ADL Recognition for Wrist-worn Accelerometer

For this experiment we were given a data set of accelerometer data and were tasked with classifying the data as one of 14 actions (Use_telephone', 'Standup_chair', 'Walk', 'Climb_stairs', 'Sitdown_chair', 'Brush_teeth', 'Comb_hair', 'Eat_soup', 'Pour_water', 'Descend_stairs', 'Eat_meat', 'Drink_glass', 'Getup_bed', 'Liedown_bed'). In order to do this we used k-means clustering with a random forest classifier. By using k-means clustering we would be able to build a better feature set for our random forest classifier to learn from. In order to do k-means clustering you must choose a proper d and k value.

Choosing the d value

For choosing the d value I initially followed what was given in lectures. The wrist accelerometer data was recorded every 32 Hz or 32 times per second. Using this information I took each individual (X, Y, Z) vectors from the raw data and built a 96 dimension vector or 32 x 3 where $d = 96$. This gave me a data point for every second of movement which was a nice dimension to use for the data.

After I got the rest of k-means working I tried tuning the d value but nothing improved my accuracy so I decided to focus on tuning the k value. 96 being the d value makes sense because it gives the finest grain data piece with the least amount of sensor noise. If we used every single 32Hz sensor value then the data set would have been too large and also that data is very dirty because the sensor could have errors in those short term reads. Getting the sensors data every 1 second is enough amount of data to handle the noise. We could have also ran the sensor data through a Kalman filter to clean up the noise but that didn't seem necessary for the scope of this project as we weren't relying on the sensor as a feedback device for a time/accuracy sensitive control system like in robotics.

Choosing the k value

Initially I started out with a k-value of 14 as the data was classified in 14 possible actions. Using this assumption I achieved fairly good results:

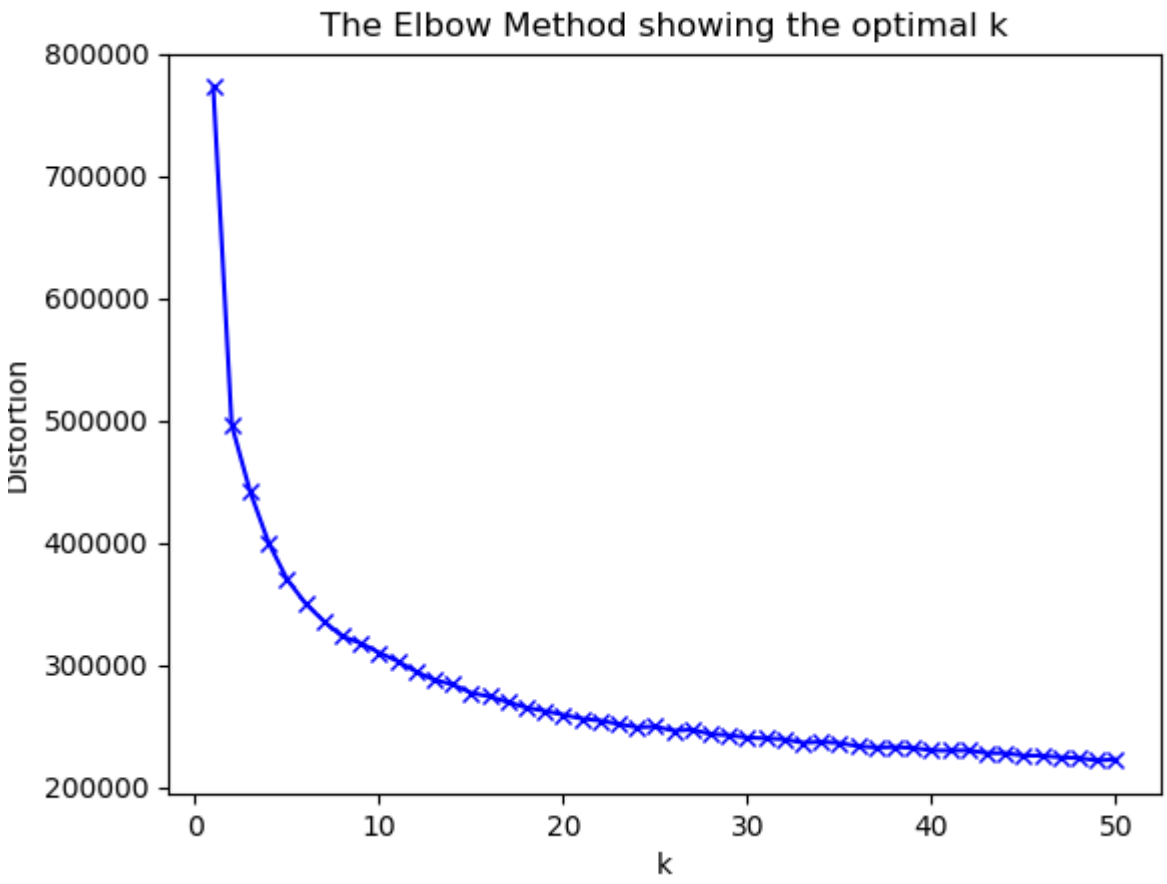
```
Final d = 96
Final k = 14
Final Accuracy: 0.8173076923076923
Confusion Matrix:

[[263   0   0   0   0   0   0   0   0   5   4   0   0   0]
 [  1 240   0   4   0   0   0   0   2   0  14   0   0 111]
 [  5   0 120   0  28   0   0   0   0  15  19   7   0  11]
 [  0   1   0 108   0   0   0   0   0   0   0   0   0  21]
 [  0   0   0   0 326  11   0   9   0  43   0   3   0   0]
 [  0   0   0   0   1 305   0   0   0   0   0   2   0   0]
 [  0   0   0   0   0   0  54   3   0   3   0   0   0   0]
 [  0   0   0   0  16   9   4 338   0  18   0   2   0   1]
 [  0   1   3   0   0   0   0   0  54   0  16   0   0  32]
 [  0   0   1   0  57   3   0  10   0 283   0   6   0   0]
 [  0   7  10   0   2   0   0   6   7   0 155   0   0  29]
```

```
 [  1   1   4   0  59   5   0  34   0  30   0  97   1   0]
 [  5   0   0   0   4   0   1   2   0   2   0   3 115   0]
 [  0   4   0   5   3   0   0   0   0   3  11   0   0 857]]
```

The data above shows that with a k value of 14 the k-means clustering and random forest classifier had an 81.7% accuracy.

Then I began to play with the k value using some techniques I had read about. First, I attempted to use the elbow test. The elbow test is when you compute the k-means for n k values and find the distortion of each k value then choose the k-value at the elbow of the graph or the point with low distortion and when all the other points after it have very marginal improvements. For this experiment I ran a loop from $k = 1$ to $k = 50$ and printed the following graph of distortion values:



From this graph we can see that 14 is the point at which the graph starts to flatten and the effects of the k value go down. But I still wanted to see what raising the k-value more would do to the classifier, so I made the k-value 20 and got these results:

```
Using elbow method for choosing k-value
Final d = 96
Final k = 20
Final Accuracy: 0.8121301775147929
Confusion Matrix:

[[260   0   0   0   0   0   0   0   0   7   4   0   0   1]
 [  1 245   0   3   0   0   0   0   0   0  14   0   0 109]
```

```
 [  1   1 115   0  35   0   0   0   0  17  18   3   0  15]
 [  0   3   0 108   0   0   0   0   0   0   0   0   0  19]
 [  0   0   0   0 317  12   0  10   0  46   0   7   0   0]
 [  0   0   0   0   1 307   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  50   4   0   6   0   0   0   0]
 [  0   0   0   0  17   7   4 336   0  19   0   2   2   1]
 [  0   1   2   0   0   0   0   0  60   0  13   0   0  30]
 [  0   0   0   0  55   2   0  12   0 285   0   6   0   0]
 [  0   8   6   0   1   0   0   5   5   0 155   0   0  36]
 [  1   1   4   0  64   7   0  38   0  26   0  89   2   0]
 [  7   0   0   0   3   0   1   2   0   1   0   5 113   0]
 [  0   7   0   4   3   0   0   0   0   3  12   0   0 854]]
```

Interestingly, when the k-value was 20 the accuracy dropped to 81.2%. I wasn't satisfied and decided to run a loop from $k = 1$ to $k = 1000$. When this finally finished, the k-value with the highest accuracy was $k = 644$. Here were the results:

```
Final d = 96
Final k = 644
Final Accuracy: 0.8293885601577909
Confusion Matrix:

[[263   0   0   0   0   0   0   0   0   4   4   0   0   1]
 [  0 252   1   3   0   0   0   0   0   0  13   0   0 103]
 [  2   0 125   0  29   0   0   1   0  17  17   4   0  10]
 [  0   2   0 107   0   0   0   0   0   0   0   0   0  21]
 [  0   0   0   0 334  11   1   7   0  35   0   2   2   0]
 [  0   0   0   0   1 304   0   0   0   1   0   2   0   0]
 [  0   0   0   0   0   0  52   5   0   3   0   0   0   0]
 [  0   0   0   0  16   9   4 337   0  19   0   2   1   0]
 [  0   2   1   0   0   0   0   0  59   0  13   0   0  31]
 [  0   0   0   0  51   1   0  12   0 293   0   3   0   0]
 [  0   2   5   0   1   0   0   5   4   0 162   0   0  37]
 [  1   1   5   0  62   4   0  30   0  25   0 102   2   0]
 [  5   0   0   0   3   0   1   2   0   2   0   5 114   0]
 [  0   4   1   2   3   0   0   0   0   3  10   0   0 860]]
```

From these results the accuracy reached 82.9% which was higher than the 81.7% when k was 14, but this better accuracy came at a cost. With a k-value of 644 the code took much longer to run and for only a small improvement in accuracy wouldn't be worth it.

But I wanted to look at two more thing before coming to conclusions. In all the tests above, I was adding the k-means as an additional feature the random forest classifier could use. I wanted to see how the random forest classifier would perform with just the k-means data and the other original data removed. Here are the results:

```
K-Means + Random Forest only on K-means data
Final d = 96
Final k = 658
Final Accuracy: 0.6442307692307693
Confusion Matrix:
```

```
[[222   0   6   0  13   0   0   0   0  15   7   0   4   5]
 [  0 176   1  23   1   0   0   0   0   0  33   0   0 138]
 [ 14   3  75   0  21   0   0   6   0  18  41   2   2  23]
 [  0  11   0  86   0   0   0   0   0   0   0   0   0  33]
 [  4   0  28   0 220  25  18  19   0  49   0  21   8   0]
 [  0   0   0   0   4 293   0   0   0   1   0  10   0   0]
 [  0   0   0   0   8   2  34   7   0   9   0   0   0   0]
 [  2   0   6   0  35  20   0 304   0  10   0   5   3   3]
 [  0   3   9   0   0   0   0   0  28   0  48   0   0  18]
 [  3   0   9   0  84   9   6  25   0 183   5  24   9   3]
 [  2  17  12   0   2   0   0   9   3   0 144   2   0  25]
 [  6   2  22   0  74   9   0  43   1  25   0  44   6   0]
 [  5   0   3   0   1   2   3   8   0   6   0   3 101   0]
 [  1  69  14  26   5   0   0   2   4   1  54   4   0 703]]
```

From the above data, we can see that using just the k-means data for the random forest classifier wasn't very accurate (64.4%). Also, I ran another long loop to find the optimal k-value for the random forest classifier and this was $k = 658$. This means that computation also took a very long time with a k-value this large. What we can see from this is that the data wasn't really clustered in a way that would have been useful to classify these 14 distinct tasks.

Finally, I also wanted to see how the random forest classifier worked without k-means as we had labels given with the data so we didn't necessarily need to use the unsupervised technique of k-means. Here are the results:

```
 Only Random Forest Classifier:
Final d = n/a
Final k = n/a
Final Accuracy: 0.8163214990138067
Confusion Matrix:

[[263   0   1   0   0   0   0   0   0   4   4   0   0   0]
 [  0 238   0   3   0   0   0   0   0   0  14   0   0 117]
 [  3   0 119   0  23   0   0   2   1  18  22   5   0  12]
 [  0   2   0 108   0   0   0   0   0   0   0   0   0  20]
 [  0   0   0   0 332  10   2   5   0  39   0   4   0   0]
 [  0   0   0   0   3 304   0   0   0   0   0   1   0   0]
 [  0   0   0   0   0   0  52   5   0   3   0   0   0   0]
 [  0   0   0   0  15   8   4 339   0  17   0   3   2   0]
 [  0   1   2   0   0   0   0   0  63   0  12   0   0  28]
 [  0   0   0   0  56   3   0  10   0 287   0   4   0   0]
 [  0   5   8   0   0   0   0   5   4   0 153   0   0  41]
 [  1   1   6   0  61   6   0  38   0  30   0  89   0   0]
 [  5   0   0   0   8   0   3   2   0   3   0   5 106   0]
 [  1   4   0   1   3   0   0   0   0   3  13   0   0 858]]
```

As we can see from this, the random forest classifier with the given data is pretty accurate and adding the k-means to this data didn't improve it that much. Now in this situation we were given a data set that was labeled with the actions which isn't necessarily accurate to a real world use-case of this very same problem. For example, if I was an employee at Fitbit tasked with trying to classify different actions a user is

performing based off of accelerometer data in order to log the daily actions of that user then I most likely wouldn't have a clean and labeled dataset. What I would have is a daily dump of the raw accelerometer data taken every 32 Hz. The first step is trying to figure out what this data means and if there are any distinct categories that the accelerometer data falls in. In order to do this, I need to use some sort of unsupervised learning technique as there is no labeled data. The simplest solution to this is k-means. By performing k-means on the data and finding the ideal k-value by using the "elbow" test, I would then know if and how many clusters are formed by the data. Using these k clusters I could then train a classifier like a random forest classifier to classify the data in one of these k categories. Lastly, I would then take a test user set and have them perform tasks throughout the day and log what they are doing. With these logs we could then figure out what the categories our k-means process found are (e.g.: running, walking, sitting, brushing, etc.). The process I described above shows the value in unsupervised techniques like k-means especially in real world scenarios where data is usually messy and unlabeled.