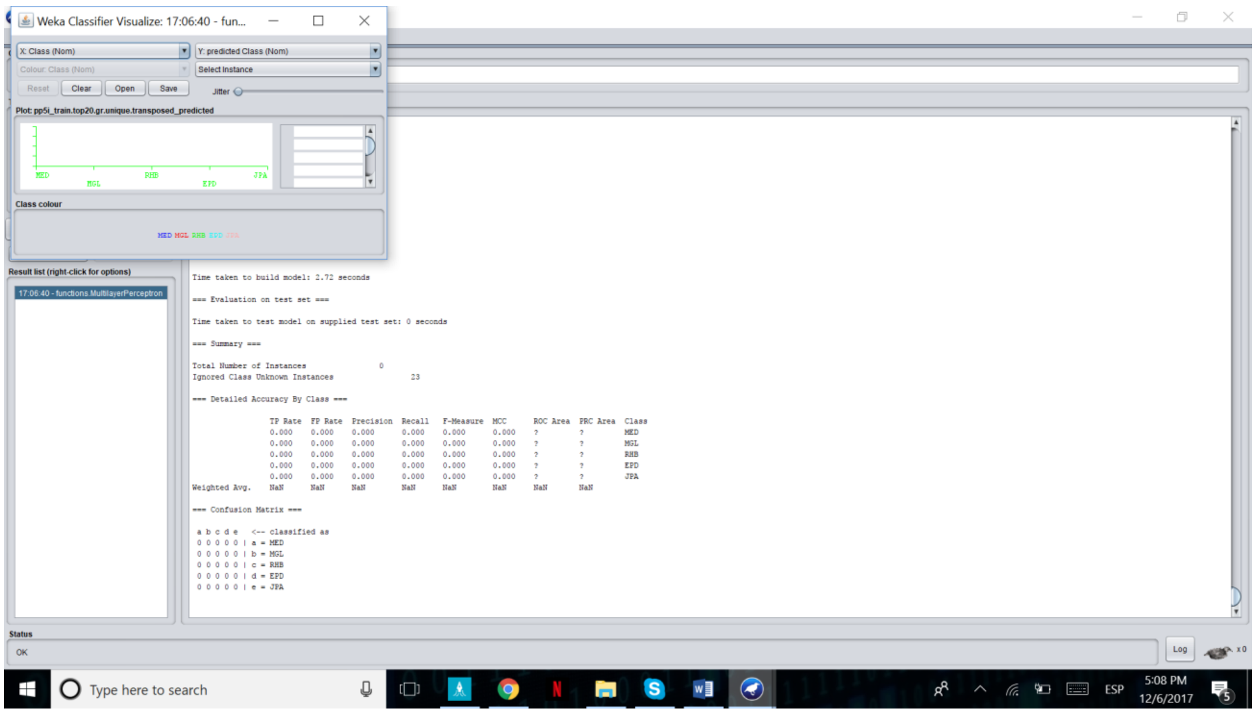
B0We used Word to change the class line in test file to match the train file.



```
@attribute U30827_s_at numeric
@attribute U49957_s_at numeric
@attribute U52828_s_at numeric
@attribute X60787_s_at numeric
@attribute U90552_s_at numeric
@attribute X81637_at numeric
@attribute Class {MED,MGL,RHB,EPD,JPA}
```

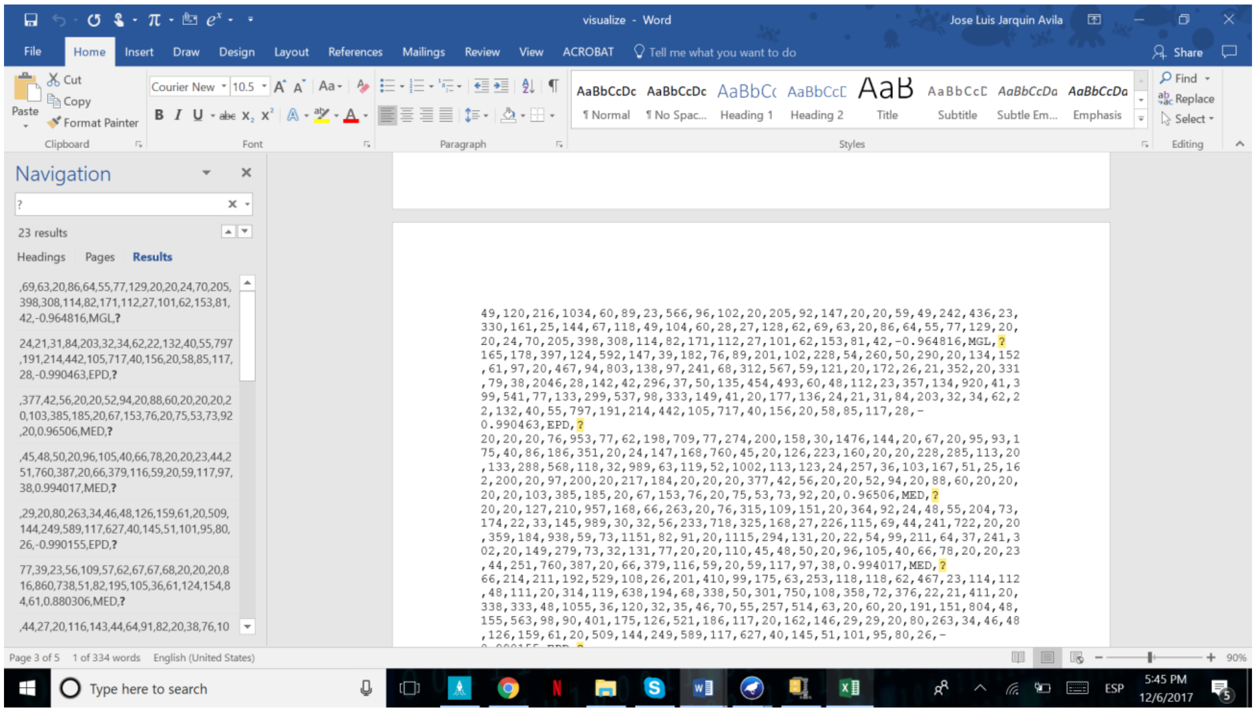
Figure 9 Adding the matching attribute line from the train to the test file

Per the assignment instructions, we used Weka to generate predictions by using its Preprocess>Classify tab. We added the predictions with Excel, and the *?* class by hand.



```
Time taken to build model: 2.72 seconds

=== Evaluation on test set ===

Time taken to test model on supplied test set: 0 seconds

=== Summary ===

Total Number of Instances                0
Ignored Class Unknown Instances                 23

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.000    0.000    0.000      0.000    0.000      0.000    ?         ?         MED
                 0.000    0.000    0.000      0.000    0.000      0.000    ?         ?         MGL
                 0.000    0.000    0.000      0.000    0.000      0.000    ?         ?         RHB
                 0.000    0.000    0.000      0.000    0.000      0.000    ?         ?         EPD
                 0.000    0.000    0.000      0.000    0.000      0.000    ?         ?         JPA
Weighted Avg.    NaN      NaN      NaN        NaN      NaN        NaN      NaN       NaN

=== Confusion Matrix ===

 a b c d e   <-- classified as
 0 0 0 0 0 | a = MED
 0 0 0 0 0 | b = MGL
 0 0 0 0 0 | c = RHB
 0 0 0 0 0 | d = EPD
 0 0 0 0 0 | e = JPA
```

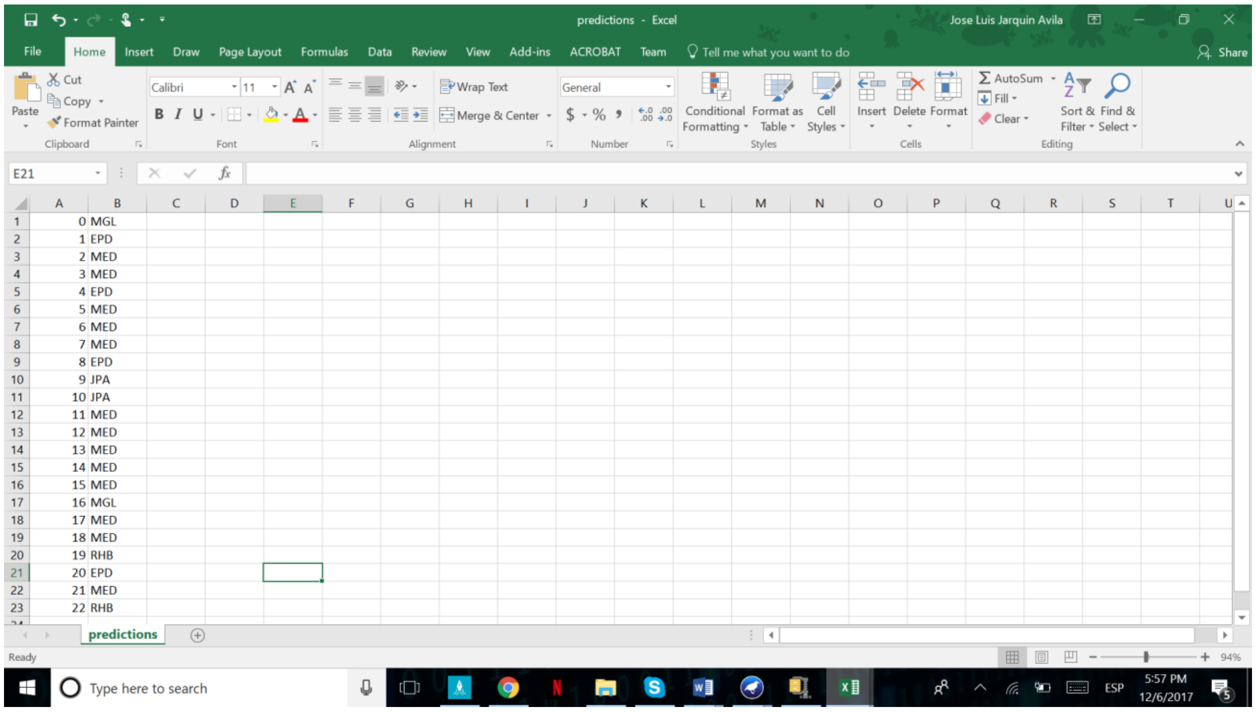
Figure 10 Weka's Visualize feature



```
,69,63,20,86,64,55,77,129,20,20,24,70,205,
398,308,114,82,171,112,27,101,62,153,81,
42,-0.964816,MGL,?

24,21,31,84,203,32,34,62,22,132,40,55,797
,191,214,442,105,717,40,156,20,58,85,117,
28,-0.990463,EPD,?

,377,42,56,20,20,52,94,20,88,60,20,20,20,2
0,103,385,185,20,67,153,76,20,75,53,73,92
,20,0.96506,MED,?

,45,48,50,20,96,105,40,66,78,20,20,23,44,2
51,760,387,20,66,379,116,59,20,59,117,97,
38,0.994017,MED,?

,29,20,80,263,34,46,48,126,159,61,20,509,
144,249,589,117,627,40,145,51,101,95,80,
26,-0.990155,EPD,?

77,39,23,56,109,57,62,67,67,68,20,20,20,8
16,860,738,51,82,195,105,36,61,124,154,8
4,61,0.880306,MED,?

,44,27,20,116,143,44,64,91,82,20,38,76,10
```

```
49,120,216,1034,60,89,23,566,96,102,20,205,92,147,20,20,59,49,242,436,23,
330,161,25,144,67,118,49,104,60,28,27,128,62,69,63,20,86,64,55,77,129,20,
20,24,70,205,398,308,114,82,171,112,27,101,62,153,81,42,-0.964816,MGL,?
165,178,397,124,592,147,39,182,76,89,201,102,228,54,260,50,290,20,134,152
,61,97,20,467,94,803,138,97,241,68,312,567,59,121,20,172,26,21,352,20,331
,79,38,2046,28,142,42,296,37,50,135,454,493,60,48,112,23,357,134,920,41,3
99,541,77,133,299,537,98,333,149,41,20,177,136,24,21,31,84,203,32,34,62,2
2,132,40,55,797,191,214,442,105,717,40,156,20,58,85,117,28,-
0.990463,EPD,?
20,20,20,76,953,77,62,198,709,77,274,200,158,30,1476,144,20,67,20,95,93,1
75,40,86,186,351,20,24,147,168,760,45,20,126,223,160,20,20,228,285,113,20
,133,288,568,118,32,989,63,119,52,1002,113,123,24,257,36,103,167,51,25,16
2,200,20,97,200,20,217,184,20,20,20,377,42,56,20,20,52,94,20,88,60,20,20,
20,20,103,385,185,20,67,153,76,20,75,53,73,92,20,0.96506,MED,?
20,20,127,210,957,168,66,263,20,76,315,109,151,20,364,92,24,48,55,204,73,
174,22,33,145,989,30,32,56,233,718,325,168,27,226,115,69,44,241,722,20,20
,359,184,938,59,73,1151,82,91,20,1115,294,131,20,22,54,99,211,64,37,241,3
02,20,149,279,73,32,131,77,20,20,110,45,48,50,20,96,105,40,66,78,20,20,23
,44,251,760,387,20,66,379,116,59,20,59,117,97,38,0.994017,MED,?
66,214,211,192,529,108,26,201,410,99,175,63,253,118,118,62,467,23,114,112
,48,111,20,314,119,638,194,68,338,50,301,750,108,358,72,376,22,21,411,20,
338,333,48,1055,36,120,32,35,46,70,55,257,514,63,20,60,20,191,151,804,48,
155,563,98,90,401,175,126,521,186,117,20,162,146,29,29,20,80,263,34,46,48
,126,159,61,20,509,144,249,589,117,627,40,145,51,101,95,80,26,-
```

Figure 11 Adding the ? class



	A	B
1	0	MGL
2	1	EPD
3	2	MED
4	3	MED
5	4	EPD
6	5	MED
7	6	MED
8	7	MED
9	8	EPD
10	9	JPA
11	10	JPA
12	11	MED
13	12	MED
14	13	MED
15	14	MED
16	15	MED
17	16	MGL
18	17	MED
19	18	MED
20	19	RHB
21	20	EPD
22	21	MED
23	22	RHB

Figure 12 Getting the predicted classes in a column for easy transfer to the test file

(5) (Recursively) Write a paper describing your paper.

ZeroR is the simplest possible classifier. It selects the class that appears most often in the set.

IBk is Weka's implementation of the k-Nearest Neighbors classifier. 1-Nearest Neighbors (IB1) is also a simple classifier, by selecting the class of the closest training sample when given a test sample. 2-NN (IB2) and k-NN (IBk) generalizes this classifier by selecting the majority class of the N neighbors that lie closest to the test instance. In general, odd numbers of N are preferred to avoid ties.

Naïve Bayes is a simple classifier that operates on Bayes Theorem using the unrealistic (or at least generous) assumption that the classes in the data set are independent (hence Naïve). Bayes Theorem states that

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})},$$

(courtesy Wikipedia) The term on the left-hand side is the posteriori probability, X is the evidence, and Ck are the classes. The Naïve Bayes classifier is thus constructed by making a frequency table of classes for each class, and taking the product of them divided by the likelihood of that class overall. A Laplace modifier is also usually added to ensure there is no class with zero probability. Test samples are classified by calculating the probability of the sample being a given class and the probability of the sample not being that class.

J48 is a decision-tree classifier. It works by recursively calculating the information gain of splitting on each class and constructing a tree. Once constructed, J48 classifies test samples by recursively walking down the tree and choosing to move to each child node that has the highest information gain (difference of entropy). This means that J48 uses the most predictive classes at every level of the tree to classify the instance.

Finally, we chose the top-20 unique genes from the classes, along with the Multilayer Perceptron, for our classifier. According to Weka, the multilayer perceptron is a classifier "that uses backpropagation to classify instances." Multilayer perceptrons have 3 layers, an input layer, an output layer, and a hidden layer. The train set is used to build the hidden layer, and then test inputs are fed into the input layer to classify the instances.

Below is a graph showing the incorrectly classified samples by class and gene subset size.

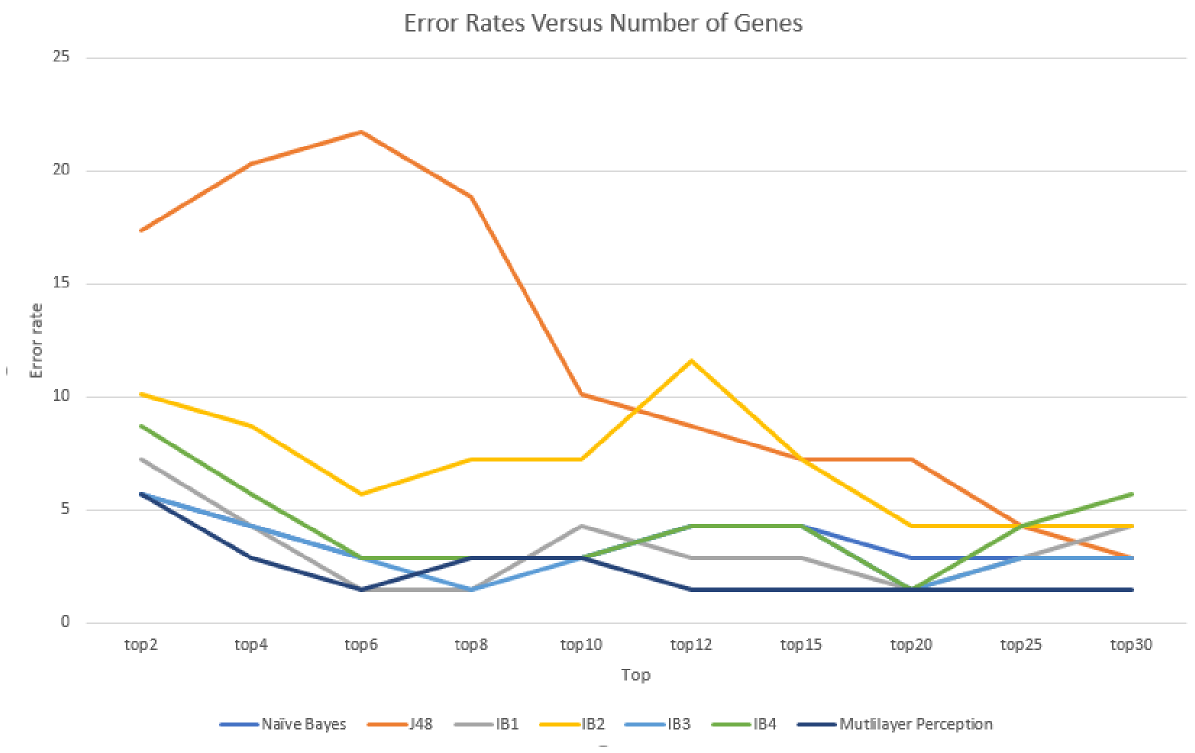


Figure 13 Graph of each classifier's error rate against the top-n subsets generated.