Assignment 1 Report

The first thing I did was edit the given data as instructed (removing team attribute, as per instructor's directions). However, I left the names intact to be used for indexing, as well as the positions (which I could convert to integers for use)

The first order of business (for both problems) was correctly setting up the data for proper use. This involved using Python's csv library to parse the data spreadsheet. Next, means and standard deviations were calculated for each attribute. To simplify the code (and to reduce some tedium), I used the NumPy library to calculate standard deviations. For convenience, I treated all attributes as floating-points (including age). Next, the data itself must be standardized. I wrote my standardization code from scratch. I verified my results by standardizing a few values directly in the spreadsheet. There is a slight discrepancy between the values calculated by the spreadsheet, and the values calculated by the programs. My best conclusion would be that those discrepancies are due to the differences in how floating-points are handled by the spreadsheet and the programs, and I was also informed that a discrepancy that small was acceptable. Just to make sure, I tested the theory by using the SKLearn library's scale function from the preprocessing module, and there was still a discrepancy between the program-standardized values and the spreadsheet-standardized values.

Next, I wrote some test code that utilized pre-existing K-Means function. This code was only be used for testing and verifying the results of my own implementations, and was not used in the actual implementation. The library I used for the test code was scikit-learn. However, I ran into a problem trying to install this. For some reason, NumPy cannot be properly uninstalled/updated on OSX. I circumvented this issue by creating a virtual environment and then directly installing scikit-learn (which also includes NumPy and SciPy). All subsequent work would be done within this virtual environment (Python 2.7.10). Unfortunately, I couldn't get the k-NN scikit module to work, so I don't have any test code for k-NN.

I began my work on my kmeans function. I have two different containers meant for containing clusters. One will be carrying the current/updated clusters while the other will be carrying the cluster values from before the update. These two containers will be used to check whether or not the clusters have changed position. I then placed the initial clusters randomly. One important thing about the clusters is that they contain only numbers. This meant that I had to separate the raw numbers from the names and put them in their own containers. That way, the data points can be easily compared with the centroids. To calculate the Euclidean distance, I used the SciPy library. From there, I implemented the Kmeans algorithm as given. The clusters returned by the function are still standardized, however, so I just used the means and standard deviations from earlier to "unstandardize" the clusters. To shorten some of the output of the program, I rounded off the final cluster values to 5 places, which can be easily undone.

I finished what I believe to be a complete (but definitely not perfect) K-Means function. After that, I began work on the k-NN function. First, I used mostly the same code that I used in the K-Means program for parsing and standardizing the data. The only difference between the code for this part between the two programs is that the k-NN program splits table entries between a testing container and a training container. The data was standardized the same way as before.

One thing I did differently in this program was that I made much more efficient use of lists. In my K-means program, I used far too many lists that make reading the code somewhat difficult at times, mostly because I'm still rather inexperienced with Python and Python lists. Fortunately, some more research has shown me some better ways of storing my data in lists, making certain things much easier (like linking a data point with the corresponding distance from the). Unfortunately, due to the complicated nature of my code in the K-means program, I can't really go back and optimize the lists without rewriting some significant parts of the code. Of course, this would be a huge risk, if I messed something up, and it would also take a lot of time to do. As such, I have left my K-means program as is. But the k-NN program will be much cleaner and easier to follow, I believe. I've also split my program up into more functions, as opposed to having one giant function (like I did with K-means). The predicted/actual values were kept in standardized form.

Problem 1:
Part 1: Programming Section
Part 2: To get the results for this section, I just ran the program twice, and just changed the number of clusters that were passed to my function

k = 3:

```
Cluster 1: [3.00155, 26.62582, 47.53418, 16.79712, 16.49202, 2.36589, 5.39384, 0.4323,
0.55347, 1.60744, 0.2587, 1.81344, 3.78644, 0.4642, 0.48074, 1.06662, 1.44102, 0.70326,
0.69782, 2.18876, 2.88131, 1.41531, 0.52341, 0.32728, 0.91044, 1.49691, 6.35145]

Cluster 2: [3.00217, 26.57425, 56.92091, 24.07302, 20.36108, 3.08773, 6.86649, 0.4478,
0.67711, 1.93734, 0.26753, 2.41097, 4.9313, 0.48031, 0.49434, 1.37843, 1.84616, 0.7227,
0.89031, 2.76765, 3.65507, 1.79466, 0.65537, 0.42174, 1.12729, 1.78606, 8.23247]

Cluster 3: [3.02786, 26.56835, 59.97649, 36.81407, 23.68122, 3.89943, 8.62283, 0.44976,
0.83595, 2.3529, 0.28376, 3.06654, 6.27026, 0.48056, 0.49644, 1.86738, 2.45766, 0.73407,
1.02724, 3.31853, 4.34004, 2.27277, 0.78416, 0.49897, 1.40374, 1.95472, 10.50113]

Number of centers in cluster  1 :  36
Number of power forwards in cluster  1 :  55
Number of small forwards in cluster  1 :  41
Number of shooting guards in cluster  1 :  41
Number of point guards in cluster  1 :  42

Number of centers in cluster  2 :  20
Number of power forwards in cluster  2 :  18
Number of small forwards in cluster  2 :  17
Number of shooting guards in cluster  2 :  19
Number of point guards in cluster  2 :  18

Number of centers in cluster  3 :  33
Number of power forwards in cluster  3 :  30
Number of small forwards in cluster  3 :  34
Number of shooting guards in cluster  3 :  36
Number of point guards in cluster  3 :  35

k = 5:
Cluster 1: [2.97649, 26.59067, 47.55375, 16.97407, 16.50192, 2.37254, 5.39773, 0.43295,
0.54926, 1.59669, 0.25707, 1.82454, 3.80131, 0.46486, 0.48095, 1.06733, 1.44236, 0.70173,
0.70887, 2.20945, 2.91295, 1.41436, 0.52465, 0.33369, 0.90671, 1.5001, 6.36188]

Cluster 2: [2.89166, 26.55592, 55.58808, 25.09607, 20.09887, 3.08847, 6.83624, 0.44911,
0.64251, 1.83769, 0.25941, 2.44709, 4.99974, 0.47968, 0.49335, 1.41211, 1.89777, 0.71675,
0.93356, 2.81312, 3.74375, 1.75134, 0.64697, 0.44378, 1.12721, 1.7808, 8.23281]

Cluster 3: [3.02593, 26.56565, 59.76066, 36.78914, 23.5998, 3.88983, 8.58975, 0.45027,
0.81934, 2.31233, 0.28263, 3.07379, 6.27777, 0.48113, 0.49618, 1.86738, 2.45749, 0.73329,
1.0314, 3.32057, 4.34616, 2.27471, 0.78297, 0.49903, 1.40054, 1.95022, 10.46554]

Cluster 4: [3.01469, 26.57436, 54.76501, 25.81097, 20.16872, 3.11719, 6.96374, 0.44308,
0.68805, 1.96604, 0.26985, 2.43061, 4.99854, 0.47492, 0.49023, 1.44174, 1.92331, 0.71972,
0.87001, 2.75828, 3.62347, 1.82704, 0.65324, 0.41446, 1.15159, 1.74339, 8.36374]

Cluster 5: [3.14387, 26.66076, 56.38513, 24.80343, 20.52122, 3.12039, 7.0178, 0.44102,
0.74506, 2.11673, 0.28103, 2.37556, 4.90264, 0.47454, 0.49182, 1.39882, 1.85381, 0.72858,
0.81511, 2.69016, 3.50104, 1.87045, 0.66374, 0.38904, 1.14975, 1.75497, 8.38445]

Number of centers in cluster  1 :  36
Number of power forwards in cluster  1 :  54
Number of small forwards in cluster  1 :  41
Number of shooting guards in cluster  1 :  38
Number of point guards in cluster  1 :  37

Number of centers in cluster  2 :  21
Number of power forwards in cluster  2 :  15
Number of small forwards in cluster  2 :  7
Number of shooting guards in cluster  2 :  1
Number of point guards in cluster  2 :  0
Number of centers in cluster  3 :  32
Number of power forwards in cluster  3 :  29
Number of small forwards in cluster  3 :  32
Number of shooting guards in cluster  3 :  32
Number of point guards in cluster  3 :  35

Number of centers in cluster  4 :  0
Number of power forwards in cluster  4 :  0
Number of small forwards in cluster  4 :  1
Number of shooting guards in cluster  4 :  0
Number of point guards in cluster  4 :  1

Number of centers in cluster  5 :  0
Number of power forwards in cluster  5 :  5
Number of small forwards in cluster  5 :  11
Number of shooting guards in cluster  5 :  25
Number of point guards in cluster  5 :  22
```

Observations: My results seem to follow along with the results given by the built-in KMeans module, except for some concerning disparities. My only real hypothesis on why the disparity is there would be that, since my implementation is very basic, and doesn't come with all the optimizations that no doubt exist within the built-in KMeans module, it won't be identical to the results returned by the KMeans module. At the very least, the position and age values for the centroids are fairly close between my implementation and the built-in implementation.

Another thing that I noticed was that the distributions of my clusters would change slightly almost every time. I fixed this by seeding for the random value generator that I used. After seeding, my results are constant and repeatable.

Another strange observation is how cluster 4 (k=5) has only two data points in it. Again, this could possibly be attributed to the lack of optimization in my function. Another possible reason could be that my initial centroids are badly placed. As such, randomly selecting initial centroids may not be the best method of doing it.

Part 3: For every attribute that has a % counterpart, either the % counterpart or the original could possible be removed for being redundant. I say this under the assumption that if you have one (% or original) you can calculate/infer the other, if needed. But if it isn't necessary, then it can safely be removed from the data set (under my assumption). I would also remove eFG%, which looks like a near copy of FG%.

Part 4: For this part, I created another version of the csv containing only the attributes specified in the problem (NBAstats_less.csv). I also made a copy of my program, and altered that version of the program (kmeans_less.csv) so it would be compatible with the new csv file.

```
k = 3:
Cluster 1: [3.01731, 0.27338, 0.45859, 0.71071, 3.00841, 1.44071, 0.53899, 0.32925]
Cluster 2: [3.43444, 0.29329, 0.4733, 0.74245, 3.535, 2.36736, 0.77542, 0.36738]
Cluster 3: [2.57983, 0.24332, 0.49318, 0.70688, 4.33301, 1.67466, 0.64853, 0.55137]

Number of centers in cluster  1 :  16
Number of power forwards in cluster  1 :  44
Number of small forwards in cluster  1 :  47
Number of shooting guards in cluster  1 :  43
Number of point guards in cluster  1 :  19

Number of centers in cluster  2 :  0
Number of power forwards in cluster  2 :  1
Number of small forwards in cluster  2 :  27
Number of shooting guards in cluster  2 :  52
Number of point guards in cluster  2 :  76

Number of centers in cluster  3 :  73
Number of power forwards in cluster  3 :  58
Number of small forwards in cluster  3 :  18
Number of shooting guards in cluster  3 :  1
Number of point guards in cluster  3 :  0

k = 5:
Cluster 1: [3.02582, 0.27264, 0.45858, 0.71073, 3.0101, 1.446, 0.5384, 0.32897]
Cluster 2: [3.0084, 0.27021, 0.47491, 0.72025, 3.62689, 1.82773, 0.65378, 0.41618]
Cluster 3: [2.57786, 0.24268, 0.49248, 0.70754, 4.32533, 1.68322, 0.64585, 0.55136]
Cluster 4: [3.43244, 0.29216, 0.47294, 0.74245, 3.53611, 2.36892, 0.77494, 0.36839]
Cluster 5: [3.00811, 0.2723, 0.47622, 0.7191, 3.62893, 1.81203, 0.65861, 0.41511]

Number of centers in cluster  1 :  16
Number of power forwards in cluster  1 :  41
Number of small forwards in cluster  1 :  45
Number of shooting guards in cluster  1 :  44
Number of point guards in cluster  1 :  19

Number of centers in cluster  2 :  0
Number of power forwards in cluster  2 :  1
Number of small forwards in cluster  2 :  0
Number of shooting guards in cluster  2 :  0
Number of point guards in cluster  2 :  0
Number of centers in cluster  3 :  73
Number of power forwards in cluster  3 :  58
Number of small forwards in cluster  3 :  13
Number of shooting guards in cluster  3 :  0
Number of point guards in cluster  3 :  0

Number of centers in cluster  4 :  0
Number of power forwards in cluster  4 :  0
Number of small forwards in cluster  4 :  25
Number of shooting guards in cluster  4 :  50
Number of point guards in cluster  4 :  76

Number of centers in cluster  5 :  0
Number of power forwards in cluster  5 :  3
Number of small forwards in cluster  5 :  9
Number of shooting guards in cluster  5 :  2
Number of point guards in cluster  5 :  0
```

Observations: As expected, the position distributions are different now than when using the dataset with more attributes. However, cluster 2 and cluster 5 (k=5) are experiencing the problem of having very few data points. Again, it may be due to the lack of optimization in my implementation, or the initial centroid placement.

Problem 2:
Part 1: Programming Section
Part 2: k = {1, 5, 10, 30}
k = 1: Accuracy = 64%
k = 5: Accuracy = 66%
k = 10: Accuracy = 66%
k = 30: Accuracy = 59%

As expected, the algorithm is more accurate when it is looking for a relatively smaller number of neighbors. Too many neighbors could easily increase the uncertainty in the prediction, especially with a dataset this complex. Out of the k-values presented here, k = 5 would be the best choice. Although it has the same accuracy as k = 10, it will take less time and use less resources, since it is looking at half as many neighbors. I'm not surprised that the accuracies aren't higher, considering the complexity of the dataset.

Part 3: For this part, I created a different file (knn_less.py) in the same fashion as kmeans_less.py.
k = 1: Accuracy = 86%
k = 5: Accuracy = 84%
k = 10: Accuracy = 82%

k = 30: Accuracy = 79%

Unlike with the full dataset, the abridged dataset seems to result in accuracies that decrease as the number of neighbors increases. However, the accuracy is greater than before, all across the board. Most likely, the increase in accuracy corresponds with the decrease in the dataset complexity. Also, it would seem that finding the best accuracy is more of a trial-and-error process of running the program with several different values, but I think it is still fairly safe to assume that larger values of k will result in less accurate predictions.

Conclusions

Despite the problems that rose up along the way, I think this has been a very good and useful assignment. I certainly have no more confusion about k-Means and k-NN anymore. This assignment has also greatly helped me in entering Python, which is a language that I plan to continue using for a long time. I certainly don't have flawless implementations, but I imagine that a flawless implementation of these algorithms would take far, far longer than the amount of time spent on this assignment.

Timeline:

- Edited data as per instructions
- Wrote code to parse and standardize the data
- Wrote code for test program to test K-Means results
- Implemented own K-Means algorithm
- Verified results of K-Means algorithm
- Reported on results of K-Means algorithm and answered additional homework problems
- Wrote code for test program to test K-NN results
- Implemented own K-NN algorithm
- Verified results of K-NN algorithm
- Reported on results of K-NN algorithm and answered additional homework problems