RCluster: An Efficient and Randomized Clustering Algorithm

Sharan Duggirala

San Jose State University

December 1, 2017

Outline

This Project was taken under the guidance of Prof. **Teng Moh** from San Jose State University.

Introduction and the Problem Statement

Introduction [1]

- K-Medians clustering is a type of **unsupervised learning**, which is used when you have **unlabeled data** (i.e., data without defined categories or groups).
- The goal of this algorithm is to **find groups in the data**, with the number of groups represented by the variable $\boldsymbol{K}$.
- The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on **feature similarity**.
- The results of the K-Medians clustering algorithm are:
  - *The centroids of the K clusters, which can be used to label new data*
  - *Labels for the training data (each data point is assigned to a single cluster)*

The Clustering Problem

The K-Medians Clustering uses **iterative refinement** to produce a final result.The algorithm inputs are the number of clusters K and the data set. The algorithm, in fact, can be divided into two stages:

The Data Assignment Step (*Step 1*)

Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, based on the squared **Manhattan distance**. More formally, if c_i is the collection of centroids in set C, then each data point x is assigned to a cluster based on:

$$\underset{c_i \in C}{\arg\min}\, dist(c_i, x)$$

where $dist()$ is the standard $L1$ Manhattan distance. Let the set of data point assignments for each i^{th} cluster centroid be S_i.

The Clustering Problem 2

The `The Centroid Update Step` (*Step 2*)

In this step, the **centroids** are recomputed. This is done by taking the median of all data points assigned to that centroid's cluster.

$$C_{\text{i}} = Median(S_{\text{i}})$$

The algorithm iterates between steps **1** and **2** until a **stopping criteria** is met. Some common stopping criteria are given below:

- No data points change clusters
- The sum of the distances is minimized
- Some maximum number of iterations is reached.

This algorithm is guaranteed to **converge** to a result[2]. Please take a look at the cited paper, for the proof.

The Algorithm

```
Q = infinity
do
    for point in dataset
        min = infinity
        index = 0
        for i in k
            dist = distance(point, center[i])
            if dist < min
                min = dist
                index = i
        disjoint-sets.add(index, point)
    for i in k
        center[i] = median(disjoint-set.get(i))
    sum = 0
    for i in k
        for point in disjoint-set.get(i)
            sum = sum + distance(point, center[i])
    oldQ = Q
    Q = sum
while (oldQ - Q) > eps 1
```

Difference between K-Medians and K-Means

There are only two simple differences between K-Means and K-Medians

- The distance to centroid calculated is the **Manhattan Distance** (**L1**) with K-Medians. However with K-Means, the difference is **L2**(**Euclidean Distance**). The difference can be shown below:

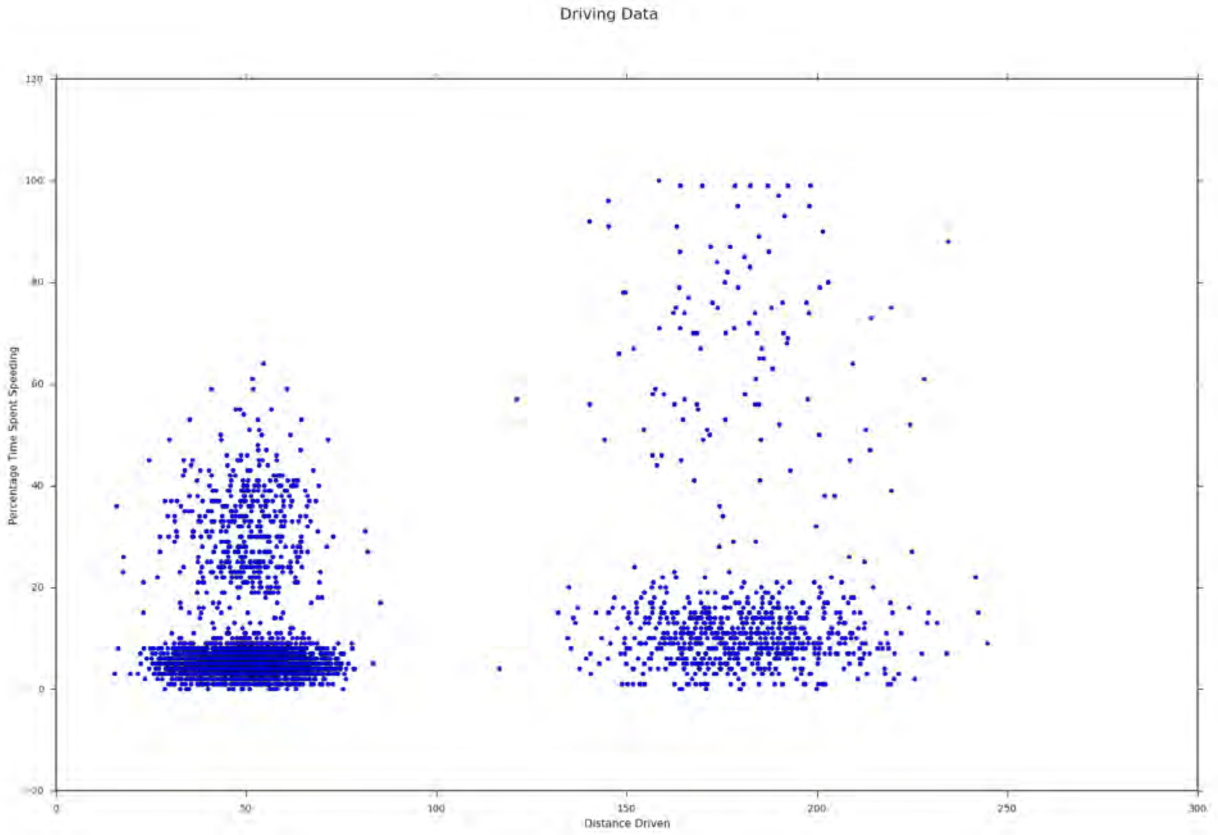
$$EuclideanDistance = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

$$ManhattanDistance = |x_1 - y_1| + |x_2 - y_2|$$

- When the centroid is computed again, K Means uses the mean of all the points which have affirmed that they are the closest to this particular point. K-Medians uses the median, thus giving their names.

Can we create a viable K-Medians Algorithm that can compare to the accuracy and the time efficiency of other clustering algorithms?

My Data Set


Driving Data
Percentage Time Spent Speeding
Distance Driven
120
100
80
60
40
20
0
-20
0
50
100
150
200
250
300

Observations about the Data

The data set being used for this experiment compares the **Distance Driven** by various truck drivers around the United States against the **Percentage of Time** they **spent Speeding**. We can observe a few things about the data set immediately:

- There seems to be **four** different **areas** where the data seems to be **congregated**. This will influence my **test bias** as the experiment proceeds.
- The data only has **two dimensions** and is therefore easy to test though human analysis. This is very important for the scope of this particular project.

This data is available on my Github in a `.csv` format (Linked at the end of this presentation)

The Existing Solutions

The Oligarchy of K Means

Oligarchy refers to the ancient Greek word which refers to a government, where the power of decisions is held by a select few people. As mentioned before, K-Means is extensively used within the industry, while K-Medians is barely used.

- K-Means enjoys library support from various technologies such as **XL Stat**, **C++** and various machine learning and data science libraries such as **SKLearn**.
- I struggled to find libraries or languages that support K-Medians. **Fortran** had a deprecated library, while there were user written code snippets within **C++** and **ELKI**.

The Oligarchy of K-Means

XLSTAT

your data analysis solution

C++

scikits

learn

machine learning in Python

The Current Situation of K-Medians

Fortran

C++

Environment for
DeveLoping
KDD-Applications
Supported by Index-Structures

My Solutions and Implementation

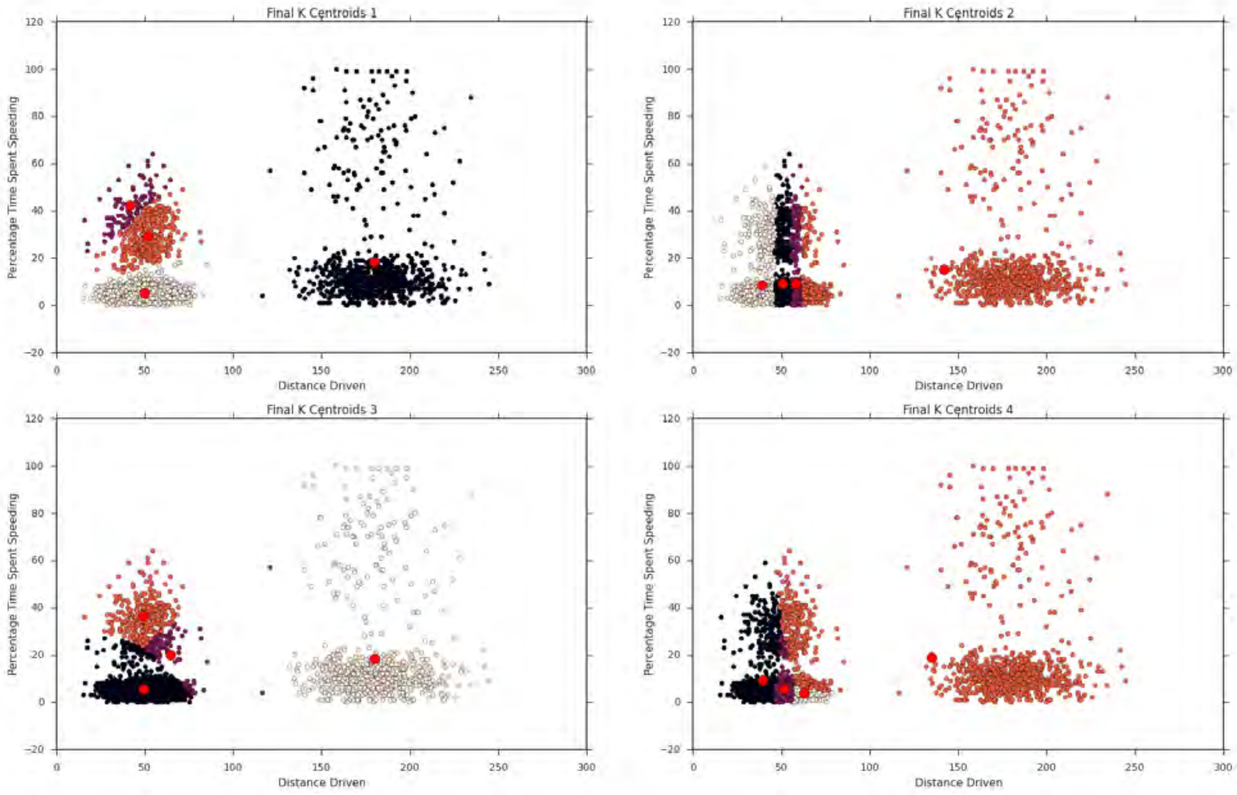
My Python Programs for K-Means and K-Medians

In order to better understand the theory and workings of K-Medians and K-Means, I decided to implement both these clustering algorithms within the `Python` Environment. There are many advantages to this selection:

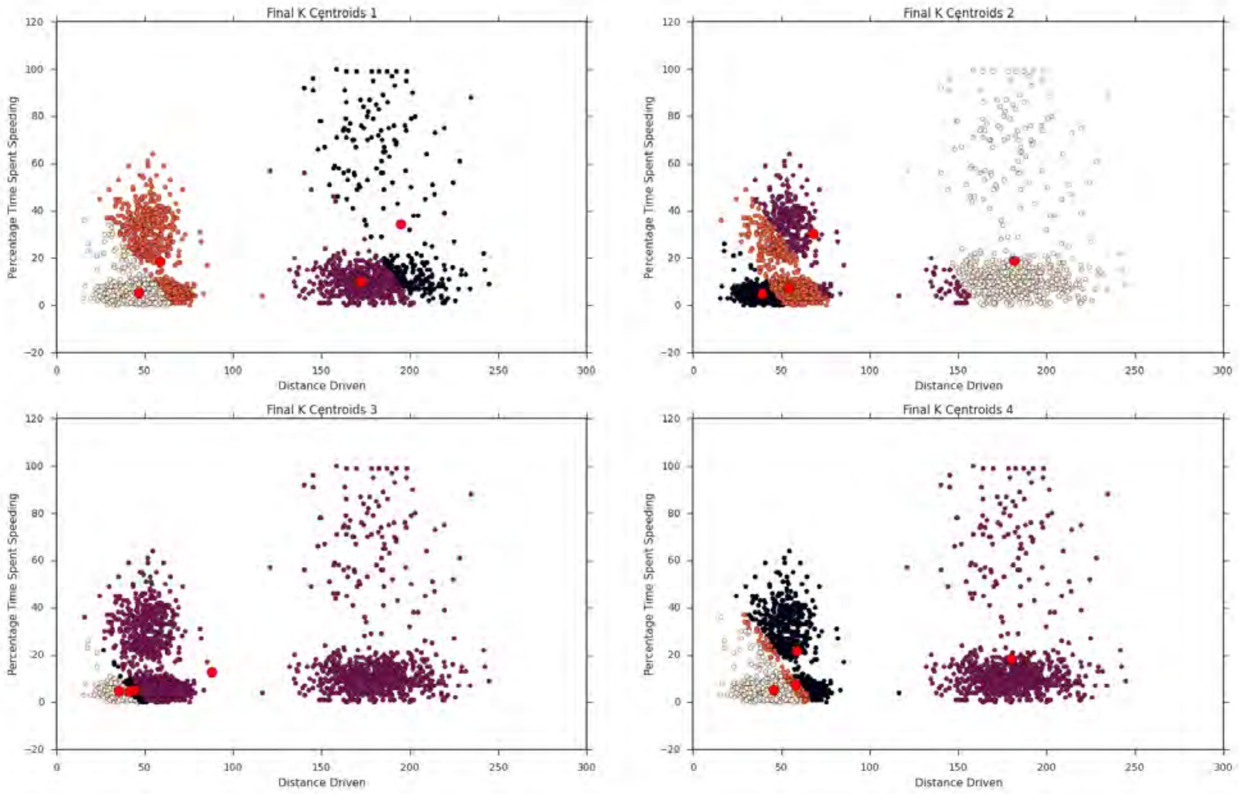
- `Python` contains many different libraries such as `Numba` and `Numpy`, which allow the access to a **multitude of functions** that are helpful for my research and programming. Other languages such as `C++` and `Java`, do not contain the convenience of such libraries.
- In fact, these libraries are **optimized** to work at a similar time efficiency as a `C++` program and therefore give us an encompassing idea of the industry standards.

The next two slides display my initial attempts at classifying the data set presented earlier within this lecture. The Code lies within the ***Github*** *Link*

My Attempt at using K-Means



My Attempt at using K-Medians



Evaluation of Naive Implementation

Holistically, there seems to be no difference between the accuracy of the results between my implementation of the K-Means algorithm and the K-Medians algorithm. In order to present K-Medians as a viable solution both the **time efficiency** and the **accuracy** must be optimized.

Let us Improve the Time Efficiency of Finding the Median

Refresher on Randomized Medians

Randomized Median Algorithm:

Input: A set S of n elements over a totally ordered universe.

Output: The median element of S, denoted by m.

1. Pick a (multi-)set R of $\lceil n^{3/4} \rceil$ elements in S, chosen independently and uniformly at random with replacement.
2. Sort the set R.
3. Let d be the $\left(\left\lfloor \frac{1}{2}n^{3/4} - \sqrt{n} \right\rfloor\right)$th smallest element in the sorted set R.
4. Let u be the $\left(\left\lceil \frac{1}{2}n^{3/4} + \sqrt{n} \right\rceil\right)$th smallest element in the sorted set R.
5. By comparing every element in S to d and u, compute the set $C = \{x \in S : d \leq x \leq u\}$ and the numbers $\ell_d = |\{x \in S : x < d\}|$ and $\ell_u = |\{x \in S : x > u\}|$.
6. If $\ell_d > n/2$ or $\ell_u > n/2$ then FAIL.
7. If $|C| \leq 4n^{3/4}$ then sort the set C, otherwise FAIL.
8. Output the $(\lfloor n/2 \rfloor - \ell_d + 1)$th element in the sorted order of C.

Why Use the Randomized Medians Algorithm?

- Deterministic Naive Medians Algorithm is an $O(nlogn)$ algorithm.
- Deterministic Complex Medians Algorithm is an $O(n)$ algorithm.
- The Monte Carlo algorithm, randomized medians, repeated enough times as a Las Vegas algorithm still possess the time complexity of $O(1)$.

Pretty useful for speeding up the algorithm right now!

My Attempt at using the Randomized K-Medians

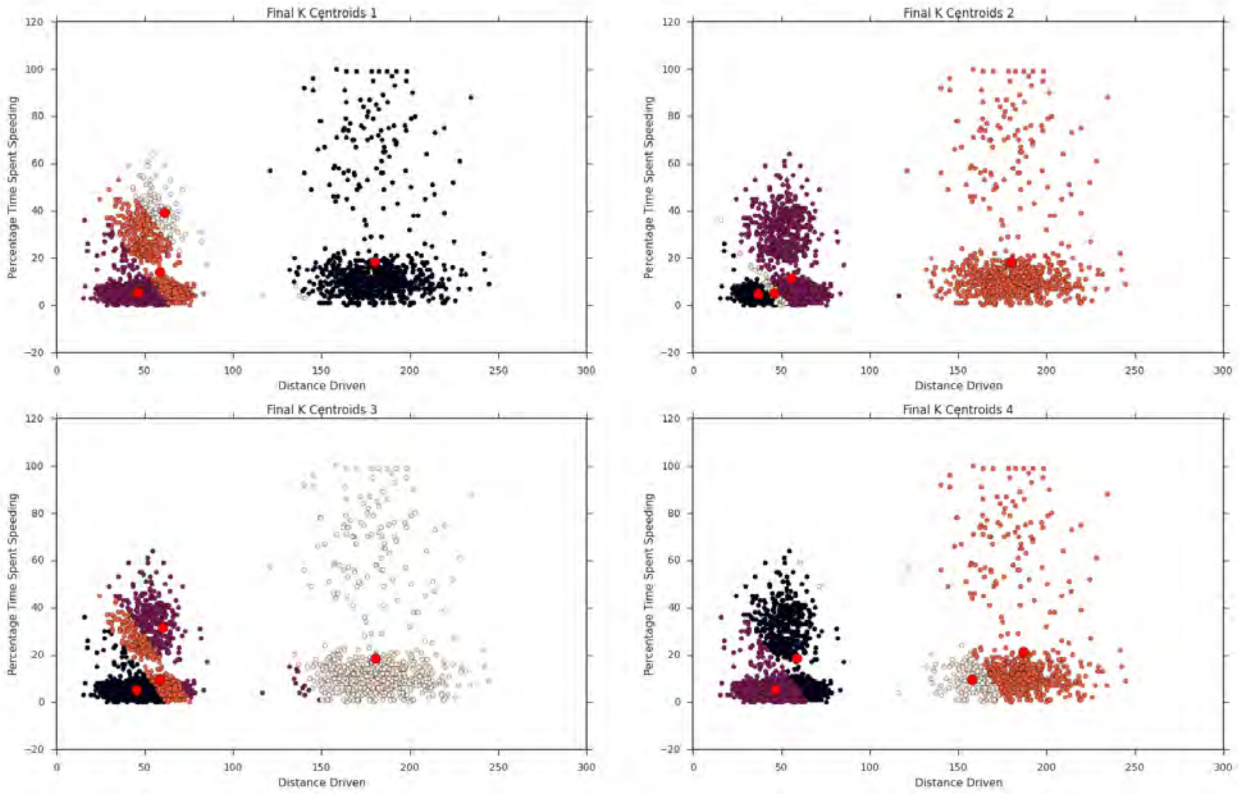
Listing 1: Deterministic Call

```
def move_centroids(points, closest, centroids):
    return np.array([points[closest == k]
                         .median(axis=0)
      for k in range(centroids.shape[0])])
```

Listing 2: Randomized Call

```
def move_centroids(points, closest, centroids):
    return np.array([points[closest == k]
                         .rand_median(0)
      for k in range(centroids.shape[0])])
```

My Attempt at using the Randomized K-Medians



Let us Improve Selection of the Centroids

Changing the way the Centroids are initialized

Listing 3: Shuffle Call

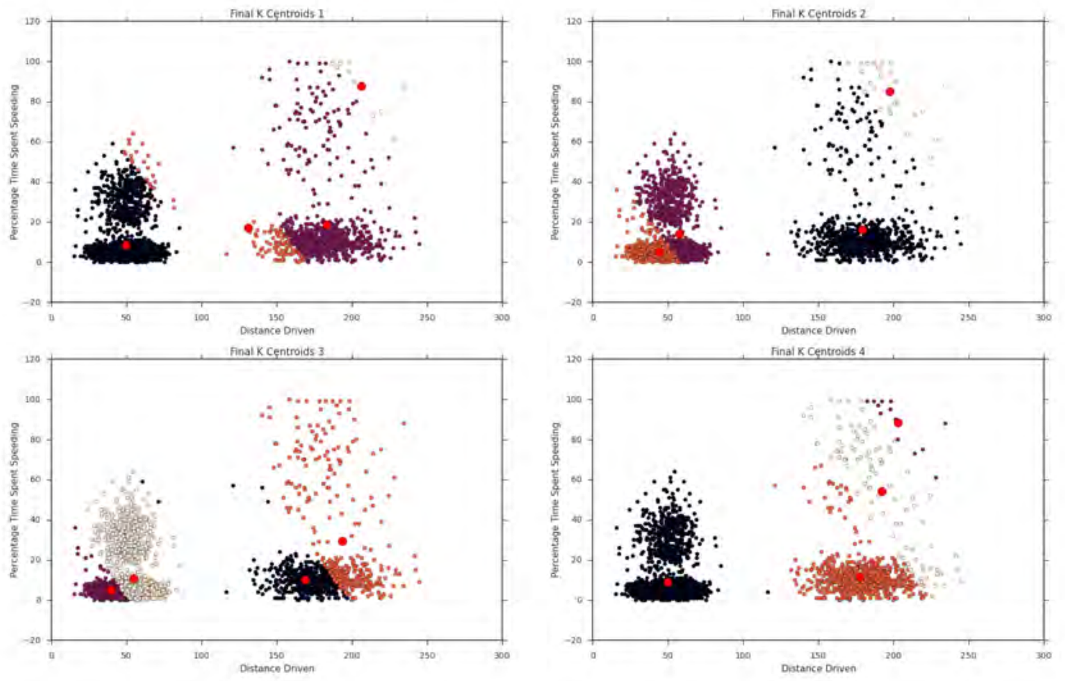
```
def initialize_centroids(points, k):
    centroids = points.copy()
    np.random.shuffle(centroids)
    return centroids[:k]
```

Listing 4: Truly Randomized Call

```
def initialize_centroids(k):
            centroids = np.random.rand(k, 2)
                              * [300, 100]
    return centroids
```

Using Truly Random Initial Centroids

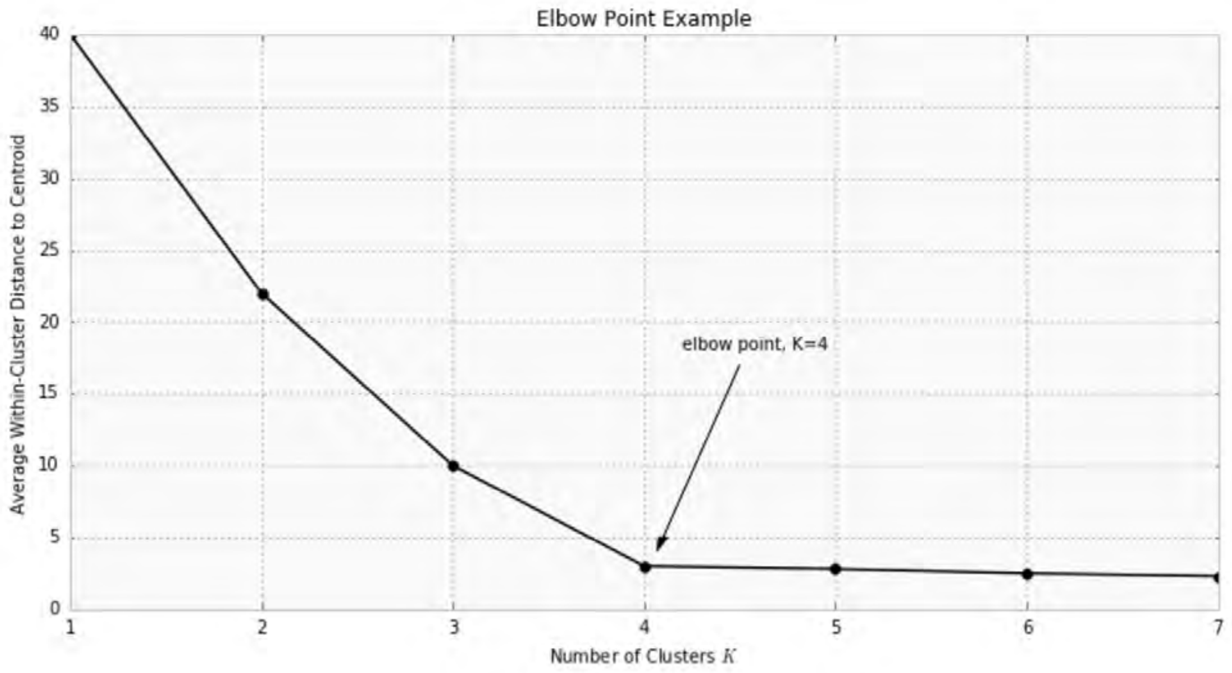
We can see a slight improvement here in terms of the clusters. However, we still haven't achieved significant improvements.



Efficiently discovering the K

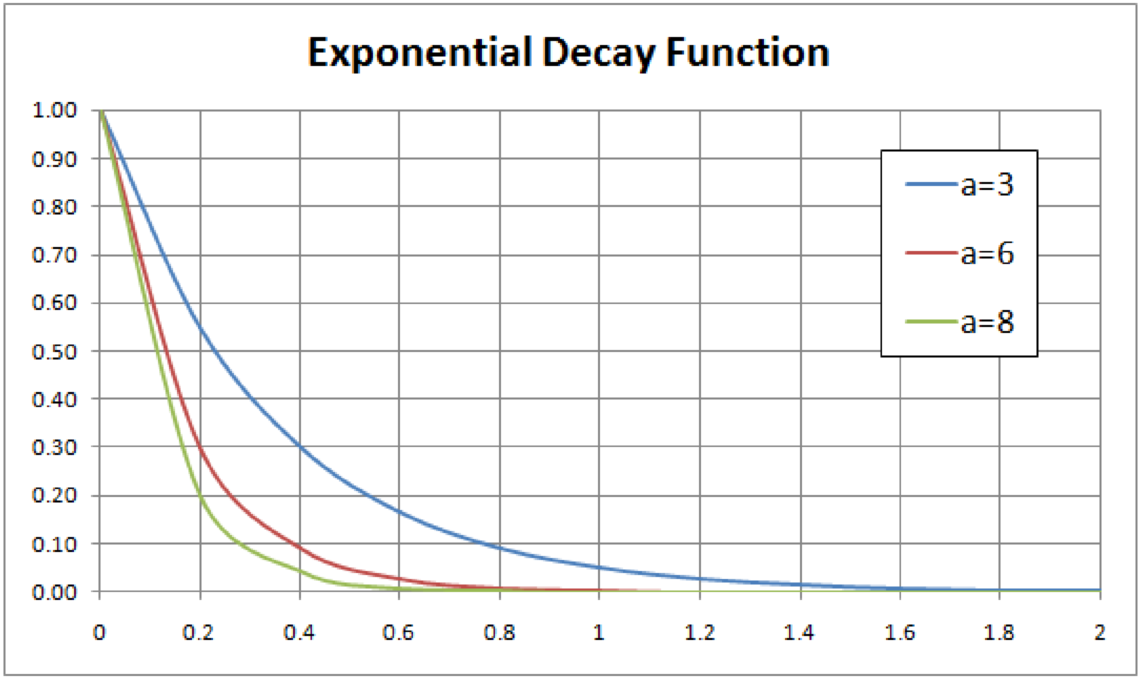
Computing the Value of K

If we calculate the ***Average Within Cluster Distance*** *to Centroid over all the different values of K, the most optimal value of K lies at an* ***elbow point****, which divides the clustering algorithm from being* ***under-fitted*** *and* ***over-fitted*** *respectively.*



Exponential Decay

In fact, this looks a lot like the ***Exponential Decay Function*** *which is given by the function R^{-x} (R is a constant given by the value a below):*



My O(1) Solution to Calculating the Optimal K

Within the Vanilla implementation of the a K Clustering Algorithm, we need to try out all the values of K before we find the optimal one at the elbow point (a cost of $O(n)$). My algorithm to combat that, follows:

- Randomly choose **five values** of K greater than each other in a successive row where:

$$0 < K_0 < |max(D_\text{i})|$$

$$K_\text{i-1} < K_\text{i} < |max(D_\text{i})|$$

  D_i *is the data set being clustered.*

- Figure out the Average Within-Cluster Distance (d_wC) to the Centroid for each of these K_i.

My O(1) Solution to Calculating the Optimal K Part 2

- Using this information, use **non-linear regression** to fit an exponential decay graph to the data, as shown in the `Python` code below:

```
params, cov = curve_fit(exp_decay,
              my_data[:,1], my_data[:,2],
              p0=guess)
```

- Once we have a curve, simply differentiating twice will let us know the point at which the change of the rate, of the change of the Average Within Cluster Distance, is maximum, thus giving us the elbow point.
- This is easily done in $O(i)$ time using a package such as `Sympy` which employs **symbolic differentiation**.

Evaluation and Analysis

Time Efficiency

Improvements of the Randomized K-Medians Algorithm (Theoretical)

- Time Efficiency of a K-Medians algorithm lies in the realm of $O(n^3 log(n))$.
- This is because the actual process of selecting a median is naively $nlog(n)$, while the process of finding the closest centroid is $O(n)$, and we need to search through n K's to find the best fit.
- Even the best deterministic algorithm for finding the medians is $O(n)$ and therefore the algorithm will be bounded by $O(n^3)$.
- My algorithm works at $O(n)$ time efficiency, since both the optimization of K and the search for the median are bounded by $O(1)$. We cannot avoid the process of finding the closest centroid, which will always be $O(n)$.

Improvements of the Randomized K-Medians Algorithm (Experimental)

Table 1: Final Time Results

ScipyKit.py	K_Medians	K_Means	Rand_K_Means	ELKI	FORTRAN
0.164273	0.189661	0.221699	0.153216	1.324781	0.835889

Correctness

How do we Evaluate? [3]

An **NLP Group** at **Stanford** has stated two different ways of evaluating the correctness of Supervised/Unsupervised Clustering Algorithms.

- **Purity**:

  $$Purity(\Omega, C) = \frac{1}{N}\Sigma_{\mathrm{k}} max_{\mathrm{j}}(|\Omega_{\mathrm{k}} \cap C_{\mathrm{j}}|)$$

  Purity will maximize as the K approaches $|D|$. Ω is the set of Clusters while C is the set of Classes.
- **Normalized Info Scores**:

  $$NMI(\Omega, C) = \frac{I(\Omega; C)}{[H(\Omega) + H(C)]/2}$$

  $I(\Omega; C)$ is the **mutual information** between Ω and C.
  We need a better evaluation parameter that can look at the tradeoff between the quality of clustering and the number of clusters.

The Code for Calculating Purity

Listing 5: The Code for Calculating Purity

```
def purity_score(y_true, y_pred):
    y_labeled_voted = np.zeros(y_true.shape)
    labels = np.unique(y_true)
    bins = np.concatenate((labels,
        [np.max(labels)+1]), axis=0)

    for cluster in np.unique(y_pred):
        hist, _ = np.histogram(
                y_true[y_pred==cluster], bins=bins)
        winner = np.argmax(hist)
        y_labeled_voted[y_pred==cluster] = winner

    return accuracy_score(y_true, y_labeled_voted)
```

Purity Results

Table 2: Final Purity Results

ScipyKit.py	**K_Medians**	**K_Means**	**Rand_K_Means**	**ELKI**	**FORTRAN**
0.35	0.20	0.20	0.10	0.40	0.40

The Normalized Mutual Info Scores

```
sklearn.metrics.normalized_mutual_info_score(
                    labels_true, labels_pred)
```

Table 3: Final NMI Results

ScipyKit.py	**K_Medians**	**K_Means**	**Rand_K_Means**	**ELKI**	**FORTRAN**
0.78	0.40	0.20	0.30	0.73	0.81

CONCLUSION

Now that we have implemented and analysed the K-Medians algorithms, we can make some conclusions about it.

- As the final results have shown, a randomized K-Medians does display a speedup, but not as much as anticipated by the shift from an $O(n^3)$ to an $O(n)$ algorithm.
- The `ScipyKit` is, in fact almost as fast as the randomized K-Medians algorithm and offers a much more accurate result in terms of the **purity** and **NMI**.
- As stated in [5] the reason for the lack of accuracy in K-Medians may be due to the fact that outliers can heavily affect the Manhattan Distance (since K-Medians only minimizes absolute deviations). We can combat this, by using a filter on the data set (future works).
- This gives us insight as to why the algorithm K-Means is far more preferred than the K-Medians algorithm. However, without further work, no definitive conclusion can be drawn.

Github Link to My Project

Please click on the OctoCat to go to my Github Project!

Future Works

- Need to improve the initialization of the Centroids [4] (An Arithmetic-Based Deterministic Centroid Initialization Method for the k-Means Clustering Algorithm).
- Do further testing to derive more conclusive results. This may require using more comprehensive data sets.
- Increase the number of dimensions to the feature vectors, to increase the robustness of my algorithm. Use data sets such as [6].

Does anyone have any Questions? THANK YOU FOR LISTENING!

References

[1] Trevino, A. (2017). Introduction to K-means Clustering. [online] Datascience.com. Available at: https://www.datascience.com/blog/k-means-clustering [Accessed 26 Nov. 2017].
[2] Bottou, L. and Bengio, Y., 1995. Convergence properties of the k-means algorithms. In Advances in neural information processing systems (pp. 585-592).
[3] https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html. (2017).
[Blog].
[4] Mayo, M.M., 2016. An Arithmetic-Based Deterministic Centroid Initialization Method for the k-Means Clustering Algorithm.
[5] k-median?, k. (2017). k-means vs k-median?. [online] Stats.stackexchange.com. Available at: https://stats.stackexchange.com/questions/109547/k-means-vs-k-median [Accessed 1 Dec.
2017].