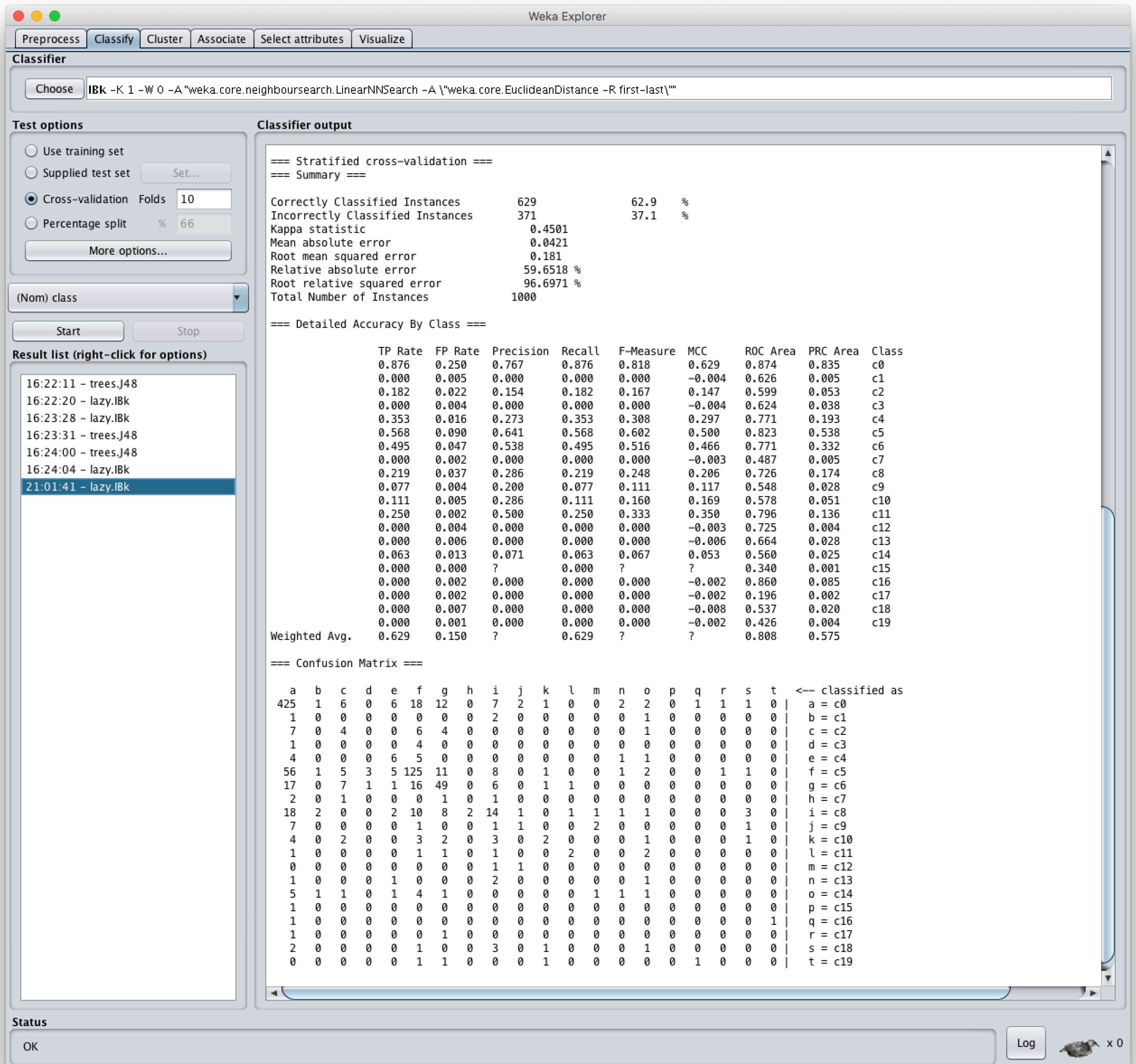
Ikhlas Attarwala
MSAI 349 – Machine Learning
December 2nd, 2018

PS_4

# NOTE: ANSWERS IN RED

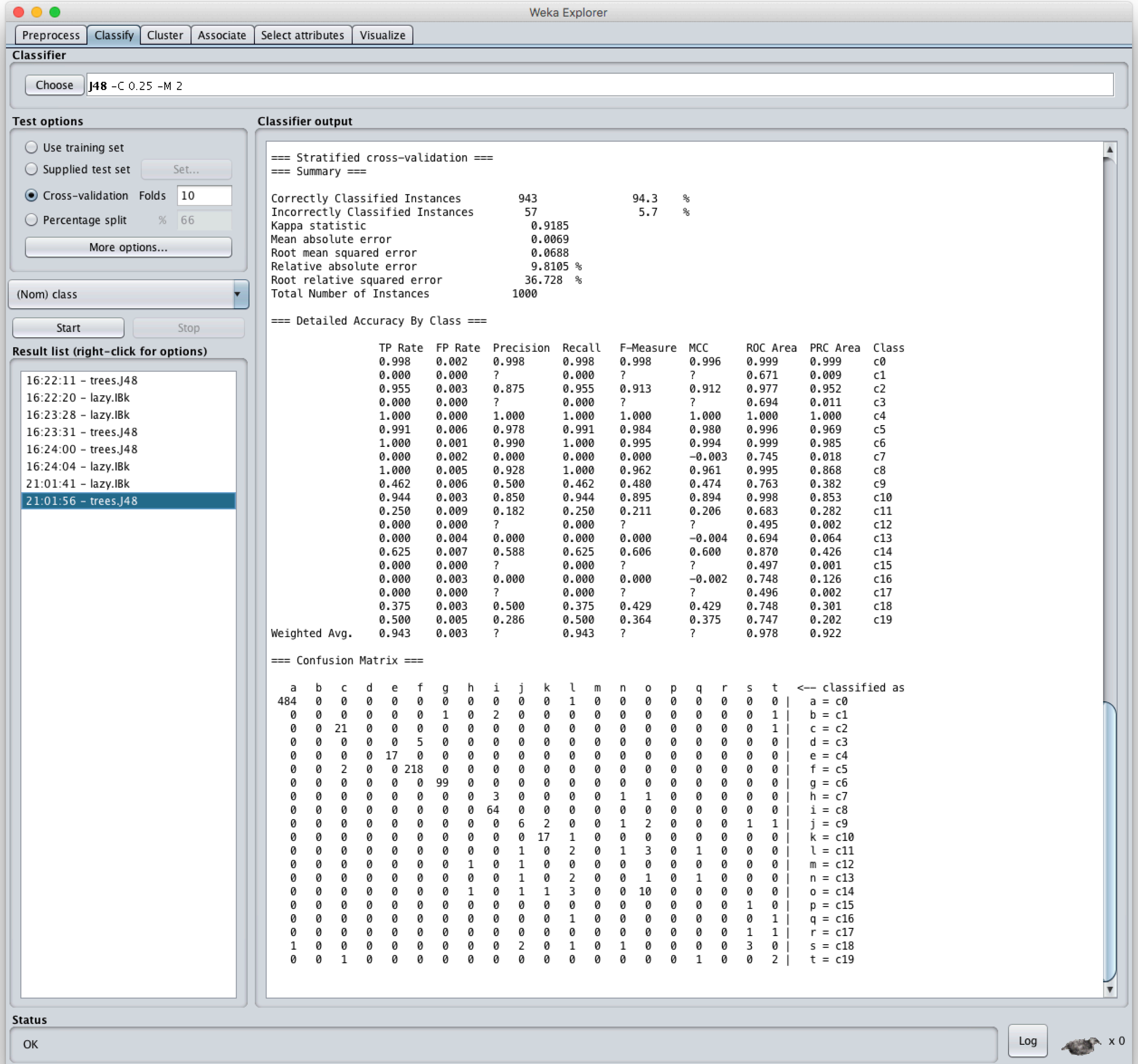
## Part 1

1) Data set created through Weka. Here is the output, although the csv is also included in the submission:




```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         629               62.9    %
Incorrectly Classified Instances       371               37.1    %
Kappa statistic                          0.4501
Mean absolute error                      0.0421
Root mean squared error                  0.181
Relative absolute error                 59.6518 %
Root relative squared error             96.6971 %
Total Number of Instances             1000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.876    0.250    0.767      0.876    0.818      0.629    0.874     0.835     c0
                 0.000    0.005    0.000      0.000    0.000      -0.004   0.626     0.005     c1
                 0.182    0.022    0.154      0.182    0.167      0.147    0.599     0.053     c2
                 0.000    0.004    0.000      0.000    0.000      -0.004   0.624     0.038     c3
                 0.353    0.016    0.273      0.353    0.308      0.297    0.771     0.193     c4
                 0.568    0.090    0.641      0.568    0.602      0.500    0.823     0.538     c5
                 0.495    0.047    0.538      0.495    0.516      0.466    0.771     0.332     c6
                 0.000    0.002    0.000      0.000    0.000      -0.003   0.487     0.005     c7
                 0.219    0.037    0.286      0.219    0.248      0.206    0.726     0.174     c8
                 0.077    0.004    0.200      0.077    0.111      0.117    0.548     0.028     c9
                 0.111    0.005    0.286      0.111    0.160      0.169    0.578     0.051     c10
                 0.250    0.002    0.500      0.250    0.333      0.350    0.796     0.136     c11
                 0.000    0.004    0.000      0.000    0.000      -0.003   0.725     0.004     c12
                 0.000    0.006    0.000      0.000    0.000      -0.006   0.664     0.028     c13
                 0.063    0.013    0.071      0.063    0.067      0.053    0.560     0.025     c14
                 0.000    0.000    ?          0.000    ?          ?        0.340     0.001     c15
                 0.000    0.002    0.000      0.000    0.000      -0.002   0.860     0.085     c16
                 0.000    0.002    0.000      0.000    0.000      -0.002   0.196     0.002     c17
                 0.000    0.007    0.000      0.000    0.000      -0.008   0.537     0.020     c18
                 0.000    0.001    0.000      0.000    0.000      -0.002   0.426     0.004     c19
Weighted Avg.    0.629    0.150    ?          0.629    ?          ?        0.808     0.575

=== Confusion Matrix ===

   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o   p   q   r   s   t   <-- classified as
 425   1   6   0   6  18  12   0   7   2   1   0   0   2   2   0   1   1   1   0 |   a = c0
   1   0   0   0   0   0   0   0   2   0   0   0   0   0   1   0   0   0   0   0 |   b = c1
   7   0   4   0   0   6   4   0   0   0   0   0   0   0   1   0   0   0   0   0 |   c = c2
   1   0   0   0   0   4   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |   d = c3
   4   0   0   0   6   5   0   0   0   0   0   0   0   1   1   0   0   0   0   0 |   e = c4
  56   1   5   3   5 125  11   0   8   0   1   0   0   1   2   0   0   1   1   0 |   f = c5
  17   0   7   1   1  16  49   0   6   0   1   1   0   0   0   0   0   0   0   0 |   g = c6
   2   0   1   0   0   0   1   0   1   0   0   0   0   0   0   0   0   0   0   0 |   h = c7
  18   2   0   0   2  10   8   2  14   1   0   1   1   1   1   0   0   0   3   0 |   i = c8
   7   0   0   0   0   1   0   0   1   1   0   0   2   0   0   0   0   0   1   0 |   j = c9
   4   0   2   0   0   3   2   0   3   0   2   0   0   0   1   0   0   0   1   0 |   k = c10
   1   0   0   0   0   1   1   0   1   0   0   2   0   0   2   0   0   0   0   0 |   l = c11
   0   0   0   0   0   0   0   0   1   1   0   0   0   0   0   0   0   0   0   0 |   m = c12
   1   0   0   0   1   0   0   0   2   0   0   0   0   0   1   0   0   0   0   0 |   n = c13
   5   1   1   0   1   4   1   0   0   0   0   0   1   1   1   0   0   0   0   0 |   o = c14
   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |   p = c15
   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1 |   q = c16
   1   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0 |   r = c17
   2   0   0   0   0   1   0   0   3   0   1   0   0   0   1   0   0   0   0   0 |   s = c18
   0   0   0   0   0   1   1   0   0   0   1   0   0   0   0   0   1   0   0   0 |   t = c19
```

Nearest Neighbor w/ accuracy 62.9%

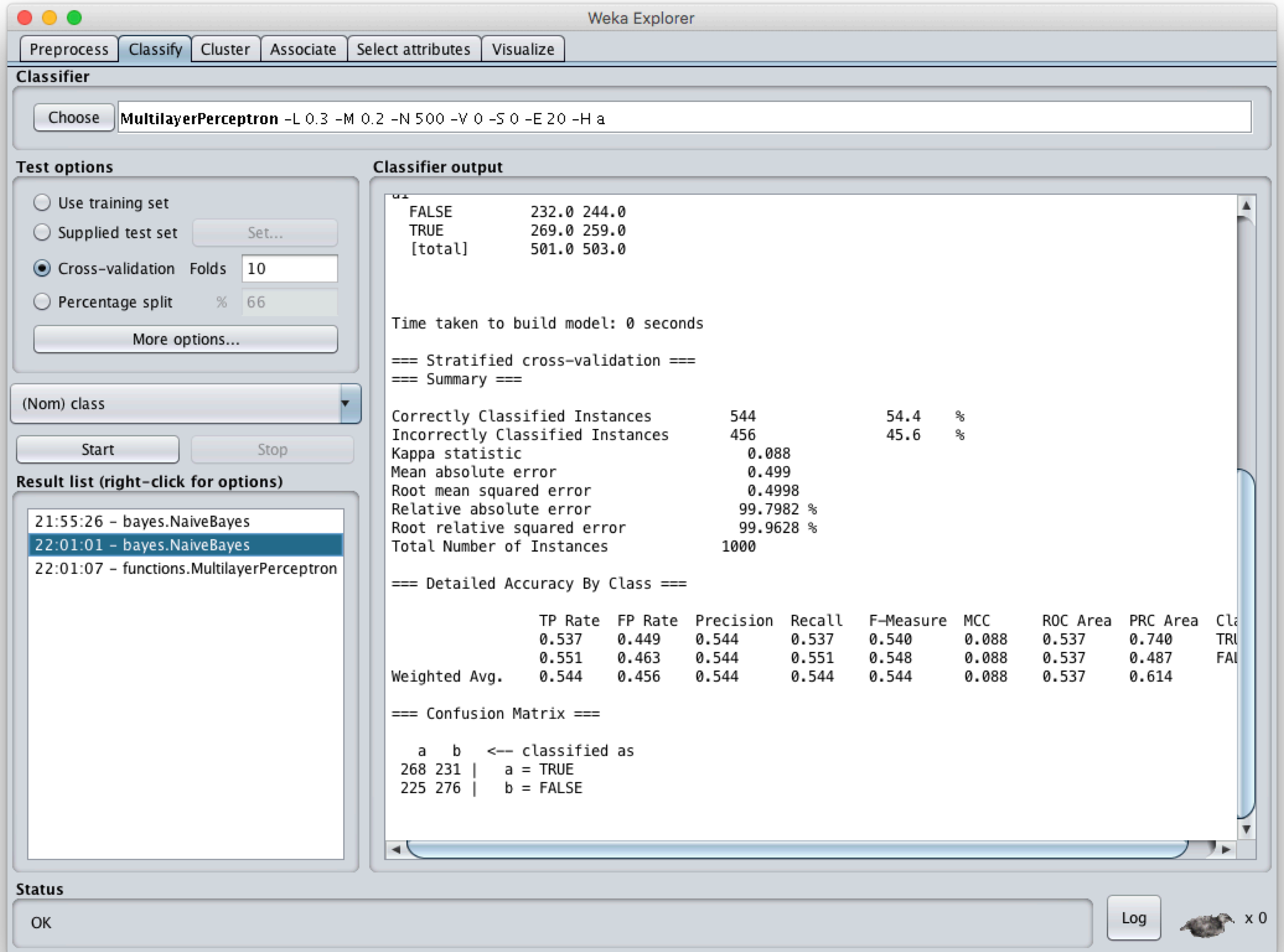



```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         943               94.3    %
Incorrectly Classified Instances        57                5.7    %
Kappa statistic                          0.9185
Mean absolute error                      0.0069
Root mean squared error                  0.0688
Relative absolute error                  9.8105 %
Root relative squared error             36.728  %
Total Number of Instances             1000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.998    0.002    0.998      0.998    0.998      0.996    0.999     0.999     c0
                 0.000    0.000    ?          0.000    ?          ?        0.671     0.009     c1
                 0.955    0.003    0.875      0.955    0.913      0.912    0.977     0.952     c2
                 0.000    0.000    ?          0.000    ?          ?        0.694     0.011     c3
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     c4
                 0.991    0.006    0.978      0.991    0.984      0.980    0.996     0.969     c5
                 1.000    0.001    0.990      1.000    0.995      0.994    0.999     0.985     c6
                 0.000    0.002    0.000      0.000    0.000      -0.003   0.745     0.018     c7
                 1.000    0.005    0.928      1.000    0.962      0.961    0.995     0.868     c8
                 0.462    0.006    0.500      0.462    0.480      0.474    0.763     0.382     c9
                 0.944    0.003    0.850      0.944    0.895      0.894    0.998     0.853     c10
                 0.250    0.009    0.182      0.250    0.211      0.206    0.683     0.282     c11
                 0.000    0.000    ?          0.000    ?          ?        0.495     0.002     c12
                 0.000    0.004    0.000      0.000    0.000      -0.004   0.694     0.064     c13
                 0.625    0.007    0.588      0.625    0.606      0.600    0.870     0.426     c14
                 0.000    0.000    ?          0.000    ?          ?        0.497     0.001     c15
                 0.000    0.003    0.000      0.000    0.000      -0.002   0.748     0.126     c16
                 0.000    0.000    ?          0.000    ?          ?        0.496     0.002     c17
                 0.375    0.003    0.500      0.375    0.429      0.429    0.748     0.301     c18
                 0.500    0.005    0.286      0.500    0.364      0.375    0.747     0.202     c19
Weighted Avg.    0.943    0.003    ?          0.943    ?          ?        0.978     0.922

=== Confusion Matrix ===

   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o   p   q   r   s   t   <-- classified as
 484   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0 |   a = c0
   0   0   0   0   0   0   1   0   2   0   0   0   0   0   0   0   0   0   0   1 |   b = c1
   0   0  21   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1 |   c = c2
   0   0   0   0   0   5   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |   d = c3
   0   0   0   0  17   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |   e = c4
   0   0   2   0   0 218   0   0   0   0   0   0   0   0   0   0   0   0   0   0 |   f = c5
   0   0   0   0   0   0  99   0   0   0   0   0   0   0   0   0   0   0   0   0 |   g = c6
   0   0   0   0   0   0   0   0   3   0   0   0   0   1   1   0   0   0   0   0 |   h = c7
   0   0   0   0   0   0   0   0  64   0   0   0   0   0   0   0   0   0   0   0 |   i = c8
   0   0   0   0   0   0   0   0   0   6   2   0   0   1   2   0   0   0   1   1 |   j = c9
   0   0   0   0   0   0   0   0   0   0  17   1   0   0   0   0   0   0   0   0 |   k = c10
   0   0   0   0   0   0   0   0   0   1   0   2   0   1   3   0   1   0   0   0 |   l = c11
   0   0   0   0   0   0   0   1   0   1   0   0   0   0   0   0   0   0   0   0 |   m = c12
   0   0   0   0   0   0   0   0   0   1   0   2   0   0   1   0   1   0   0   0 |   n = c13
   0   0   0   0   0   0   0   1   0   1   1   3   0   0  10   0   0   0   0   0 |   o = c14
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0 |   p = c15
   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   1 |   q = c16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1 |   r = c17
   1   0   0   0   0   0   0   0   0   2   0   1   0   1   0   0   0   0   3   0 |   s = c18
   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   2 |   t = c19
```

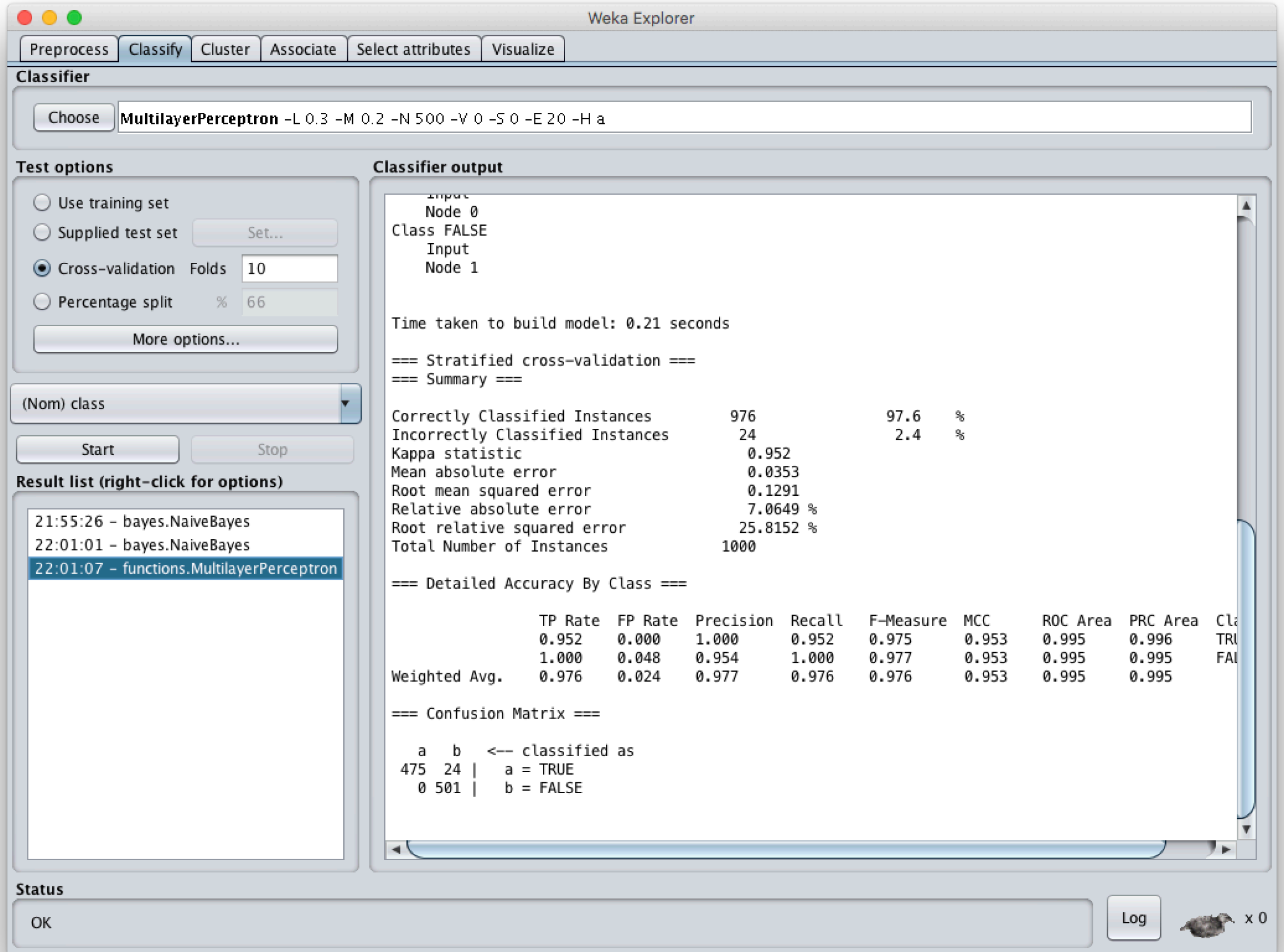
Decision Tree w/ accuracy 94.3%

a. Absolute difference = |94.3% – 62.9% | = 31.4%
b. I generated the dataset through Weka itself. I spent a long time generating the data through Excel sheets but realized that wasn't necessary. Through Weka I generated randomly produced data by producing a decision list. In particular, I set the number of attributes to 30, number of classes to 20, kept 1000 examples as requested, and the number of irrelevant attributes to 10. The simple answer for why the classifiers perform so differently is that decision trees are eager learners that first build a classification model on the training dataset and predict a class for a given input vector. Nearest neighbor, however, does not build a classification model and instead learns directly from the training observations. It is coined a lazy learner and must use a distance metric. It lowers in accuracy as we add more dimensions (features).

2) Data set created through Weka. Here is the output, although the csv is also included in the submission:




Naïve Bayes w/ accuracy 54.4%




MLP w/ accuracy 97.6%

a. Absolute difference = |97.6% − 54.4% | = 43.2%
b. I created the dataset without the help of Weka this time. While my two attributes were randomly true or false, my output was the XOR function. This enabled me to achieve such a low percentage with Naïve Bayes. Naïve Bayes classifiers work by making strong assumptions of conditional between attributes. As mentioned above, I specifically created the data set so that this wasn't the case at all. The MLP model had a better result because it iteratively changes the weights between each successive node to find a stronger correlation and thus classified with better accuracy.

3) Through Jupiter Notebook:

```
In [46]: import numpy as np

         from sklearn.naive_bayes import BernoulliNB

         from sklearn.linear_model import LogisticRegression

         a = BernoulliNB()

         b = LogisticRegression(C=5)

In [47]: attributes = np.array([[1,1,1,1],[1,0,0,1],[1,1,0,1]])

         output = np.array([0,0,1])

In [48]: a.fit(attributes,output)

         b.fit(attributes,output)

Out[48]: LogisticRegression(C=5, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)

In [49]: a.score(attributes,output)

Out[49]: 0.6666666666666666

In [50]: b.score(attributes,output)

Out[50]: 1.0
```

Bernoulli's Model (Naïve Bayes) vs Logistic Regression

a. For this question, while I worked alone for the most part, I asked Jack Richard for a little guidance (Downey allowed). He suggested I attempt this problem via the Jupiter Notebook. In it I used the Bernoulli Model for Naïve Bayes, and Logistic Regression to achieve a 1.0 and 2/3 training accuracy respectively, as depicted above. In order to achieve a 2/3 training accuracy, it was simpler to get 2 out of 3 example sets to train correctly, while still keeping true to 4 attributes in the matrix labeled `attributes`. Because Naïve Bayes assumes conditional independence, it does not look for any relations between attributes. Logistic Regression can learn a function by a feature. As one of the example sets is classified differently, Naïve Bayes classifies it wrong (2/3). Logistic Regression measures the relationship between the output and one or more independent variables, and probabilistically predicted correctly, thus scoring a 1.0 accuracy.

4) Computing # of parameters
   a. 9 * 3 * 3 * 3 – 1 = 242 independent parameters
   b. 9 * 3 * 3 * (3 – 1) = 162 independent parameters
   c. (9-1)*3 + (3-1)*3 + (3-1)*3 + (3-1) = 38 independent parameters

5) Attached.

6) Simiarlity scores:
   a. VGG Representation:
      i. cat & mj1 = ~15%
      ii. cat & mj2 = ~14%
      iii. mj1 & mj2 = ~96%
   b. Pixel Representation:
      i. cat & mj1 = ~47%
      ii. cat & mj2 = ~62%

        iii. mj1 & mj2 = ~37%

   c. In VGG representation, the most similar pair is MJ1.jpg and MJ2.jpg
      In Pixel representation, the most similar pair is Cat.jpg and MJ2.jpg

7) The main problem with pixel representation is that it is very sensitive to its position on the image. It does not extract any other features, it simply compares one pixel to another. Pixel representation is considered a “low-level presentation”, and CNNs use an effective supervised learning solution, where the training process uses data that is labeled to close the semantic gap between low-level representations and higher-leveled ones.

8) I actually did something a little different I took the original image, cropped it a little, and returned used that new image to test. Here is the slightly cropped image:


Slightly cropped image of motorcycles. You can compare to the original

   a. VGG Representation provided a better summary of the image than Pixel Representation. They are defined below:
      i. VGG : “A pair of motorcycles parked in a bike slot.”
      ii. Pixel : “a white red and gray boat some people a bird and some water”
   b. I didn’t try a lot of examples, but for the most part, VGG was better, I think in part because it localizes where an image is in an object. I thought I could break that with cropping, but looks like the prediction worked out just fine for VGG.

9) 2 Images:


Bad VGG Representation : “A tall brown brick building next to a street with traffic”

Pretty sure that we used this example in class. While some elements do exist that are akin to both images (brick building, traffic, etc), the main idea of the images is not there. There is no *tall* brown brick building and there is nothing to represent a street. Still, very close.

Bad Pixel Representation : “Two men that are each trying to catch a Frisbee at the same time”

There are no similar elements to either two men, frisbee, or running around. Basically it’s completely off, but when I looked for an image that did have people playing frisbee, I noticed that the largest similarity was the abundance of green grass in both images. I would imagine that color from both images creates a stronger chance for pixel representation to predict based off of these factors.