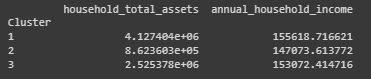
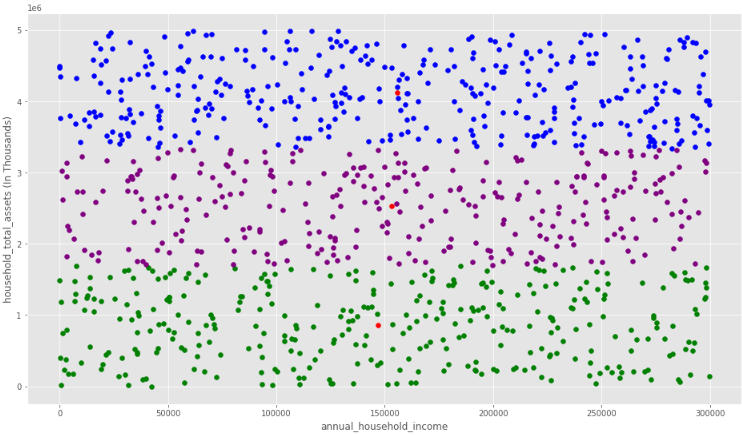
# Question 1 - Clustering

**a)** Please see KMeans1.py for implementation

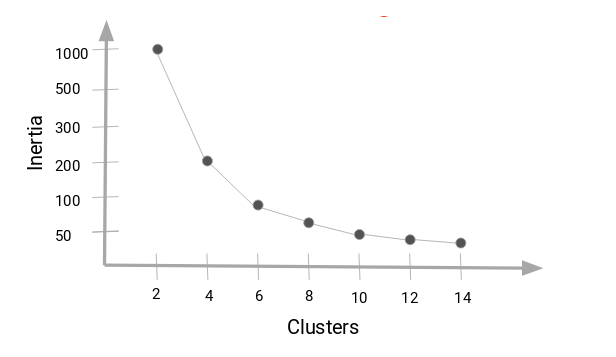
**b)** Please see the final three centroids below

****

The centroids form a vertical line with a slight slant. This can be interpreted as household income only has a slight effect on household total assets. The three clusters segment the customers into groups based on household total assets. For the green cluster(lowest household assets between 0 and ~175k), the centroid sits just left of 150000 annual household income, while the centroid in the blue cluster(highest household assets between 335k-500k) sits just to the right of 150000 annual household income.



**c)** One way to decide on the optimal number of clusters is to plot a graph, where the x-axis represents the number of clusters, and the y-axis is an evaluation metric like inertia or the Dunn index.



(Sharma, 2019)

Inertia lowers as we increase clusters, however as we add more clusters the inertia reduces less. Increasing the number of clusters also increases the computation cost, so choosing the correct K becomes a matter of getting the acceptable amount of inertia for the least amount of computation cost.

Another method is via the use of K-means++, where the steps are:

* Choose one of your data points at random as an initial centroid.
* Calculate D(x), the distance between your initial centroid and all other data points.
* Choose your next centroid from the remaining data points with probability proportional to D(x)2

Repeat until all centroids have been assigned.

By augmenting k-means with a simple, randomised seeding technique, we obtain an algorithm that is O(log k)-competitive with the optimal clustering. Experiments show our augmentation improves both the speed and the accuracy of k-means, often quite dramatically.(Arthur & Vassilvitskii, 2006)

# Question 2 - Weighted KNN

**a)**

| For 1NN, the class label of the new data point is determined solely by the closest neighbour.If K is one, or a small number, the classification is sensitive to noise points. |  |
| --- | --- |

,

| For KNN, the class label of the new data point is determined by the Kth closest neighbours, for example to the right we see K=2, which fails to determine a class label as there is 1 - and one +(which is generally why an even value for K is avoided). When K=3, the data point’s class label is +. |  |
| --- | --- |

| For nNN, the class label of the new data point is determined by the most common class label in the training sample. For large values of K, bias is increased and the model becomes computationally expensive. | In general practice, choosing the value of k, we use K=sqrt(N), where N is the number of samples in your training dataset. |
| --- | --- |

To overcome these disadvantages, weighted-KNN is used. Weighted-KNN implements the inverse distance function, implying as the distance increases weight decreases, and as distance decreases, weight increases.

**b)** See WeighteddNN.py for implementation

**c)** Yes, through playing around with the query point and K the weighting factor conforms to my intuition pretty well. However, it could be improved by assigning training sample points closer to the new data point a higher weight than those that are far away. Achieved by:

1. Compute all euclidean distances, and find k-nearest distances.
2. Compute the inverse of each distance, find the sum of the inverses, then divide each inverse by the sum.
3. Use each weight as a vote for its associated class, the predicted class is the one with the greatest vote(McCaffrey, 2019).

# Question 3 - Multi-Layer NN

**a)** I decided to use the ReLU activation function because it is the default activation function as it is easier to train and often achieves better performance than the others. While not an issue here, the sigmoid and hyperbolic tangent activation functions cannot be used in networks with many layers due to the vanishing gradient problem.

Since we are only using one hidden layer, I decided to up the neurons to 500. It increases the computation time a little but it's worth it to increase precision, recall, f1-score and support numbers.

**b)** The implementation was evaluated using a confusion matrix, where accuracy was 0.52.

**c)**

| **Hidden Layer** | **Neurons** | **Accuracy** |
| --- | --- | --- |
| **1** | **50** | **0.54** |
| **4** | **50** | **0.51** |
| **7** | **85** | **0.52** |
| **3** | **30** | **0.54** |
| **2** | **25** | **0.54** |
| **6** | **50** | **0.53** |
| **8** | **100** | **0.51** |
| **4** | **10** | **0.49** |
| **5** | **5** | **0.49** |
| **9** | **20** | **0.49** |

**(d) If the ‘optimal’ number of hidden layers and neurons is determined by the process described in (c), discuss any drawbacks of this approach. Your answer in this question should not exceed 200 words. (5 marks).**

We can see above that the number of hidden layers doesn’t really affect accuracy, and the best number of neurons - within the tested range - is probably between 25-50.

A drawback to picking random hidden layers and Neurons for testing is that there are two variables affecting the performance metric. Keeping the Hidden Layer as a constant and changing the Neurons would be a better approach.

In fact, during my research I found that one hidden layer is sufficient for the large majority of problems similar to this, and the optimal size of the hidden layer is usually between the size of the input and the size of the output layers.

# References

Sharma, P. (2019, August 19). The Most Comprehensive Guide to K-Means Clustering You’ll Ever Need

Source. <https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/?#h2_13>

Arthur, D & Vassilvitskii, S. (2006). k-means++: The Advantages of Careful Seeding.

Technical Report. Stanford. <http://ilpubs.stanford.edu:8090/778/?ref=https://githubhelp.com>

McCaffrey, J. (2019, April 10). Weighted k-NN Classification Using Python

Source. <https://visualstudiomagazine.com/articles/2019/04/01/weighted-k-nn-classification.aspx>