Alex Shung
Netid: ashung2
Credits: 3
Assignment 3

Part 1 Implementation

Training

During training, the numbers were read in from the text file and converted to a 2d numpy matrix of integers where a 0 represented whitespace, and a 1 represented everything else. The labels were also read in, and the count of each label was taken using Python's Counter, which returns a dictionary from the value it discovered to the number of times it discovered. This was essentially the number of instances that any class was seen in the training set. Class priors were built by dividing the number of labels provided for that class by the total number of labels provided, and stored in a map from the class to the prior.

A class to features map was then initialized with a smoothing constant. The map was initialized to a 2d 28 x 28 numpy matrix where each cell contained the smoothing constant provided (0.1). Then for each number represented as a matrix read in from the text file, the label was used as the key to obtain the 2d features matrix associated with that class, and this number was added to the features. The end result is a map which has the class as the key and the feature matrix which has the number of times any particular pixel was 1. This feature matrix was then divided by the number of instances that the class appeared in, from the classCounter described previously, plus 2 as part of the smoothing constant as the values that could be taken were either 1 or 0. The feature matrix now represented the probability that any given pixel was turned on across the training data for that particular class, and was thus the likelihood estimates for the class (however not the log likelihood). This map of key to feature matrix / likelihood matrix and the class priors were passed in to the next phase: Testing.

The best smoothing constant was identified using findBestSmoothingConstant, which for a range from 0.1 to 2.0 in 0.2 increments set the value of the smoothing constant. The class to features map was then initialized with that constant, and then testing was done on the training data to see how many it correctly classified. The smoothing constant which produced the greatest number of correct classifications was selected. This way the testing data was completely shielded from the hyper parameter tuning. I could / should have used a hold out set, where an additional portion of the training data was set aside to tune this hyper parameter during the testing phase. The smoothing constant which produced the greatest number of correct classifications on the training data was selected as the hyper parameter, and that seemed to be 0.1.

Testing

The test set was read in the same way as the training data. For each number in the read in numbers, it was compared against every class. For every class comparison, the probability of each pixel matching with the class was the value of the likelihood matrix at that particular pixel's location if that pixel was turned on / was a 1. If the pixel was not, or a 0, then the probability was one minus the value of the likelihood matrix at that location. The log of this probability was summed across all pixels, and the log of the class prior was also added. The maximum probability across all classes was then used to identify the associated classification of the number (eg. If the probability of 0 was .8, and the probability of 1 was .5, the number would be classified as 0). The resulting classifications were then passed into the next stage: Evaluation.

Evaluation

The test labels were read in from the file, and compared to the results obtained from testing. A confusion matrix was created with the expected labels as the rows, and the actual / tested values as the columns. The classification for each class was then obtained from the classification matrix, and defined as the diagonal entry divided by the sum across the row. The results are presented in the Results section.

Best Worst Numbers for Class

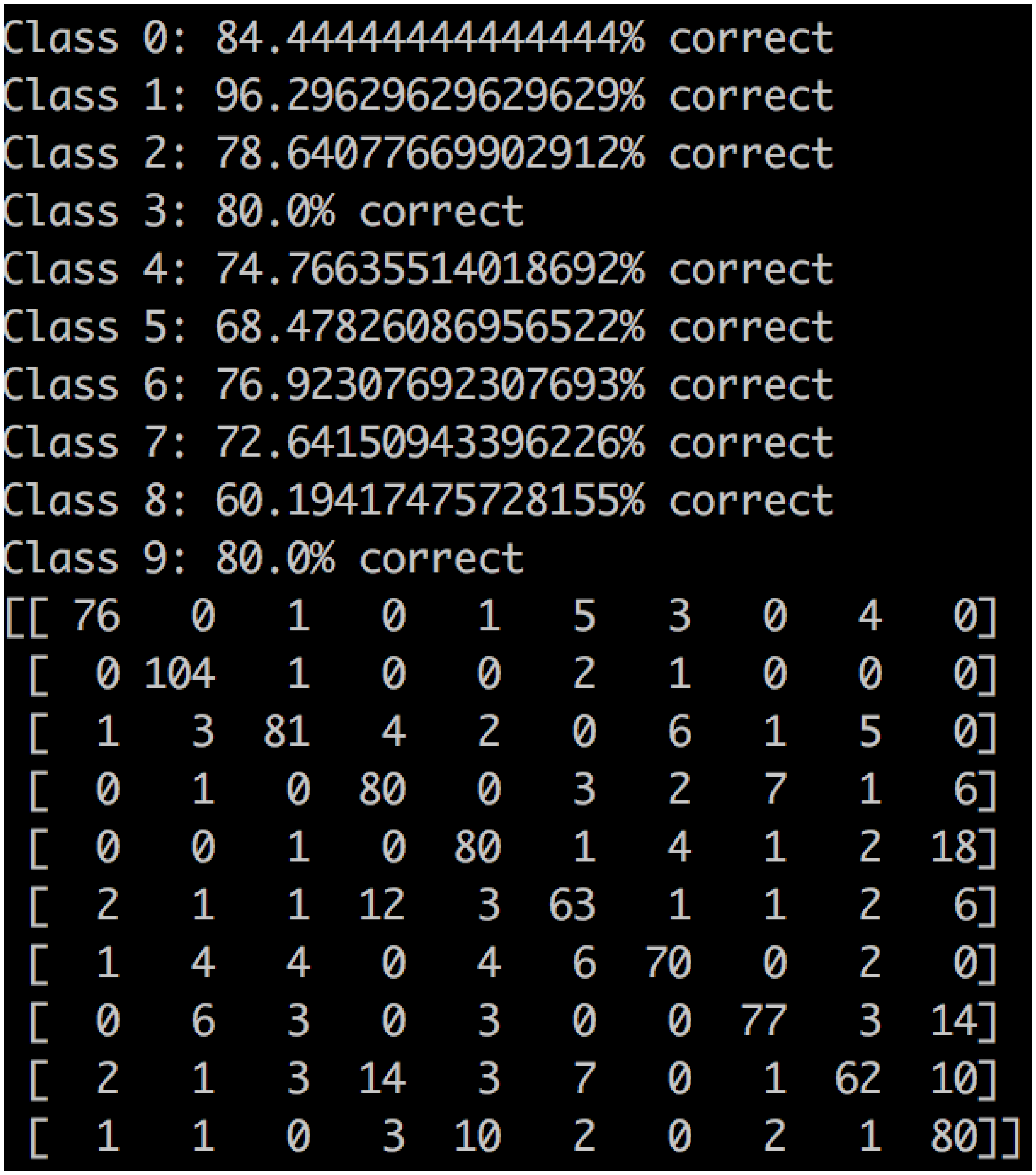
During testing, the numbers which produced the best and worst probabilities for each class were stored in a map. The map could then be used to identify and view the best and worst cases for any particular class.

Odds Ratio

The four worst cases according to my confusion matrix was 4 as 9, 18 times, 7 as 9, 14 times, 8 as 3, 14 times, and 5 as 3, 12 times. The results section shows a color map of each digit and the first digit over the second digit. These were printed using python’s matplotlib. Each map represents the log of the probabilities, and the map of one number over another number is represented by the log(Prob(num1)) – log(Prob(num2)).

Results

Classification Rate and Confusion Matrix

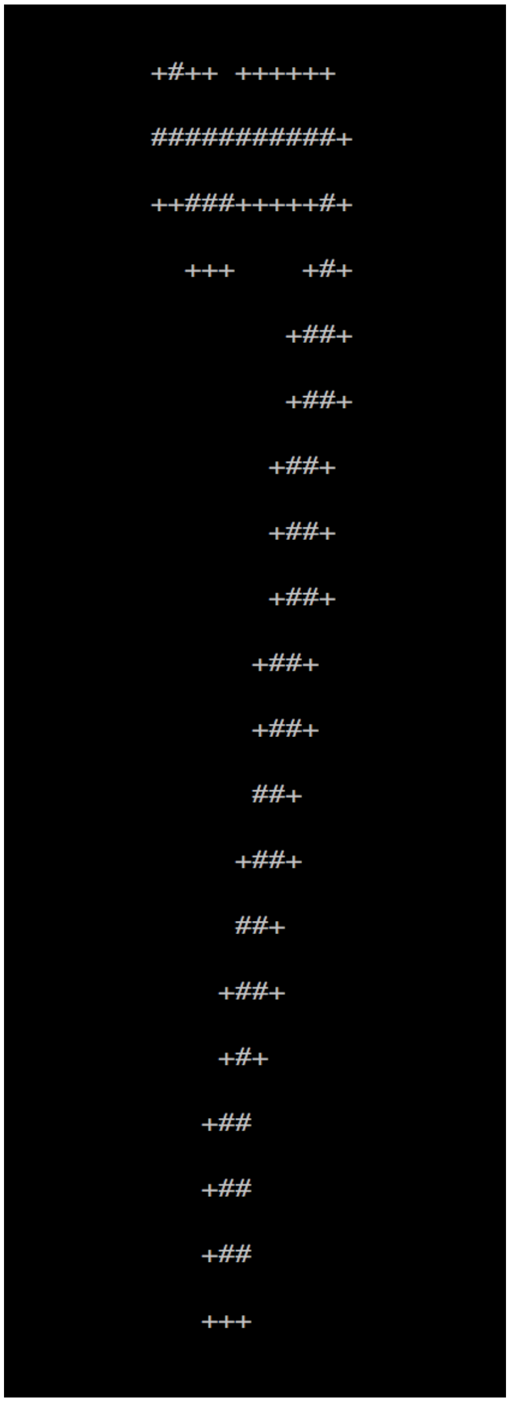


```
Class 0: 84.44444444444444% correct
Class 1: 96.29629629629629% correct
Class 2: 78.64077669902912% correct
Class 3: 80.0% correct
Class 4: 74.76635514018692% correct
Class 5: 68.47826086956522% correct
Class 6: 76.92307692307693% correct
Class 7: 72.64150943396226% correct
Class 8: 60.19417475728155% correct
Class 9: 80.0% correct
[[ 76   0   1   0   1   5   3   0   4   0]
 [  0 104   1   0   0   2   1   0   0   0]
 [  1   3  81   4   2   0   6   1   5   0]
 [  0   1   0  80   0   3   2   7   1   6]
 [  0   0   1   0  80   1   4   1   2  18]
 [  2   1   1  12   3  63   1   1   2   6]
 [  1   4   4   0   4   6  70   0   2   0]
 [  0   6   3   0   3   0   0  77   3  14]
 [  2   1   3  14   3   7   0   1  62  10]
 [  1   1   0   3  10   2   0   2   1  80]]
```

Rows: 0-9 expected label classification. Each row number corresponds to its expected class. Ie row 0 is the actual class of all 0’s.

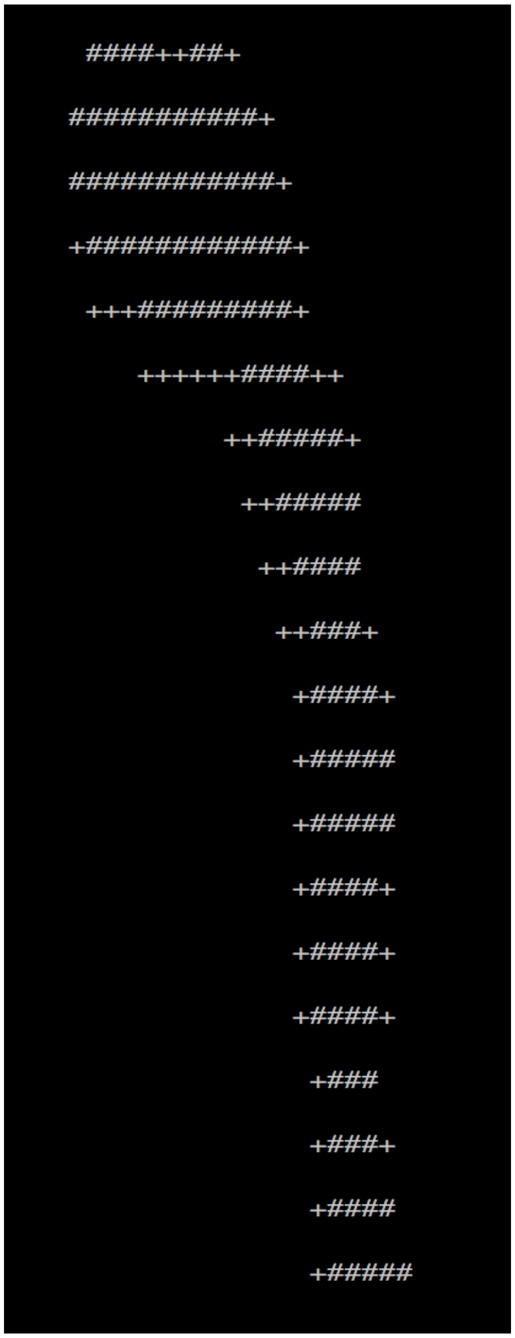
Columns: 0-9 test result label classification. Each column number corresponds to its predicted class. Ie column 0 is the predicted results of all classified 0’s by the classifier.

Highest Lowest posterior probability per class
Class 7:
Best:



```
+#++ ++++++
###########+
++###+++++#+
  +++    +#+
        +##+
        +##+
       +##+
       +##+
       +##+
      +##+
      +##+
      ##+
     +##+
     ##+
    +##+
    +#+
   +##
   +##
   +##
   +++
```

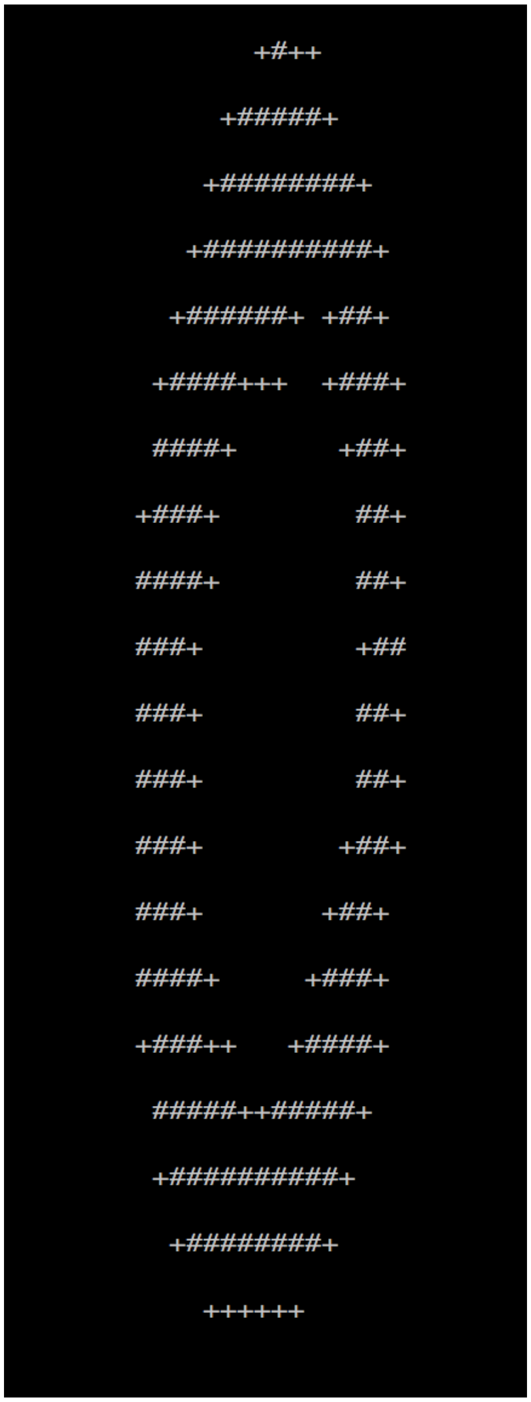
Worst:



```
 ####++##+
###########+
############+
+############+
 +++#########+
    ++++++####++
         ++#####+
          ++#####
           ++####
            ++###+
             +####+
             +#####
             +#####
             +####+
             +####+
             +####+
              +###
              +###+
              +####
              +#####
```

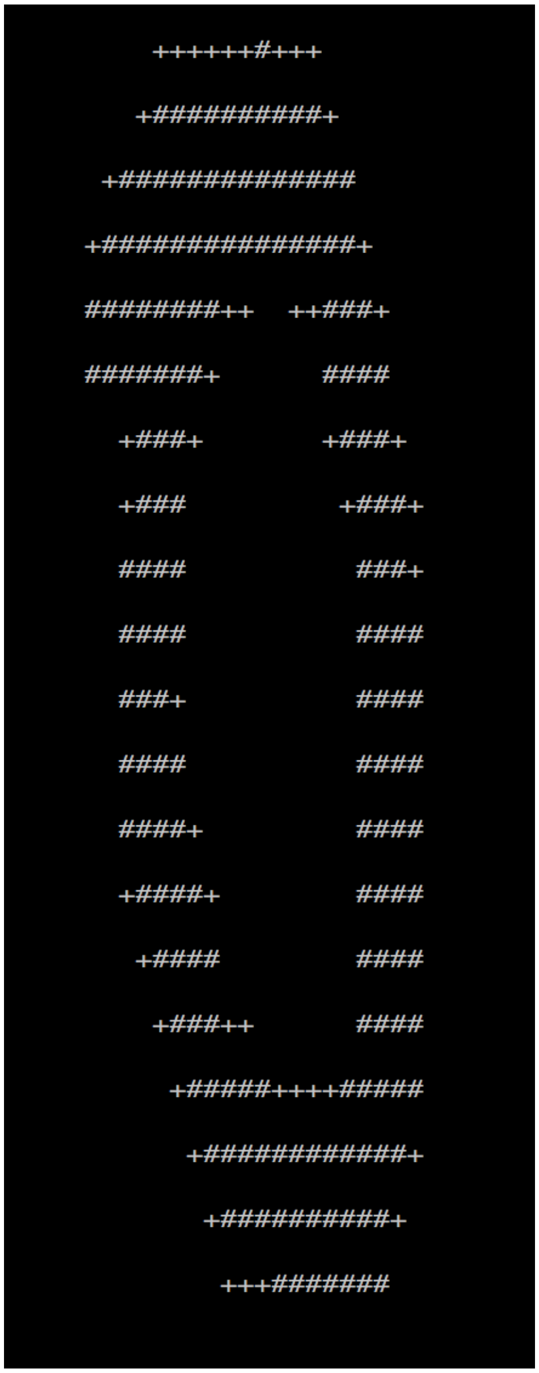
Class 0:

Best:



```
       +#++
     +#####+
    +########+
   +##########+
  +######+ +##+
 +####+++  +###+
 ####+      +##+
+###+        ##+
####+        ##+
###+         +##
###+         ##+
###+         ##+
###+        +##+
###+       +##+
####+     +###+
+###++   +####+
 #####++#####+
 +##########+
  +########+
    ++++++
```

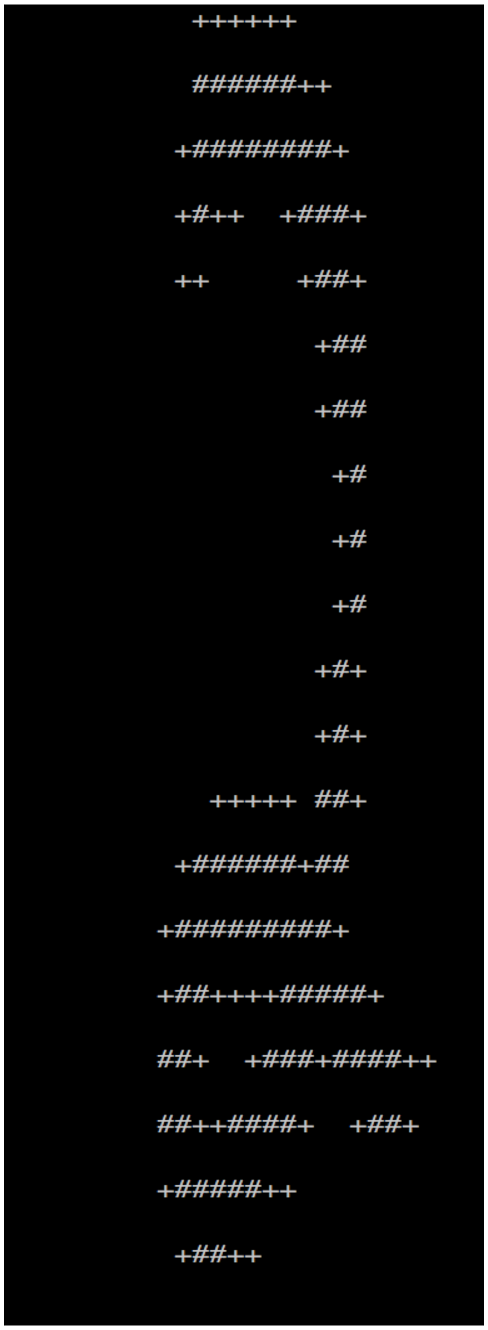
Worst:



```
    ++++++#+++
   +##########+
 +##############
+###############+
########++  ++###+
#######+      ####
  +###+       +###+
  +###         +###+
  ####          ###+
  ####          ####
  ###+          ####
  ####          ####
  ####+         ####
  +####+        ####
   +####        ####
    +###++      ####
     +#####++++#####
      +############+
       +##########+
        +++#######
```

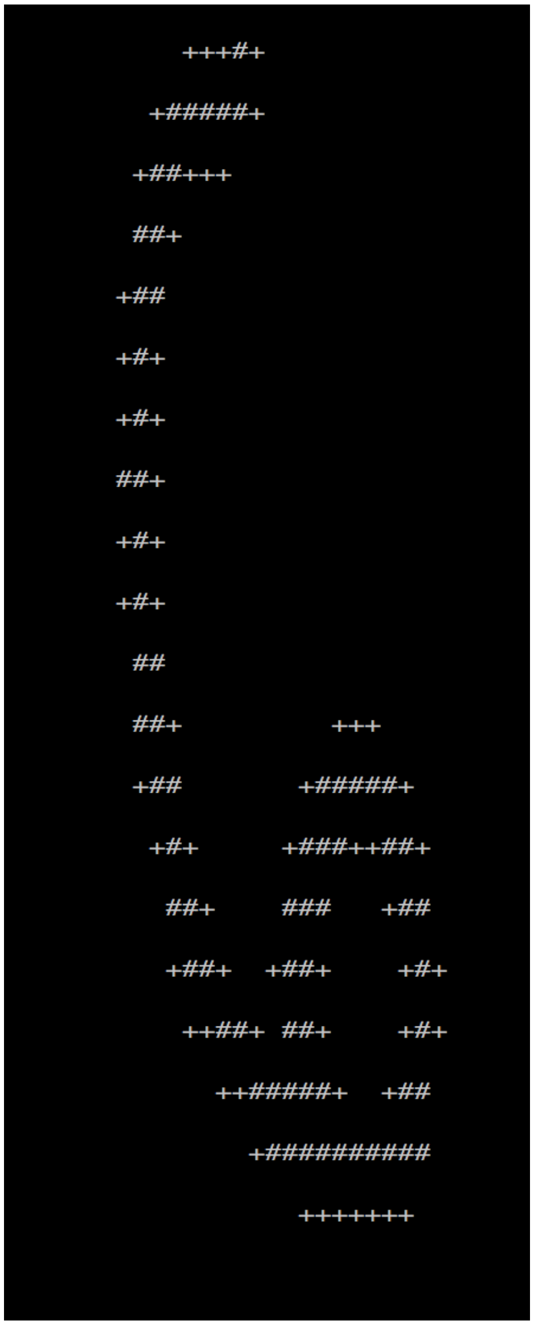
Class 2:

Best:



```
  ++++++
  ######++
 +########+
 +#++  +###+
 ++     +##+
         +##
         +##
          +#
          +#
          +#
         +#+
         +#+
   +++++ ##+
 +######+##
+#########+
+##++++#####+
##+  +###+####++
##++####+  +##+
+#####++
 +##++
```

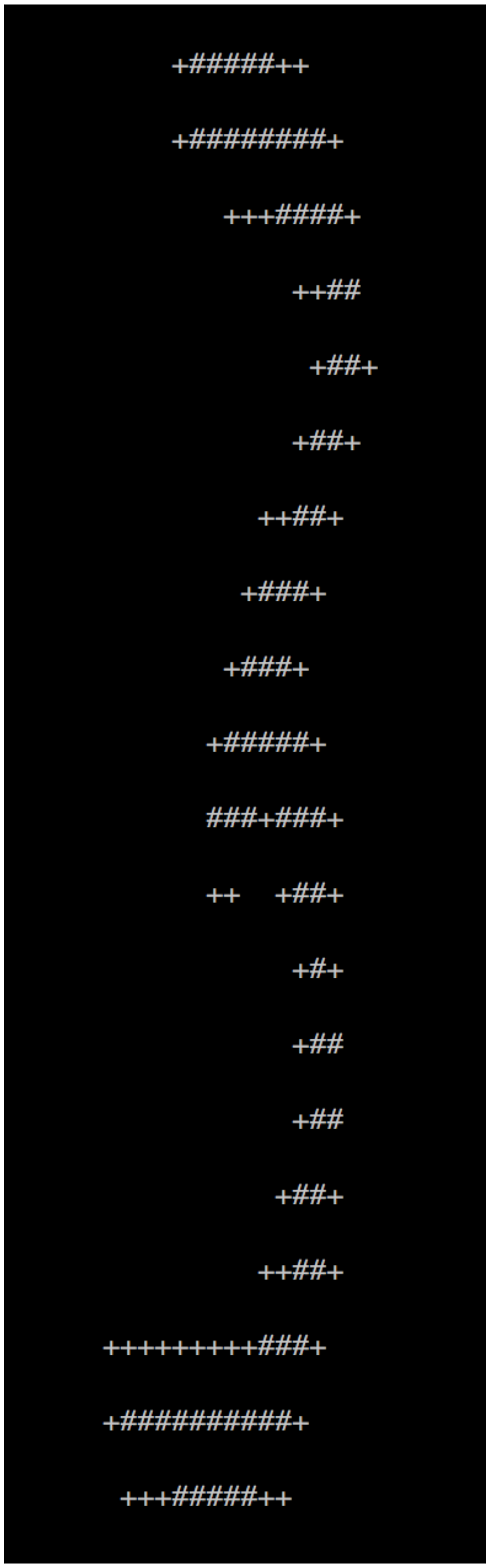
Worst:



```
    +++#+
  +#####+
 +##+++
 ##+
+##
+#+
+#+
##+
+#+
+#+
 ##
 ##+         +++
 +##       +#####+
  +#+     +###++##+
   ##+    ###   +##
   +##+  +##+    +#+
    ++##+ ##+    +#+
      ++#####+  +##
        +##########
           +++++++
```

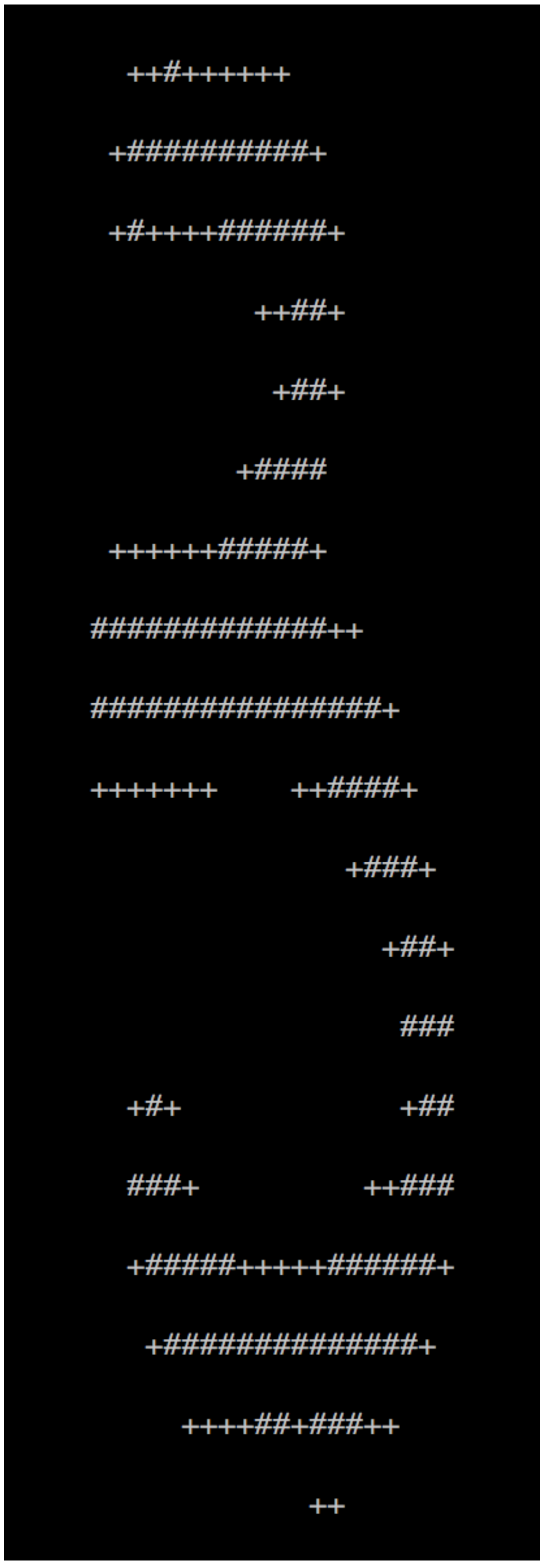
Class 3:

Best:



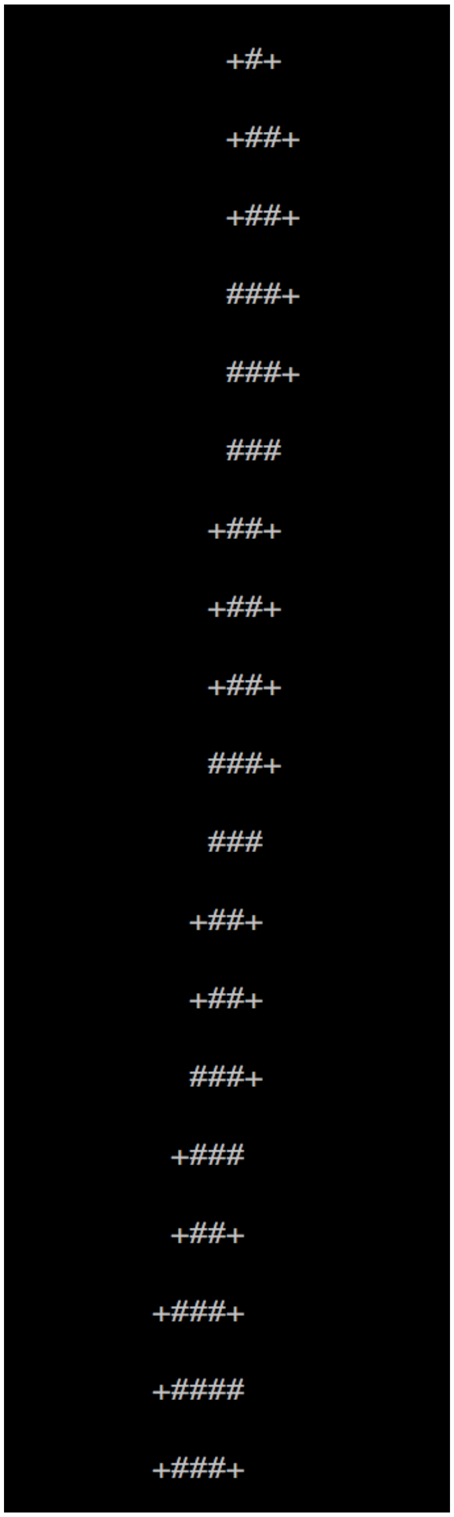
```
    +#####++
    +########+
       +++####+
           ++##
            +##+
           +##+
         ++##+
        +###+
       +###+
      +#####+
      ###+###+
      ++  +##+
           +#+
           +##
           +##
          +##+
         ++##+
+++++++++###+
+##########+
 +++#####++
```

Worst:



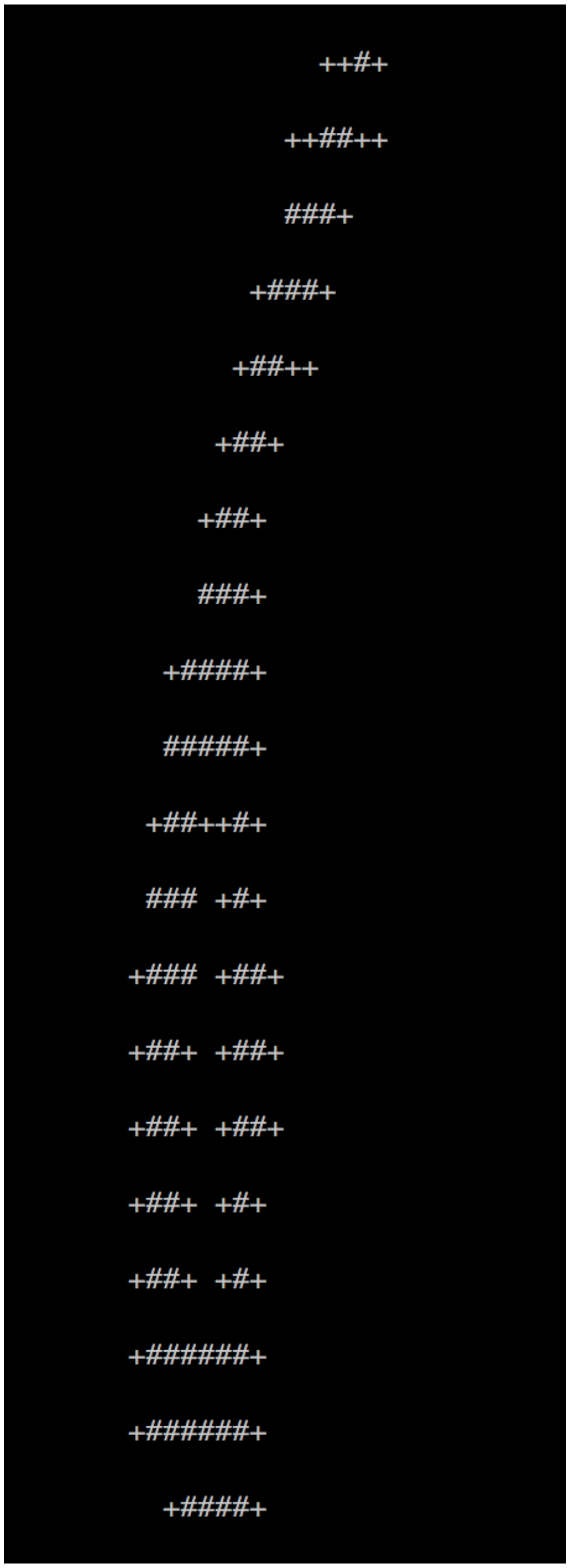
```
  ++#+++++++
 +##########+
 +#++++######+
         ++##+
          +##+
        +####
 ++++++#####+
#############++
################+
+++++++    ++####+
              +###+
                +##+
                 ###
  +#+            +##
  ###+         ++###
  +#####+++++######+
   +##############+
     ++++##+###++
            ++
```

Class 1:
Best:



```
            +#+
            +##+
            +##+
            ###+
            ###+
            ###
           +##+
           +##+
           +##+
           ###+
           ###
          +##+
          +##+
          ###+
         +###
         +##+
        +###+
        +####
        +###+
```

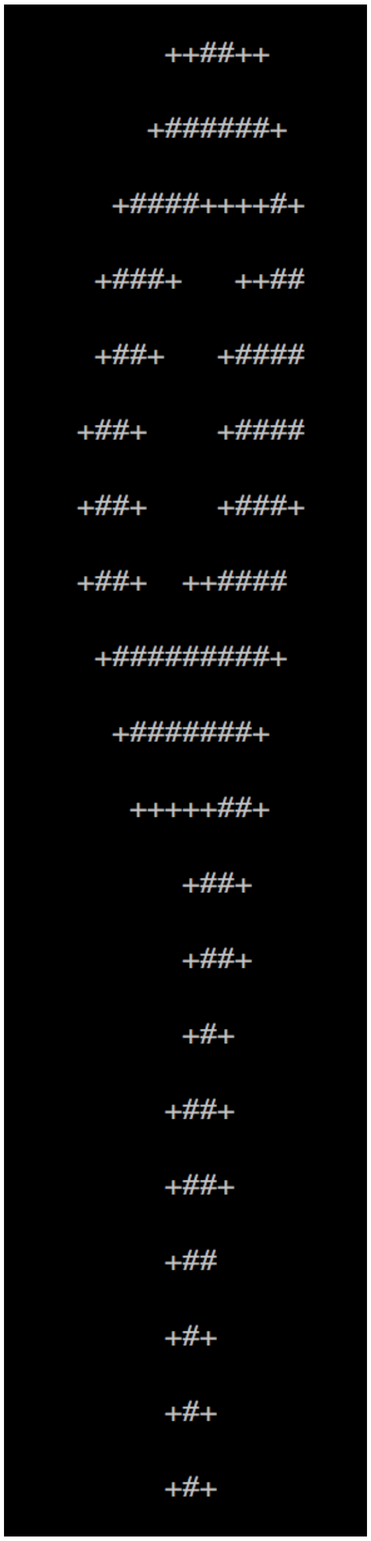
Worst:



```
                  ++#+
                ++##++
                ###+
              +###+
             +##++
            +##+
           +##+
           ###+
         +####+
         #####+
        +##++#+
        ### +#+
       +### +##+
       +##+ +##+
       +##+ +##+
       +##+ +#+
       +##+ +#+
       +######+
       +######+
         +####+
```

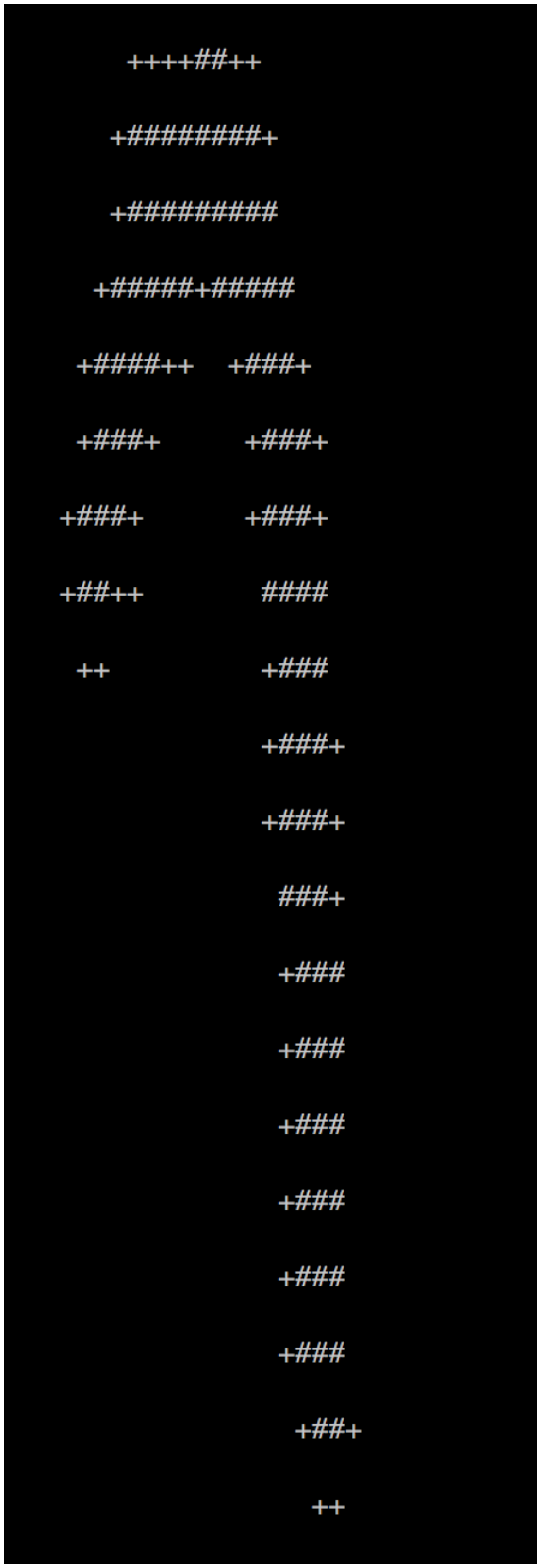
Class 9:

Best:



```
     ++##++
    +######+
  +####++++#+
 +###+   ++##
 +##+   +####
+##+    +####
+##+    +###+
+##+  ++####
 +#########+
  +#######+
   +++++##+
      +##+
      +##+
      +#+
     +##+
     +##+
     +##
     +#+
     +#+
     +#+
```

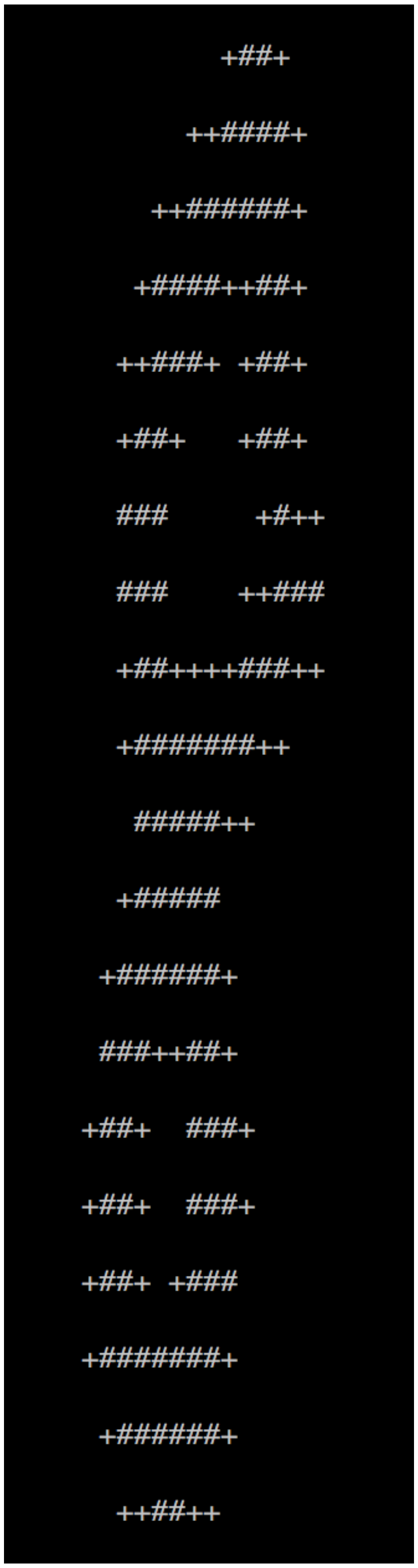
Worst:



```
    ++++##++
   +########+
   +#########
  +#####+#####
 +####++  +###+
 +###+     +###+
+###+      +###+
+##++       ####
 ++         +###
            +###+
            +###+
             ###+
             +###
             +###
             +###
             +###
             +###
             +###
              +##+
               ++
```

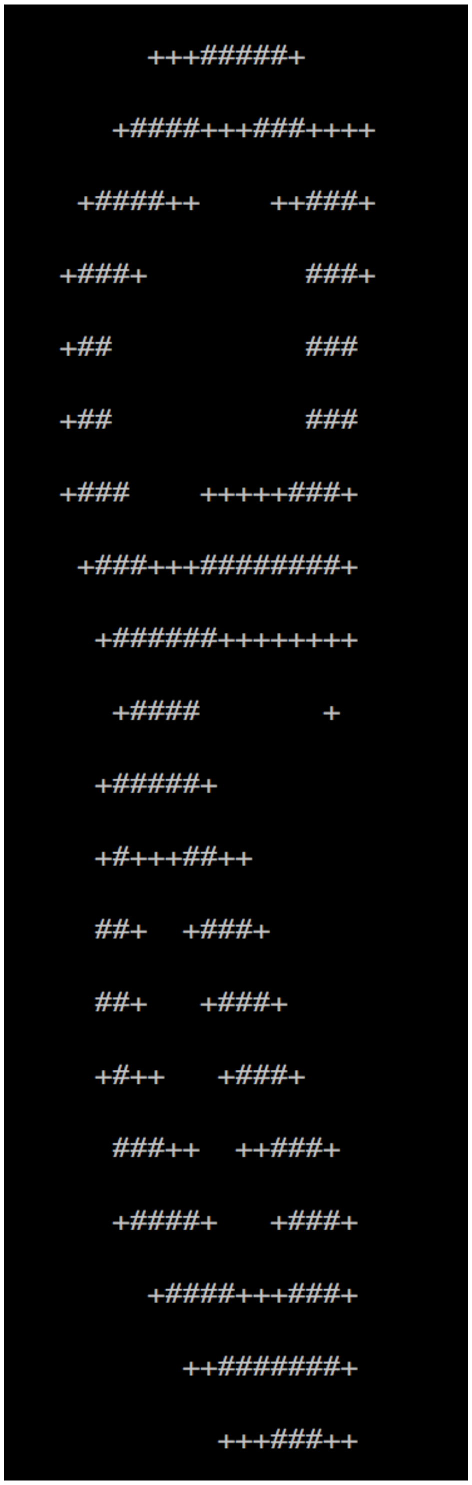
Class 8:

Best:



```
        +##+

      ++####+

    ++######+

   +####++##+

  ++###+ +##+

  +##+   +##+

  ###     +#++

  ###    ++###

  +##++++###++

  +#######++

   #####++

  +#####

 +######+

 ###++##+

+##+  ###+

+##+  ###+

+##+ +###

+#######+

 +######+

  ++##++
```

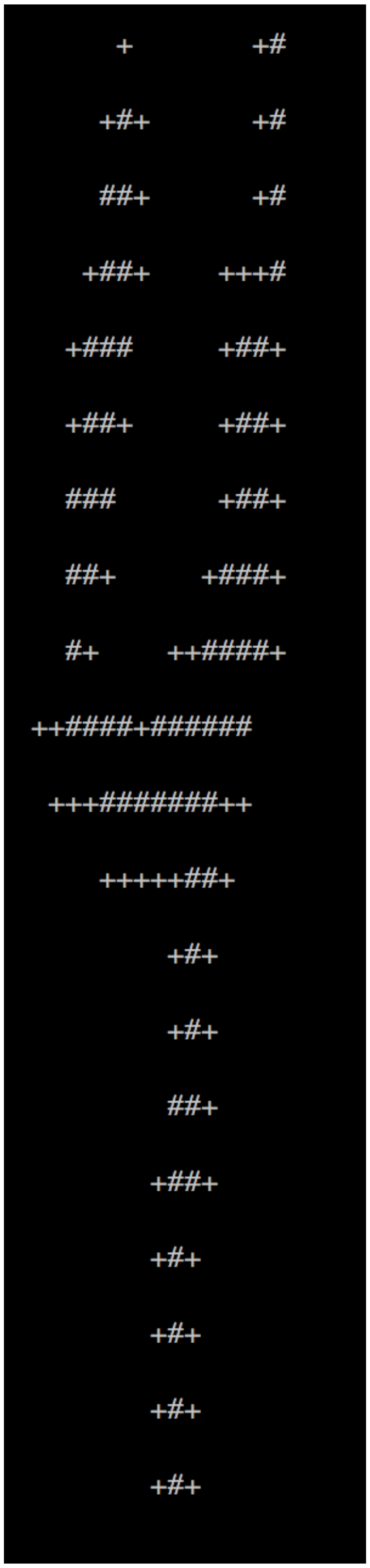
Worst:



```
     +++#####+

   +####+++###++++

 +####++    ++###+

+###+         ###+

+##           ###

+##           ###

+###    +++++###+

 +###+++#########+

  +######++++++++

   +####       +

  +#####+

  +#+++##++

  ##+  +###+

  ##+   +###+

  +#++   +###+

   ###++  ++###+

   +####+   +###+

     +####+++###+

       ++#######+

         +++###++
```

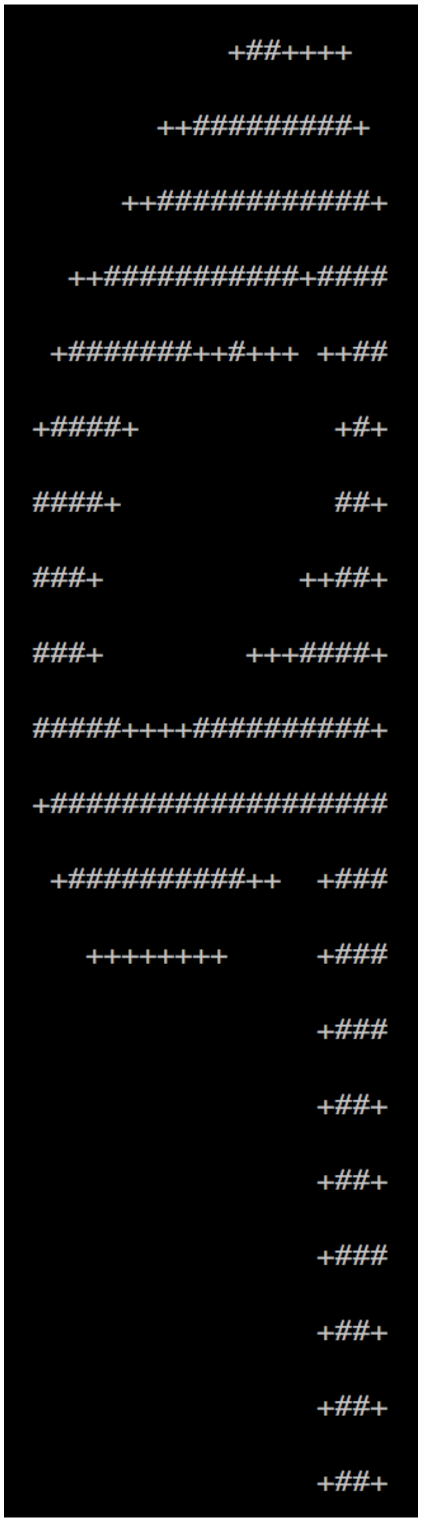
Class 4:

Best:



```
     +       +#
    +#+      +#
    ##+      +#
   +##+    +++#
  +###     +##+
  +##+     +##+
  ###      +##+
  ##+     +###+
  #+    ++####+
++####+######
 +++#######++
    +++++##+
        +#+
        +#+
        ##+
       +##+
       +#+
       +#+
       +#+
       +#+
```

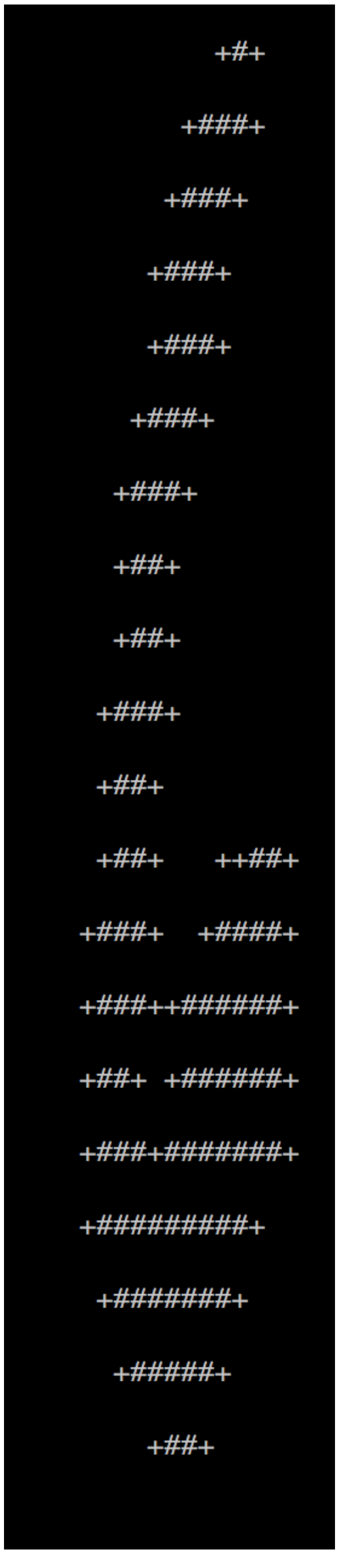
Worst:



```
           +##++++
       ++#########+
     ++############+
  ++###########+####
 +#######++#+++ ++##
+####+           +#+
####+            ##+
###+           ++##+
###+        +++####+
#####++++##########+
+###################
 +##########++  +###
   ++++++++     +###
                +###
                +##+
                +##+
                +###
                +##+
                +##+
                +##+
```

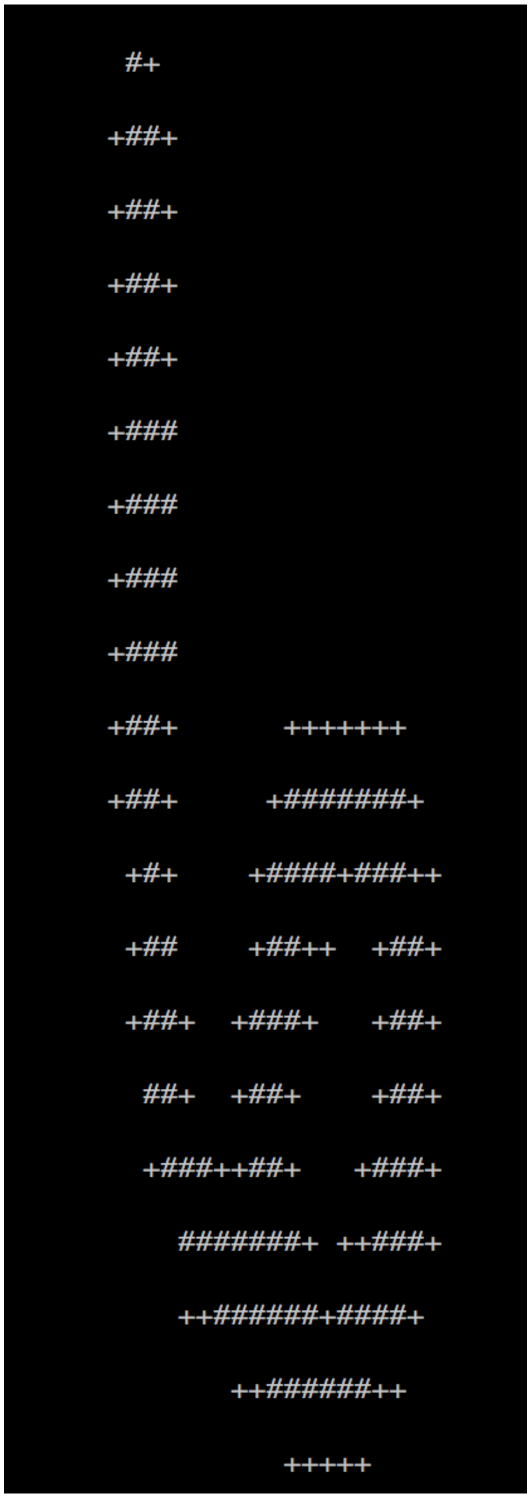
Class 6:

Best:



```
        +#+
      +###+
     +###+
    +###+
    +###+
   +###+
  +###+
  +##+
  +##+
 +###+
 +##+
 +##+   ++##+
+###+  +####+
+###++######+
+##+ +######+
+###+#######+
+#########+
 +#######+
  +#####+
    +##+
```

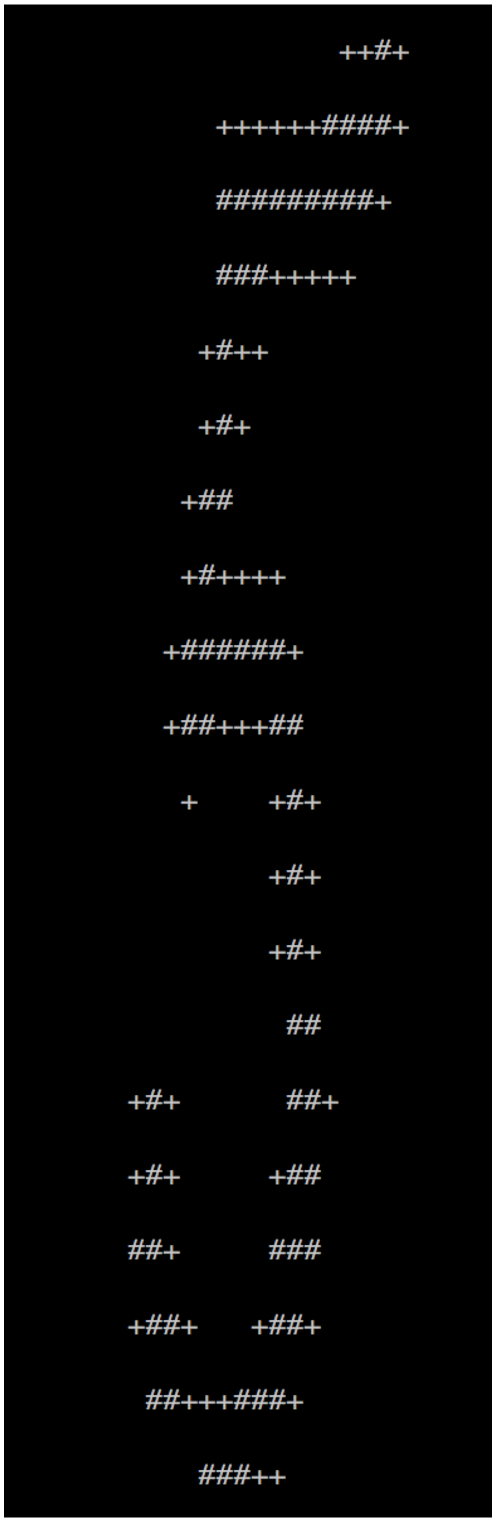
Worst:



```
 #+
+##+
+##+
+##+
+##+
+###
+###
+###
+###
+##+      +++++++
+##+     +#######+
 +#+    +####+###++
 +##    +##++  +##+
 +##+  +###+   +##+
  ##+  +##+    +##+
  +###++##+   +###+
    #######+ ++###+
    ++######+####+
       ++######++
          +++++
```

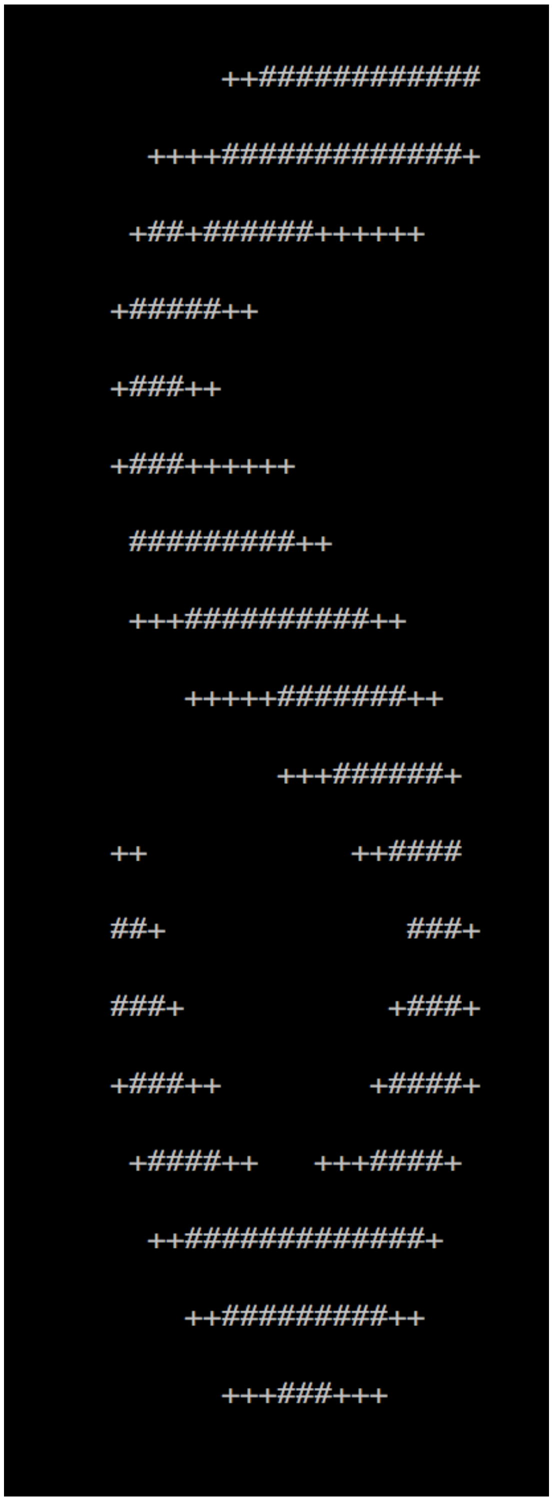
Class 5:

Best:



```
            ++#+
     ++++++####+
     #########+
     ###+++++
    +#++
    +#+
   +##
   +#++++
  +######+
  +##+++##
   +    +#+
        +#+
        +#+
         ##
+#+      ##+
+#+     +##
##+     ###
+##+   +##+
 ##+++###+
    ###++
```

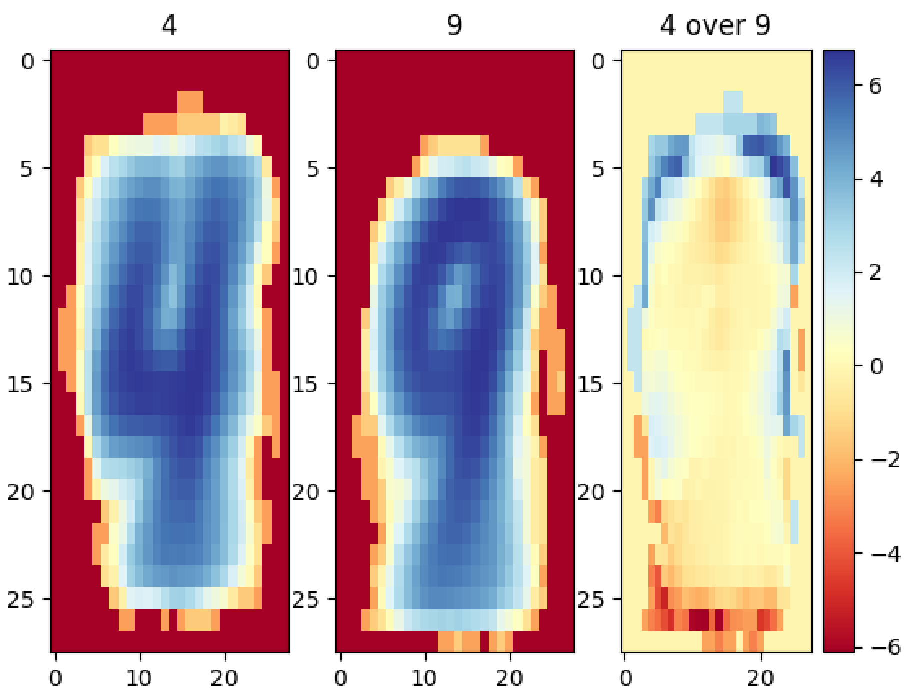
Worst:



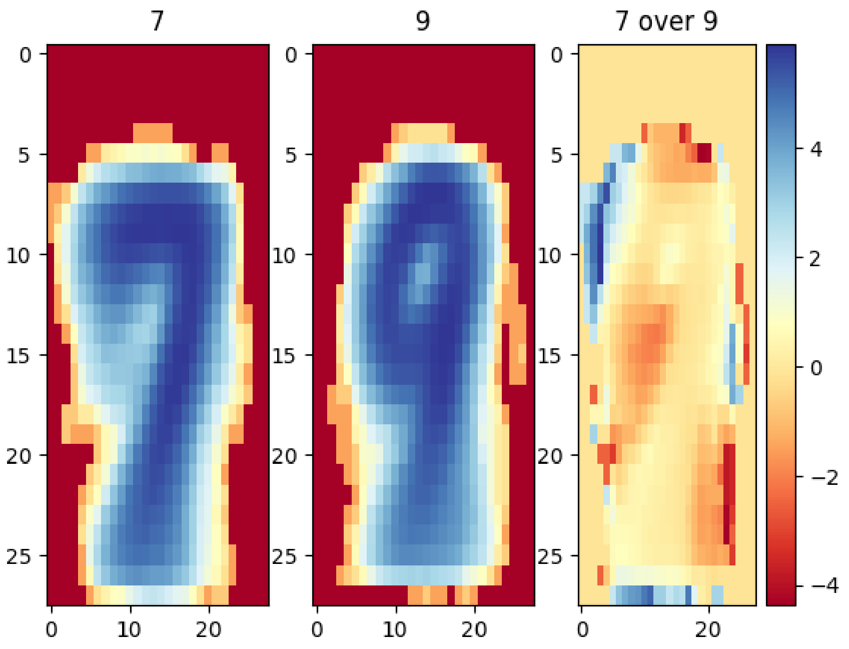
```
      ++############
  ++++#############+
 +##+######++++++
+#####++
+###++
+###++++++
 #########++
 +++##########++
    +++++#######++
         +++######+
++           ++####
##+             ###+
###+           +###+
+###++        +####+
 +####++   +++####+
  ++#############+
    ++#########++
      +++###+++
```

Feature Likelihoods and Odds Ratios:

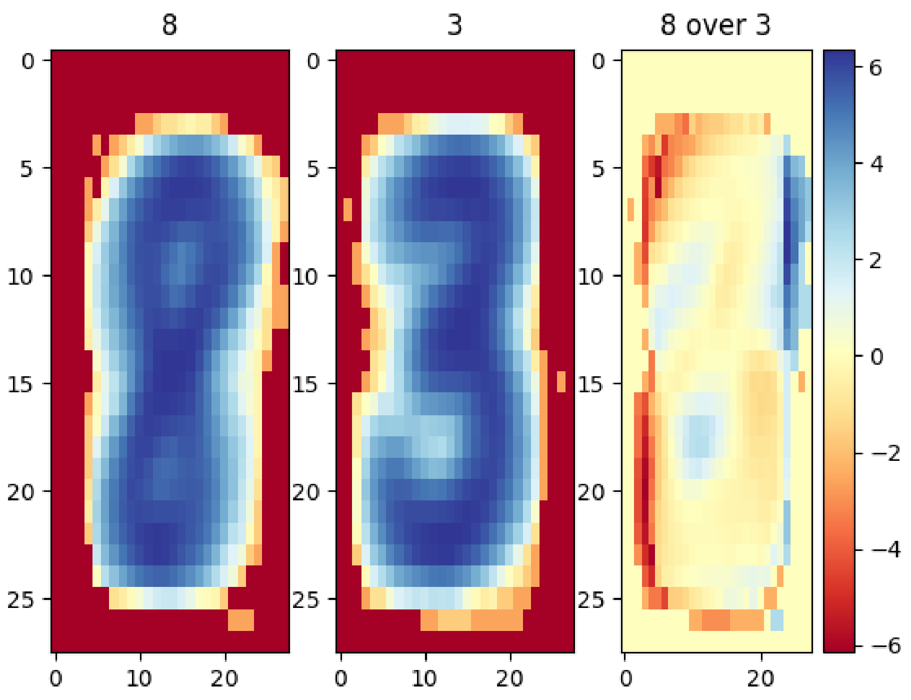
4 as 9: 18 times



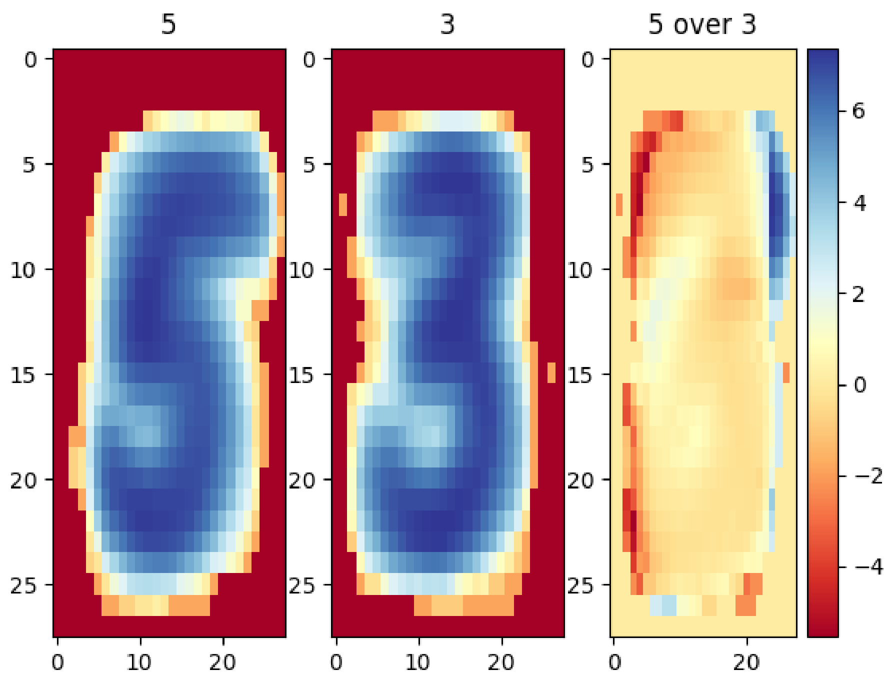
7 as 9: 14 times



8 as 3: 14 times



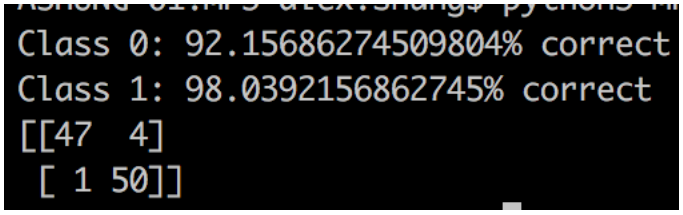
5 as 3: 12 times



Part 2: Implementation

For this part of the project, the above described implementation was mostly used. Since the input data came in a 28 x 10 format this time, the size of the matrices were adjusted by skipping the last 3 lines after each file as it was specified that the data should be 25 x 10. The data was read from both files, and then merged into a single list of numbers for both testing and training purposes. The matrix values were 1 for high energy, marked by the blank value, and 0 for everything else. The Laplace transform was discovered to not have much of an effect on this model, and was experimented using the same findBestSmoothingConstant described above. Once the data was set up in 25 x 10 matrices with values as described above, it was fed into the same implementation, and the model was run on those matrices instead except instead of numbers being represented, it was the sound of yes or no. In this case it was focusing on areas of high energy instead of pixels being present.

For testing, the implementation method was also exactly the same. Labels were generated by taking the number of input "voices" and creating that many "no" / "yes" labels respective to which file the data came from. The test data was prepared identically to how Part 1 was done, except specifying that lines should be skipped and once again that the things being represented were voices instead of numbers. Once the data was prepared, it was run and tested exactly as described in Part 1. The confusion matrix and correct % are below.



```
Class 0: 92.15686274509804% correct
Class 1: 98.0392156862745% correct
[[47  4]
 [ 1 50]]
```

Class 0 = "no" test data classified correctly.
Class 1 = "yes" test data classified correctly.
Columns are what the classifier predicted
Rows are what the test data actually were
First column is "no", second column is "yes"
First row is "no", second column is "yes"