Shaun Howard 9/2/2015
EECS 484 HW1 smh150

Seeding Clusters:

To begin, I will describe my efforts to seed clusters. I have chosen several different ways to get random clusters. I am using the randi() Matlab function to generate a random integer index to use to seed each cluster. Initially, I was not seeding and just calling randi() to get my indices. I began to notice duplicate seeds for clusters, so I knew this was a bad choice. I then chose to seed with various values and various combinations of nclusters and max_passes. I noticed a lot of replicated results after a small number of iterations. Specifically, I chose the seed value of 12345 at 15 clusters and 20 passes, and I kept getting the return rate of 5.1%. Then I chose the seed value of 400 to obtain a return rate of 5.5%. Subsequently, I tested a fair amount of times with the rng('shuffle') seeding based on the time of function call, which produced results of return rates between 1 % and 12 %, however, the average return rate was ~6 %. Hence, I stuck with the shuffled seeding method because it produced the best return rates, which were usually greater than the index fund return rate of ~4%.

Cluster Descriptions:

The clusters that I have been generating using the shuffled random seed technique have partial homogeneity. There will be approximately 6 out of 15 clusters that are relatively homogeneous, while the rest tend to have outliers which make those clusters more spread and less tightly packed.

Cluster populations tend to be fairly evenly distributed in population on average. There are some clusters with few patterns and some with many, while most tend to have a similar amount. Hence, the cluster populations, on average, follow a normal distribution. I have occasionally noticed clusters with no patterns in them, however, I believe this was a bug in the way clusters were initially seeded at one time. This did/does not happen very often, just once every 10's of iterations. I added the check in reassign_patterns to make sure the current cluster has a population greater than one (or is zero) before executing the remove_pattern_from_cluster code as a test, since that code would have removed the given pattern from a cluster no matter what size the population was. This fixed the population of zero issue, but it may have altered my return rate results. The population distribution remained the same as earlier though.

Additionally, I have experimented with a few other things to try to fix the population bug in this realm, such as checking if the closest_dist or the curr_dist were in a range between 0 and 0.00001 when finding the closest cluster in find_closest_cluster, and if they were, to take either as the closestClust and break from the loop. That change produced variable results as I was not getting the population of zero anymore, but some clusters had only one value at the beginning or later on (the initial seed, it seemed). It seems that the issue is somehow in the find_closest_cluster code. I believe that the comparisons in distance are not producing correct results. This could be due to my usage of the norm() function, which finds the distance from one vector to another; however, the root of this issue still remains a mystery.

The performance of my clusters is random (due to the nature of generation), but I feel as though my clusters are better than average because the rates of return they produce are usually comparable to or better than that of the index fund. With that being said, sometimes bad picks do happen, which result in lower return rates as compared to that of the index fund. Typically, even the bad picks give return rates of greater than 0%, so the investor would still gain some sort of

profit from those. What I am judging as performance could also be caused by my methods of investment selection, which I will discuss next.

Choosing Investments:

Here is my code for choosing investments:

```
clusters_to_pick = zeros(nclusters, 1);

Q1 = quantile(cluster_attributes,0.25);
Q3 = quantile(cluster_attributes,0.75);
quantile_diff_over_two = (Q3 - Q1) / 2;

avg_centroid = mean(cluster_centroids);

for iclust=1:nclusters
    if (Q1 < cluster_attributes(iclust) && ...
            cluster_attributes(iclust) < Q3) || ...
            (cluster_populations(iclust) > 10 && ...
            cluster_attributes(iclust) >= quantile_diff_over_two) || ...
            (sum(cluster_centroids(iclust)) > sum(avg_centroid))
        clusters_to_pick(iclust) = 1;
    end
end
```

I will explain this code step-by-step:

1. Gather the 25% and 75% quantiles for all of the training cluster attributes.
2. Calculate the difference of the quantiles and divide this value by 2.
3. Calculate the mean centroid value out of all the cluster centroids.
4. Determine whether to pick the current cluster:
    a. Check if the cluster attribute is between the 25% and 75% quantiles
    b. Or, check if the cluster population is greater than 10 and if the cluster attribute is greater than or equal to the quantile difference over 2
    c. Or, check if the sum of the cluster centroid values is greater than the sum of the average cluster centroid values (from the average cluster centroid found previously)

The following is the code that I use to calculate the average rate of return:

```
population = 0;
avg_return_rate = 0.0;
num_chosen_clusters = 0;

%find average return rate of investments
for iclust=1:nclusters
    if clusters_to_pick(iclust) == 1
        avg_return_rate = avg_return_rate +
(val_cluster_attributes(iclust)*val_cluster_populations(iclust));
        population = population + val_cluster_populations(iclust);
        num_chosen_clusters = num_chosen_clusters + 1;
    end
end
```

```
%only calculate average return rate if population > 0
if population > 0
    avg_return_rate = avg_return_rate / population;
else
    avg_return_rate = 0.0;
end

fprintf('avg return rate is: %f\n', avg_return_rate);

fprintf('number of clusters chosen from is: %d\n', num_chosen_clusters);
```

What this code does is takes the validation cluster attributes of the clusters chosen with training data, multiplies them by the population of that cluster, and adds them to the avg_return_rate. In tandem, the population is summed for each cluster that is chosen from, and then once all attributes have been accounted for, the average rate of return is calculated using the summed avg_return_rate and population variables. Finally, I print the average rate of return and the number of clusters chosen from to the console.

The average rate of return that my program receives is between 4% and 7%. As mentioned earlier, there are outliers to that generalization which may range from -1% to 12%. I would give the following estimates for rates of return by what I have observed throughout the better half of the life of my program (including it now):

Return rate <= 0% : 5% of time

Return rate <= 4% : 40% of time

Return rate <= 7% : 85% of time

Return rate <= 9% : 93% of time

Return rate <= 11% : 97% of time

Return rate <= 13% : 100% of time

Optimizations and Convergence:

From what I have noticed, a cluster count around 20 has been best for me. I have experimented with multiple values, from 2 to 40, but some are either too general or too specific and end up with either too many or too little patterns in them. 20 clusters seems to be a good number to stick with, either due to the amount of data or just the way the data is clustered together using Euclidean distance of scaled feature values.

As far as number of passes is concerned, I have tested with 2 to 20 passes. I have chosen to stick with 20 passes at this time, although most of my testing was done with 10 passes. I noticed that the more passes, the better the clustering was for whatever reason. It seemed like return rates became more consistent when I increased the number of passes. The coherence of the clustering seemed relatively similar in the range of 10 to 20 passes of clustering, but I just chose 20 and stuck with it for what seemed like a more converged clustering. As an update, I tested with 40 passes, and I believe that although some picks had better return rates, the estimates of values given above were relatively similar. Hence, after a certain number of passes, in my case 20, the clustering algorithm would simply re-arrange already well-placed patterns.

Conclusion:

What I have concluded is that there are only so many iterations of clustering that can be performed before the algorithm begins to worsen clusters. I believe that a better way to pick initial seeds than randomly would greatly improve the clustering. I also think that using a weighting metric more intricate than Euclidean distance could help cluster patterns in better ways. Both of these would also make clusters more unique and probably more tightly correspondent as far as pattern containment is concerned.

I have also gleaned that there are numerous ways to make assumptions about data in general. So many different mathematical calculations can be performed to alter clustering how the user wants it to be. The metrics I picked for choosing investments are arbitrary from guess and check. I could have picked five other metrics to check instead and probably could have attained similar results.

All in all, clustering seems very interesting to me, but I would like to see a better implementation of it to really convince myself that it is of real-world use.