Wes Dyer

CSE 494

Project C

April 21, 2004

Analysis of Clustering Algorithms in a Small Hypertext Search Engine

Introduction

Search Assistant is a small hypertext search engine that can use various search algorithms including: pure vector space, authorities/hubs, and PageRank. An analysis of the implementation of the vector space algorithm for this search engine is found in [1] and the other two algorithms are found in [2]. Recently, Search Assistant has been augmented by two document clustering algorithms. This paper analyzes the algorithms and compares their effectiveness.

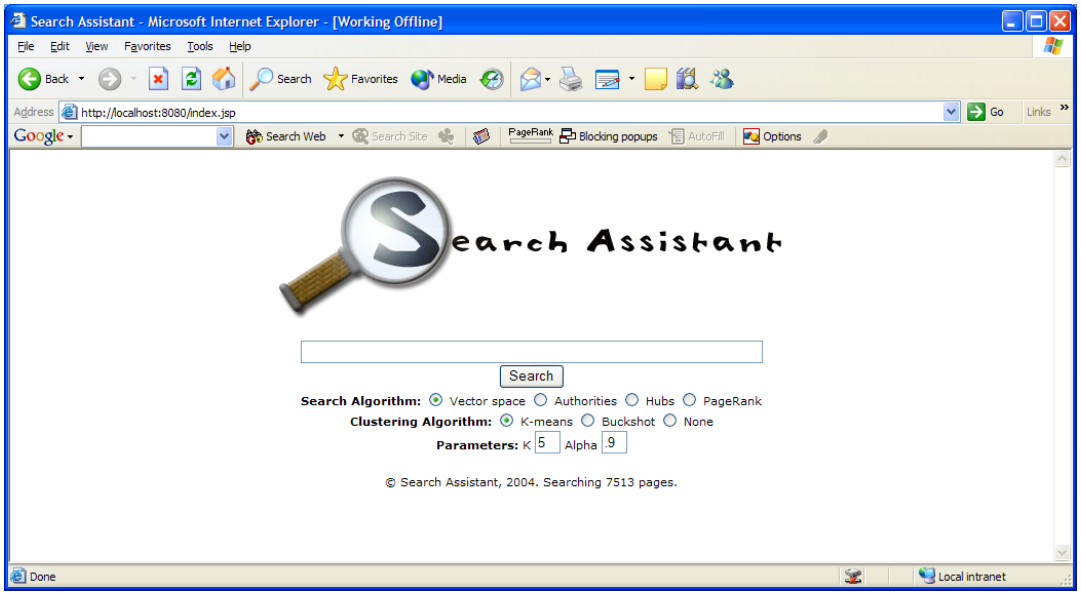

Figure 1: Homepage of Search Assistant

Figure 2: Screenshot of query results for a typical query in Search Assistant

Clustering Overview

The basic problem of data clustering is to assign each data point to one of *k* clusters such that the intra-cluster similarity of the data points is maximized and the inter-cluster similarity of the data points is minimized. Clustering text documents in a search engine assists users by putting a document in a group of related documents. Once a user locates a cluster of interest, the cluster should have better precision and recall than documents that are not clustered. Throughout this paper, cluster metrics will rely upon

the centroid of a cluster. This has been shown to provide reliable metrics [3]. Recall that the centroid *c* of a cluster is:

$$c = \frac{1}{|S|}\sum_{d \in S} d$$

Also note that the intra-cluster similarity S_a or cohesiveness of a cluster is:

$$S_a = \|c\|^2$$

Finally, the inter-cluster similarity S_e of a cluster is the average of the similarity of each centroid with all the other centroids.

K-Means Algorithm

The k-means algorithm is a basic clustering algorithm which assigns data to *k* clusters. It begins by randomly selecting *k* data points as the initial centroids and then assigning each data point to the closest centroids. This creates *k* initial clusters. The algorithm then computes the centroid of each cluster and reassigns the data points to the closest centroid. This process repeats until no data points change which cluster they are assigned to during a single iteration.

```
K-Means(data points, k) returns clusters {
    create k clusters
    for i = 1 to k {
        do {
            Randomly select a data point
        } while the data point has already been selected
        cluster[i].centroid = data point
    }
```

```
        AssignClusters(clusters, data points)
        do {
                ComputeCentroids(clusters)
                set change = AssignClusters(clusters, data points)
        } while change = true
        return clusters
}
AssignClusters(clusters,  data points) returns boolean {
        for each data point in data points {
                set newcluster = Nil
                for each cluster in clusters {
                        if newcluster = Nil
                                then newcluster = cluster
                        else if sim(cluster.centroid, data point) >
                        sim(newcluster.centroid, data point
                                 then newcluster = cluster
                }
                Assign data point to newcluster
        }
}
ComputeCentroids(clusters) {
        For each cluster in clusters {
                Set cluster.centroid = 0
```

```
                For each data point in cluster {
                        cluster.centroid += data point
                }
                cluster.centroid /= number_of(data points)
        }
}
```

Let *k, d, n,* and *l* be the number of clusters, the number of data points, the number of dimensions describing a data point, and the number of iterations respectively. Then the running time of *AssignClusters* is:

$$O(dkn)$$

Also, the running time of the *ComputeCentroids* algorithm is:

$$O(dn)$$

Finally, the time complexity of *k-means* is:

$$O(k + dkn + l(dkn + dn)) = O(ldkn)$$

Therefore, the algorithm is linear. One weakness of the basic k-means approach is that it is sensitive to the initial centroids that are chosen. Data points can easily be caught in local minima thereby providing a suboptimal clustering. Various modifications and extensions have been made to the k-means algorithm to choose better initial centroids. One such algorithm is described hereafter.

Buckshot Algorithm

The Buckshot algorithm is a hybrid algorithm that combines the k-means algorithm with hierarchical agglomerative clustering (HAC) techniques to choose better initial centroids than k-means. Therefore, the algorithm is exactly the same except for the

select of initial centroids. Thus, analysis will of this algorithm will concentrate on the HAC algorithm and use the results provided in the previous section.

HAC works by considering each data point as a separate cluster. The algorithm then compares each of these clusters with each other to find the greatest pair wise similarity between their centroids. The pair that has the greatest similarity is then combined and the new cluster replaces the original two in the list. This continues until there are only *k* clusters. Finally, the centroids of the clusters are used as the initial centroids for the k-means algorithm.

```
HAC(data points) returns clusters {
    Create list
    for i = 1 to SquareRoot(number_of(data points)) {
        create a cluster
        set the cluster's stamp to number_of(data points)
        add cluster to list
    }
    While list.size > k {
        Set cluster1 to Nil
        Set cluster2 to Nil
        For i = 1 to list.size {
            For j = i + 1 to list.size {
                if cluster1 is Nil
                    then cluster1 = list[i] and cluster2 = list[j]
                else if sim(list[i], list[j]) >
```

sim(cluster1, cluster2)

then *cluster1 = list[i]* and *cluster2 = list[j]*

}

Create *cluster* from *cluster1* and *cluster2*

Remove *cluster1* from *list*

Remove *cluster2* from *list*

Add *cluster* to *list*

}

Return *clusters* in *list*

}

}

Let *n* and *d* be the number of data points and the number of dimensions describing those data points respectively. The time complexity of HAC is:

$$O(\sqrt{n} + (\sqrt{n} - k)d\sqrt{n}^2) = O(dn^{\frac{3}{2}})$$

It should be noted that many implementations of HAC do not perform a complete pair wise comparison before combining clusters. Often, then randomly select one cluster to combine next. This would then have *O(dn)* time complexity. The pair wise comparison provides a more optimal HAC result and therefore better initial centroids so the increased time complexity is warranted in Search Assistant. It is important to note that *n* has an upper bound of 100 in Search Assistant so the time difference with such a small *n* is negligible.

Finally, the time complexity of the buckshot algorithm can be determined.

$$O(dn^{\frac{3}{2}} + ldkn)$$

Comparison of Algorithms using Similarity Measures

To evaluate the relative effectiveness of the basic k-means algorithm and the buck shot algorithm and the effect of varying *k*, a program was designed that ran five queries using both algorithms, with *k* values that ranged between 3 and 20, and with 20 trials. The average intra-cluster similarity and the average inter-cluster similarity for each configuration were recorded (see Appendix A).

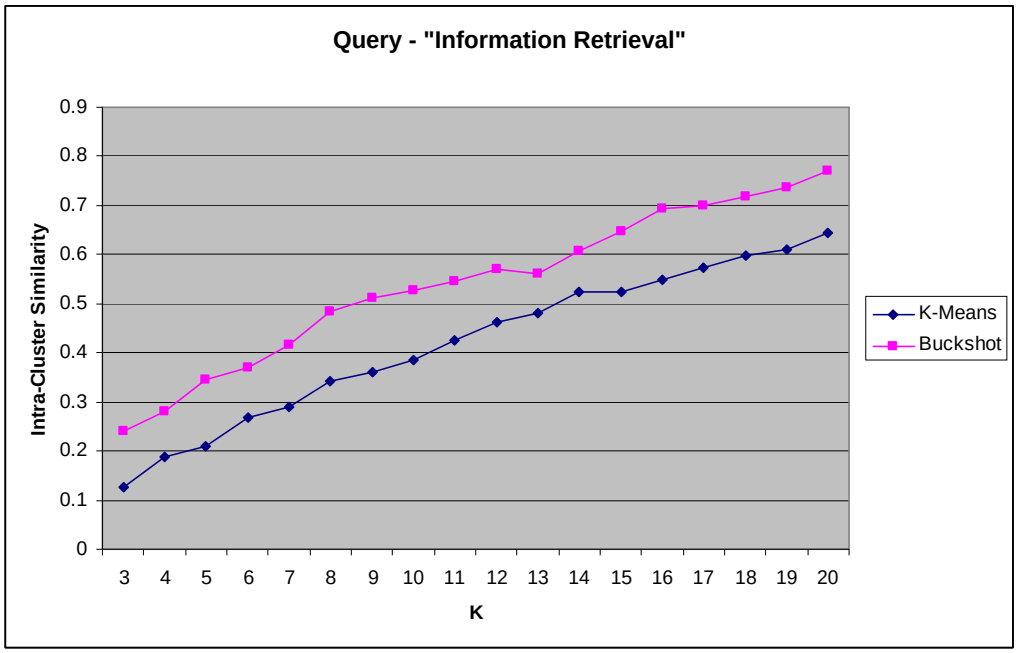

Figure 3: Intra-cluster similarity metrics using both algorithms for "information retrieval" query

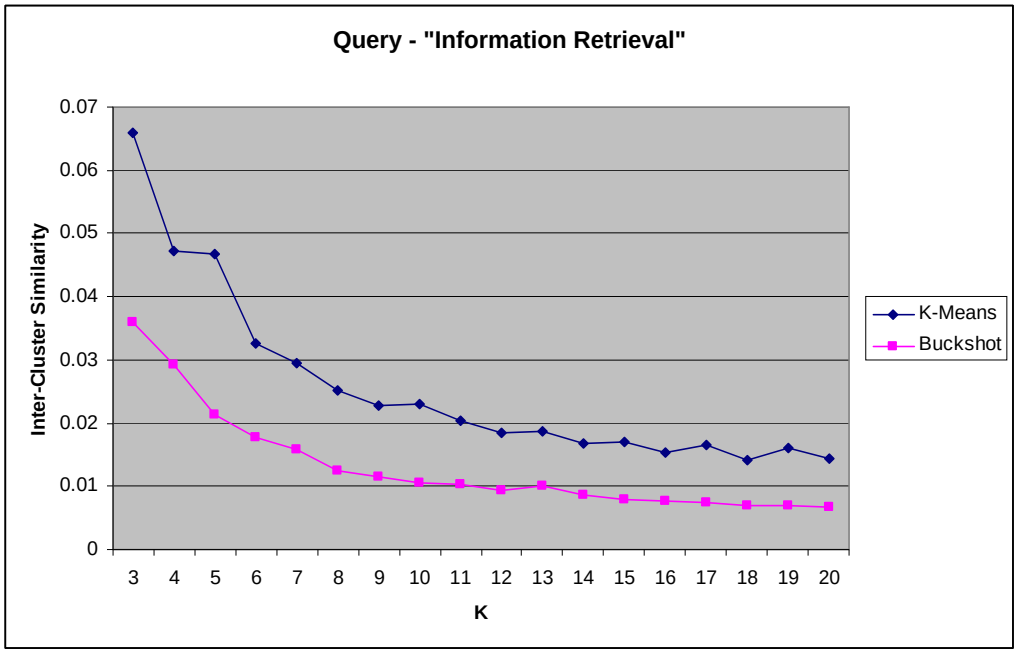

Figure 4: Inter-cluster similarity metrics using both algorithms for "information retrieval" query

Figures 3 and 4 show two of the graphs that were generated for the data set. Figure 3 shows that the buckshot algorithm had a consistently higher intra-cluster similarity than the basic k-means algorithm. Figure 3 shows that the buckshot algorithm had a consistently lower intra-cluster similarity. In each figure, the trend of both the buckshot algorithm and the basic k-means algorithm seems to be the same with only a constant differentiating the two. Note also that as *k* increases, the intra-cluster similarity will increase and the inter-cluster similarity will decrease. This is because as the number of clusters increases, the average number of document in a cluster will decrease. As the average number of documents in a cluster approaches 1, the intra-cluster similarity will approach 1 and the inter-cluster similarity will approach the inter-document similarity. Therefore, the rate of change and the offset determine to what degree a clustering algorithm is effective.

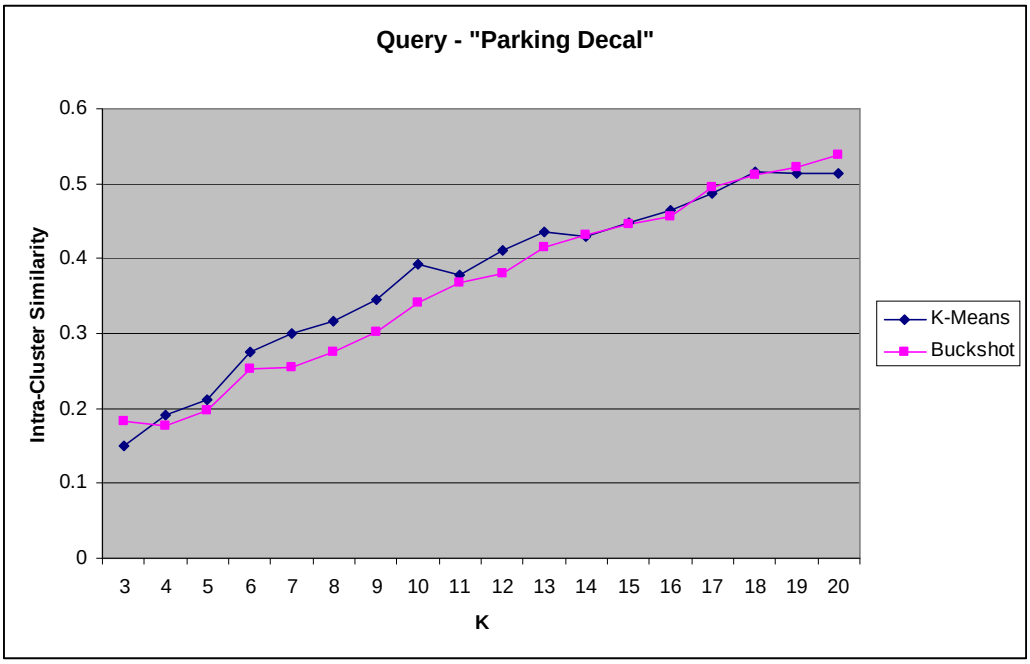

Figure 5: Intra-cluster similarity using both algorithms for "parking decal" query

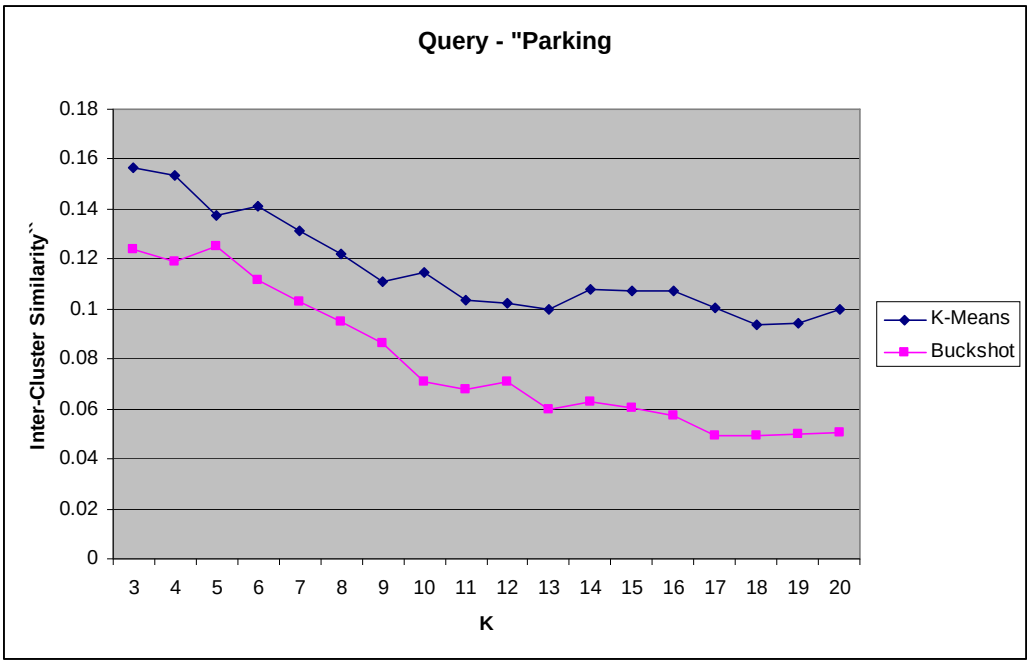

Figure 6: Inter-cluster similarity using both algorithms for "parking decal" query

Figures 5 and 6 show another set of graphs for a different query that has the same general results. In fact, each query had essentially the same results. The only interesting difference appears in figure 5. Figure 5 shows that although the buckshot algorithm

generally performs better than the basic k-means algorithm the difference between them is small. Perhaps, this is due to the fact that figure 3 represented a query with a small number of relevant documents and figure 5 represents a query with a larger number of documents. The graphs for the remaining three queries seemed to correlate with this hypothesis.

After analyzing the individual graphs, averages were computed for each algorithm and graphs were generated for the averages.

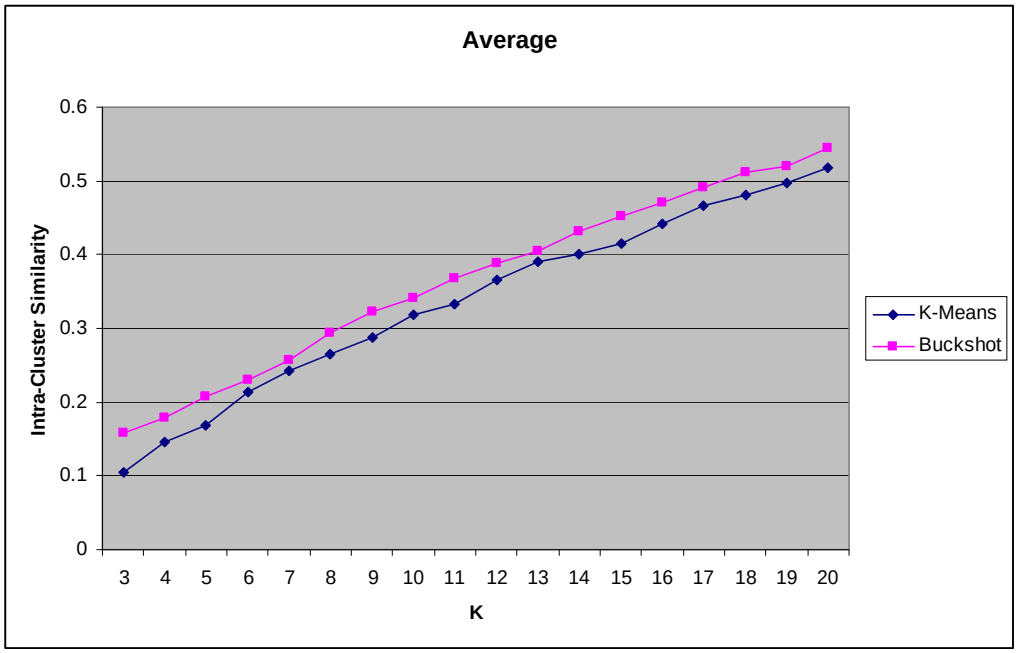

Figure 7: The average intra-cluster similarity for both algorithms

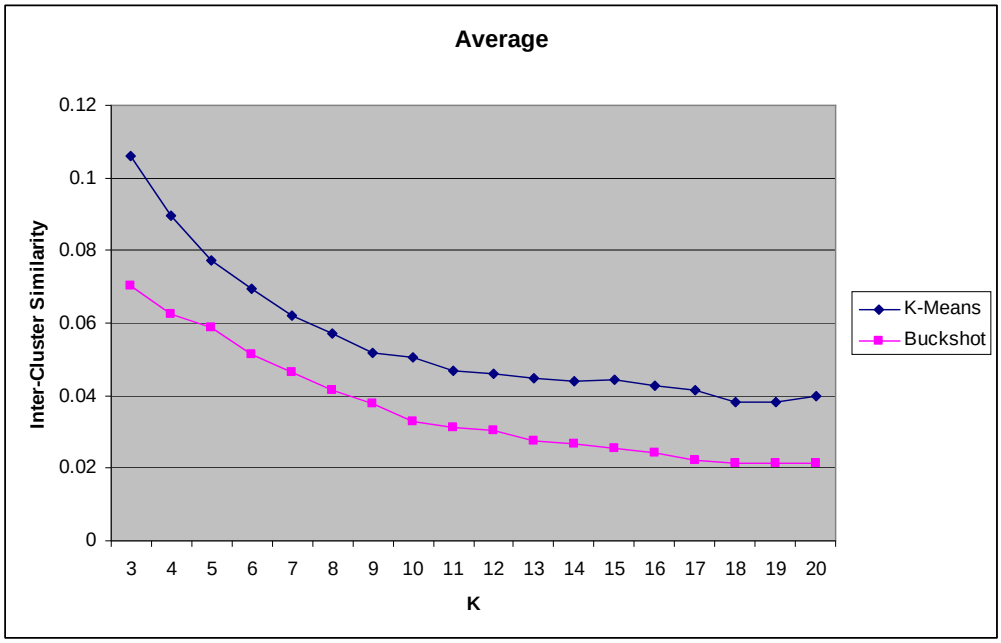

Figure 8: The average inter-cluster similarity for both algorithms

Figures 7 and 8 show the graphs of the average intra-cluster and inter-cluster similarities. These figures also demonstrate that the buckshot algorithm had a higher intra-cluster similarity and a lower inter-cluster similarity for all values of *k*.

Analysis of Algorithms using Similarity Measures

After the data and the averages were compared then regression analysis was used to find functions describing the observed behavior. It was found that the intra-cluster similarity and inter-cluster similarity of the k-means algorithm correspond to the following functions:

$$S_{ak}(k) = -.159 + .151\sqrt{k}$$

$$S_{ek}(k) = .0264 + \frac{.246}{k}$$

The intra-cluster similarity and inter-cluster similarity of the buckshot algorithm can be expressed as:

$$S_{ab}(k) = -.117 + .147\sqrt{k}$$

$$S_{eb}(k) = -.0138 + \frac{.153}{\sqrt{k}}$$

The ordered and fitted residual plots as well as the normal probability plot of the residuals all showed that the equations were correct. Also, the R^2-adj was at least .98 for each of the equations. So it can be concluded that these do indeed capture the similarities as a function of *k*.

Figures 9 through 12 show the actual versus predicted similarities for both algorithms. Note that the predicted values describe the observed results well.

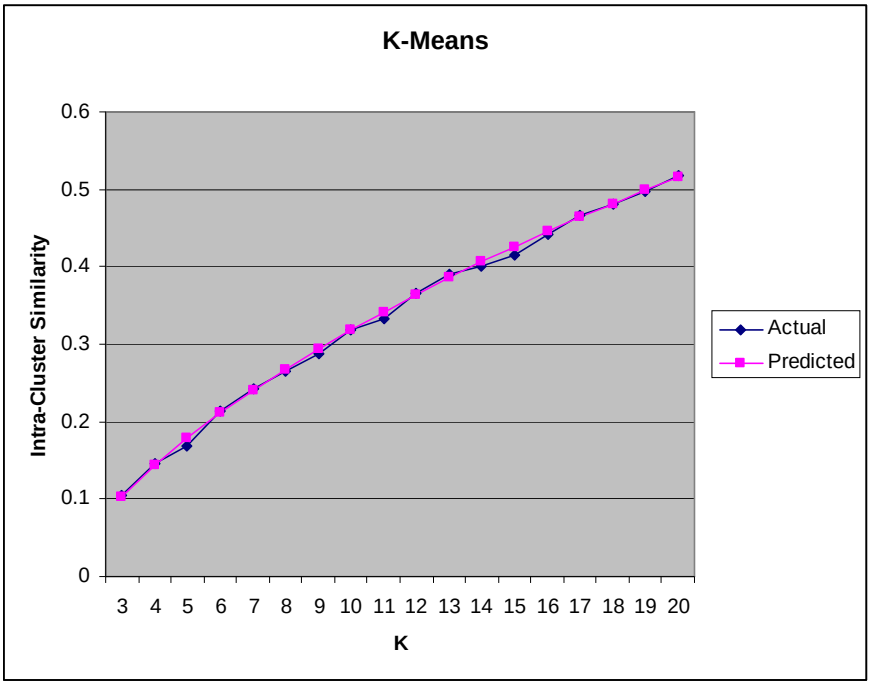

Figure 9: Actual versus predicted values for average intra-cluster similarity using the k-means algorithm

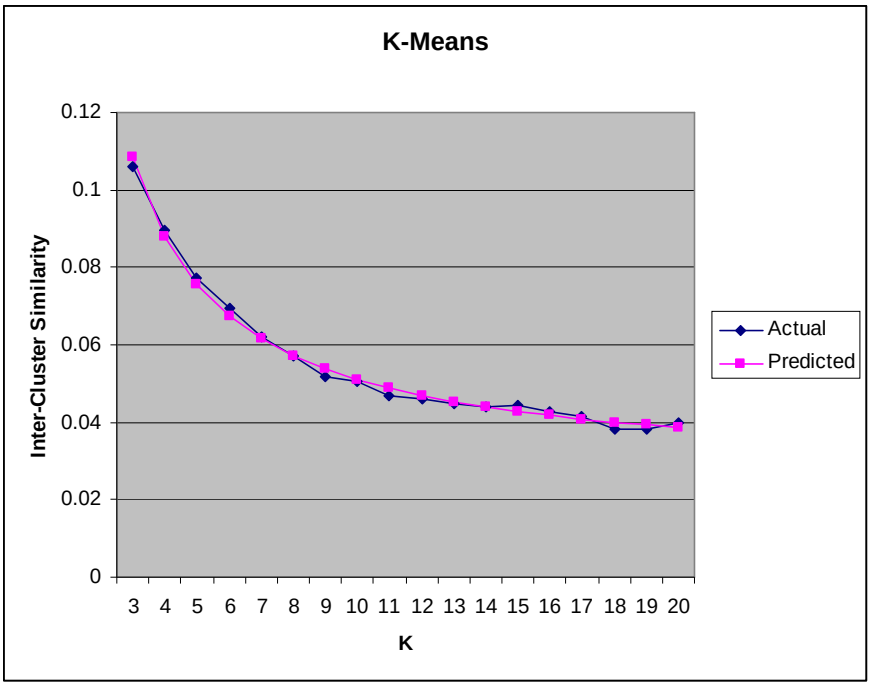

Figure 10: Actual versus predicted values for average inter-cluster similarity using the basic k-means algorithm

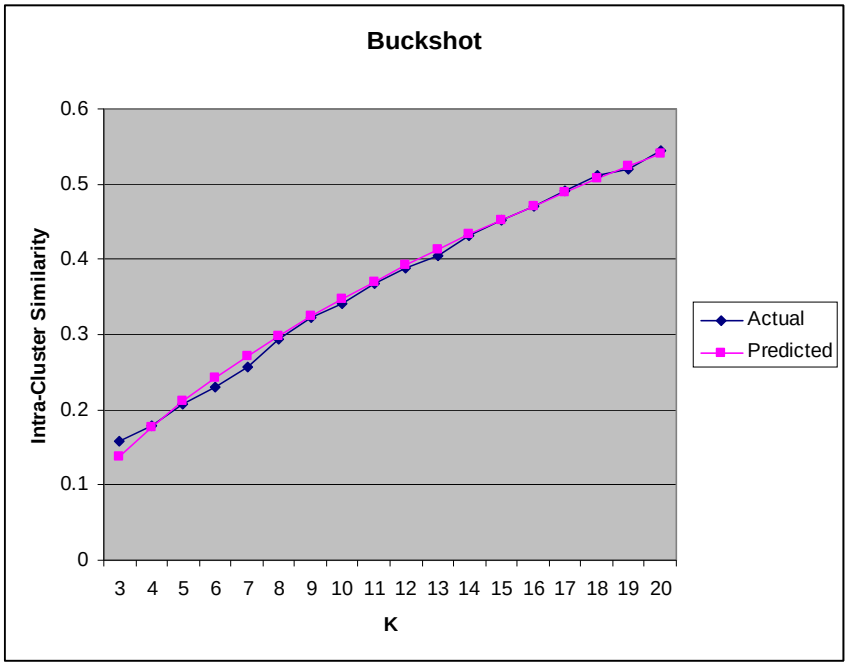

Figure 11: Actual versus predicted values for average intra-cluster similarity using the buckshot algorithm

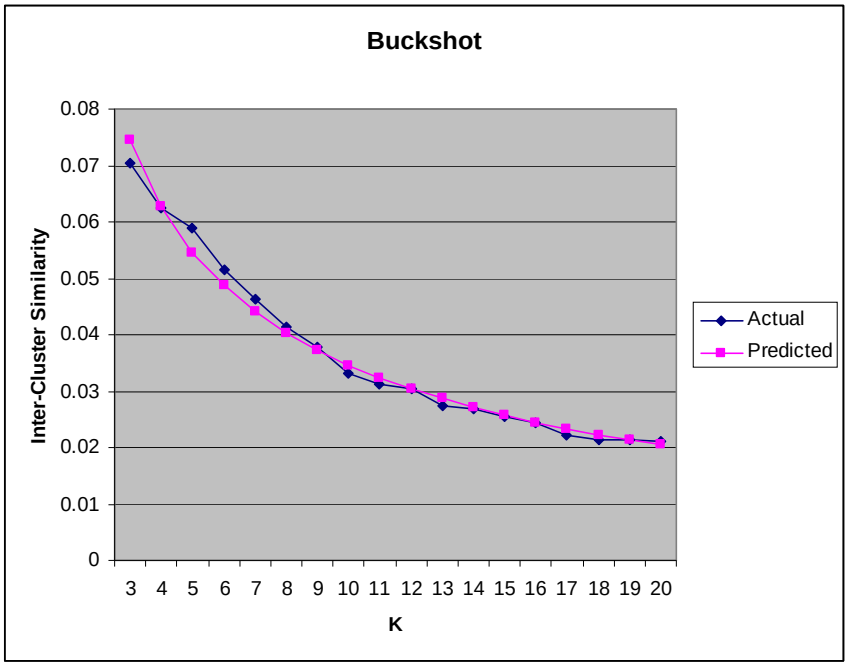

Figure 12: Actual versus predicted values for average inter-cluster similarity using the basic k-means algorithm

Finally, the equations were graphed together to compare the predicted difference between the two algorithms. This is shown in figures 13 and 14.

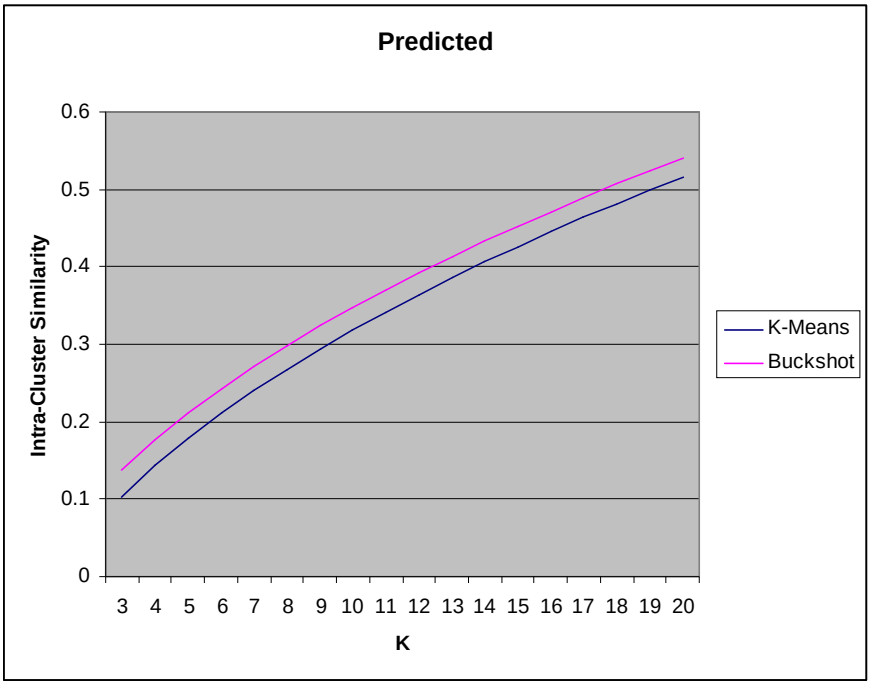

Figure 13: Predicted intra-cluster similarity for both algorithms

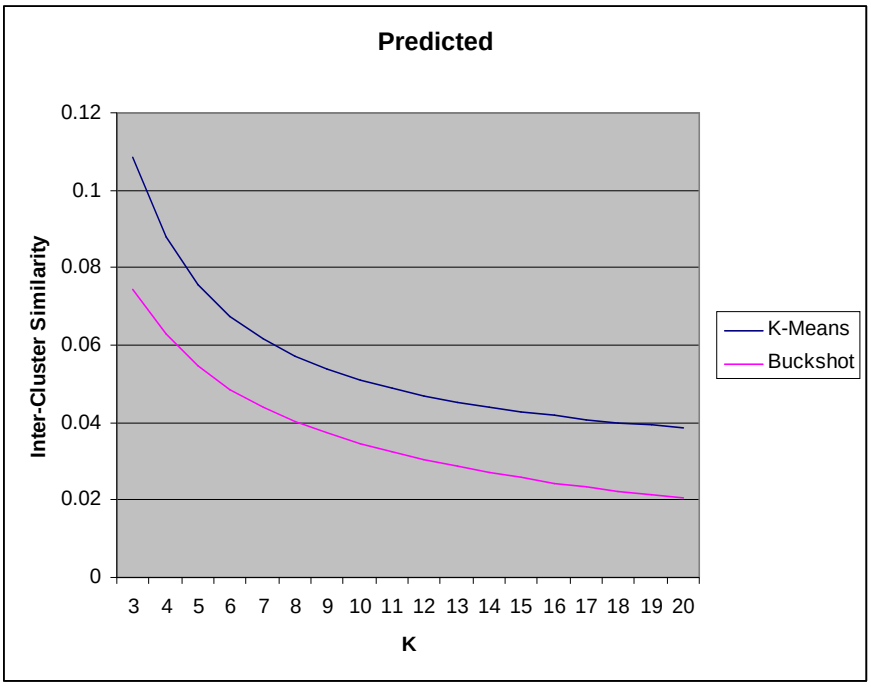

Figure 14: Predicted inter-cluster similarity for both algorithms

It is interesting that although the equations that describe the intra-cluster similarity for the two algorithms vary only by the coefficients, the equations for the inter-cluster similarity differ by the order of the polynomial that is used to express them. Also, by solving the equations to find when k-means algorithm is predicted to perform inferior to the buckshot algorithm, it is discovered that for all $k < 110$ the buckshot algorithm will outperform the basic k-means algorithm in both measures. Since Search Assistant only clusters the top 100 documents then it can be concluded that the buckshot algorithm will have a higher intra-cluster similarity and a lower inter-cluster similarity for the average query.

Another interesting result is that both of the intra-cluster similarity functions are strictly increasing for $k > 0$ and $k \leq 100$ and both of the inter-cluster similarity functions are strictly decreasing on the same interval. It might be tempting to conclude that as

large of a value of *k* as possible should be picked; however, this is not the case because the goal of clustering is to provide a manageably small number of clusters to aid users in finding relevant documents while avoiding irrelevant documents. Therefore, the smallest value of *k* should be picked that provides maximal intra-cluster similarity and minimum inter-cluster similarity. Note that there are diminishing returns as *k* increases. Therefore, a value of *k* could be picked that captures some percentage of the change in similarity over an interval.

First, the point at which 90% of the change over the interval from 3 to 100 using the S_{ek} equation is found.

$$
\begin{aligned}
& S_{ek}(k) = .9\left(S_{ek}(100) - S_{ek}(3)\right) + S_{ek}(3) \\
& \Rightarrow S_{ek}(k) = .9\left(.0264 + \frac{.246}{100} - .0264 - \frac{.246}{3}\right) + .0264 + \frac{.246}{3} \\
& \Rightarrow S_{ek}(k) = .0368 \\
& \Rightarrow .0264 + \frac{.246}{k} = .0368 \\
& \Rightarrow \frac{.246}{k} = .0104 \\
& \Rightarrow k = 23.622
\end{aligned}
$$

So, 24 clusters are required before 90% of the change is realized on the interval from 3 to 100 using the basic k-means algorithm. It can now be shown when the buckshot algorithm reaches the same similarity (not the same percentage of change).

$$
\begin{aligned}
& S_{eb}(k) = .9\left(S_{ek}(100) - S_{ek}(3)\right) + S_{ek}(k) \\
& \Rightarrow S_{eb}(k) = .0368 \\
& \Rightarrow -.0138 + \frac{.153}{\sqrt{k}} = .0368 \\
& \Rightarrow \frac{1}{\sqrt{k}} = .3307 \\
& \Rightarrow \sqrt{k} = 3.0237 \\
& \Rightarrow k = 9.14285
\end{aligned}
$$

Therefore, it only requires 10 clusters in the buckshot algorithm to reach the same level of inter-cluster similarity. A similar process will show that the k-means algorithm will reach 90% of the change for the intra-cluster similarity when *k* is 85. The buckshot algorithm requires that *k* is 84 for the same result. Note more constant rate of change for the intra-cluster similarity and the small difference between the algorithms when the change percentage is 90%. In general, for intra-cluster similarity it requires one less cluster for buckshot to achieve the same level of intra-cluster similarity. This is not true for inter-cluster similarity. In fact, the difference between the algorithms for inter-cluster similarity increases as *k* increases.

Comparison of Algorithms using Categories

Another way to gauge the effectiveness of a clustering algorithm is to determine whether the clusters that are produced correspond to natural categories or not. This measure certainly is more subjective than the other two measures that have been previously used. In order to validate that the clusters correspond to natural categories, a group of three users were selected and each was asked to name the various clusters that were generated. Each user rated the clusters on a scale from zero to five where five is the highest. The rating was based on how unique the category was, how much it corresponded to a nameable category, and how much the category made sense. The process was repeated for various values of *k* and using both algorithms. The results from the three users were averaged and recorded (see Appendix B) and the graph in figure 15 was produced.

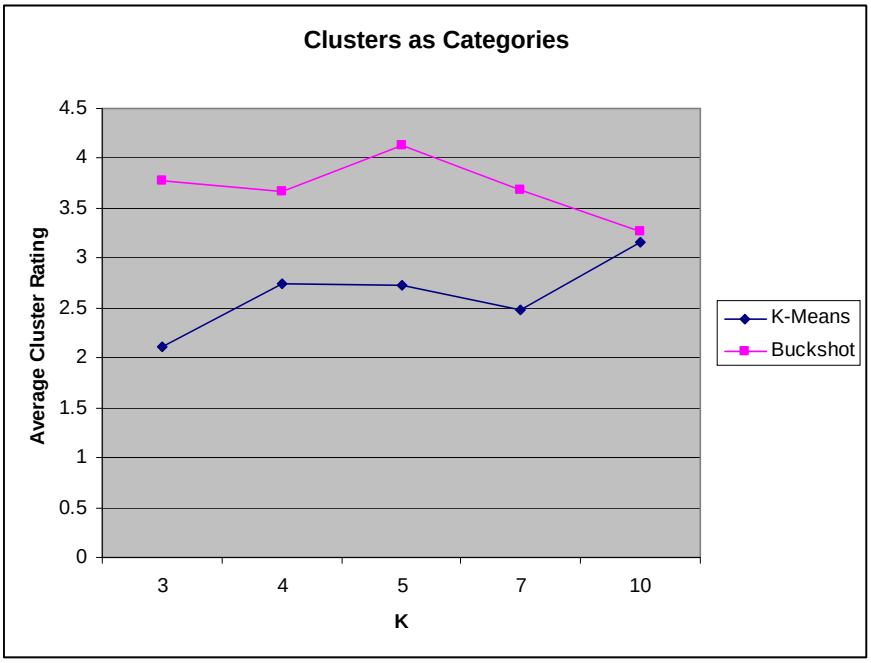

Figure 15: Average user rating for clusters of various sizes for both algorithms

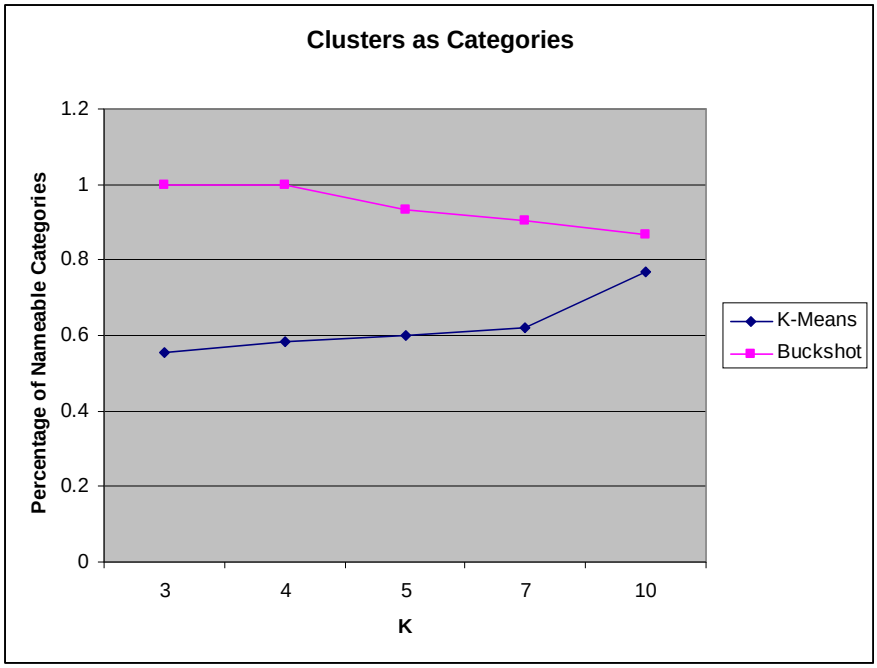

Figure 16: Percentage of clusters that users could name as distinct categories

Clearly, the buckshot algorithm outperformed the k-means algorithm at every value of *k*. However, it should be noted that it seems that the two algorithms' scores will converge as *k* increases beyond values of 10. Figure 16 depicts the percentage of clusters

for which the users could find distinct and descriptive names. Again, the buckshot algorithm is superior to the k-means algorithm and it also seems that the two percentages are converging. It is possible that this is due to the increasing number of clusters. As the number of clusters increases then it may be increasing difficult for users to differentiate the clusters from each other.

Conclusion

The user study and the similarity measures show that the buckshot algorithm performs better than the basic k-means algorithm for every value of *k*. Although initially glancing at search results for the two algorithms does not lead to this conclusion, the result is expected since the buckshot algorithm chooses better centroids and then follows the same process as the basic k-means algorithm. The experimental results show that it is an improvement on the basic k-means algorithm. Also, the analysis has shown that the clustering algorithms are sufficiently good to use in the search engine.

Works Cited

1. Dyer, Wes. *An Implementation of Vector Space Model with the Lucene API*. March 21, 2004.

2. Dyer, Wes. *Analysis of a Small Search Engine*. April 2004.

3. Michael Steinbach, George Karypis, and Vipin Kumar. *A Comparison of Document Clustering Techniques*.

Appendix A

Query	Algorithm	K	Intra-cluster	Inter-cluster
Information Retrieval	kmeans	3	0.126895049	0.065859313
Information Retrieval	kmeans	4	0.188018703	0.047250882
Information Retrieval	kmeans	5	0.208092477	0.046714421
Information Retrieval	kmeans	6	0.268327714	0.032501347
Information Retrieval	kmeans	7	0.290416691	0.029473463
Information Retrieval	kmeans	8	0.340676688	0.02507525
Information Retrieval	kmeans	9	0.361019258	0.022683083
Information Retrieval	kmeans	10	0.383804	0.022908427
Information Retrieval	kmeans	11	0.426080222	0.02029967
Information Retrieval	kmeans	12	0.461102866	0.01838545
Information Retrieval	kmeans	13	0.479643389	0.018587612
Information Retrieval	kmeans	14	0.522756746	0.016828432
Information Retrieval	kmeans	15	0.52266996	0.016981662
Information Retrieval	kmeans	16	0.548301763	0.015302295
Information Retrieval	kmeans	17	0.5740487	0.016449952
Information Retrieval	kmeans	18	0.597815206	0.014181438
Information Retrieval	kmeans	19	0.609143397	0.016113071
Information Retrieval	kmeans	20	0.645521988	0.014406246
Information Retrieval	buckshot	3	0.239942593	0.035895404
Information Retrieval	buckshot	4	0.281567533	0.029295029
Information Retrieval	buckshot	5	0.346153677	0.021410849
Information Retrieval	buckshot	6	0.368830957	0.017729792
Information Retrieval	buckshot	7	0.4154496	0.015765645
Information Retrieval	buckshot	8	0.484065071	0.012436063
Information Retrieval	buckshot	9	0.511337486	0.011437154
Information Retrieval	buckshot	10	0.528025017	0.010646563
Information Retrieval	buckshot	11	0.545955405	0.010263832
Information Retrieval	buckshot	12	0.571089364	0.009349891
Information Retrieval	buckshot	13	0.562486391	0.00996499
Information Retrieval	buckshot	14	0.606422276	0.008734009
Information Retrieval	buckshot	15	0.647630794	0.00790283
Information Retrieval	buckshot	16	0.694130073	0.007613983
Information Retrieval	buckshot	17	0.699088637	0.007343093
Information Retrieval	buckshot	18	0.718752166	0.00704089
Information Retrieval	buckshot	19	0.737338601	0.006890936
Information Retrieval	buckshot	20	0.769250972	0.00665565
Computer Science	kmeans	3	0.104184173	0.080055986
Computer Science	kmeans	4	0.155244874	0.07478292
Computer Science	kmeans	5	0.176244086	0.058595689
Computer Science	kmeans	6	0.227181646	0.053137667
Computer Science	kmeans	7	0.243053945	0.046801485
Computer Science	kmeans	8	0.277944063	0.044764426
Computer Science	kmeans	9	0.309409274	0.041765833
Computer Science	kmeans	10	0.335511146	0.037580013
Computer Science	kmeans	11	0.346629615	0.037912816

Computer Science	kmeans	12	0.382875063	0.037850483
Computer Science	kmeans	13	0.423088476	0.039061135
Computer Science	kmeans	14	0.434023179	0.032375321
Computer Science	kmeans	15	0.427486139	0.037831822
Computer Science	kmeans	16	0.465960506	0.034667074
Computer Science	kmeans	17	0.494469656	0.03397981
Computer Science	kmeans	18	0.508450321	0.030338929
Computer Science	kmeans	19	0.521334476	0.030113411
Computer Science	kmeans	20	0.547553059	0.035746403
Computer Science	buckshot	3	0.148017833	0.047163521
Computer Science	buckshot	4	0.164962623	0.05003088
Computer Science	buckshot	5	0.195625892	0.039659101
Computer Science	buckshot	6	0.19393985	0.035904985
Computer Science	buckshot	7	0.241907854	0.031100877
Computer Science	buckshot	8	0.277452323	0.030625807
Computer Science	buckshot	9	0.298335524	0.027920399
Computer Science	buckshot	10	0.296627696	0.028020856
Computer Science	buckshot	11	0.336991865	0.024522771
Computer Science	buckshot	12	0.375339183	0.022868826
Computer Science	buckshot	13	0.396238074	0.021468583
Computer Science	buckshot	14	0.430707369	0.019961226
Computer Science	buckshot	15	0.43616306	0.018828166
Computer Science	buckshot	16	0.436385853	0.018802539
Computer Science	buckshot	17	0.45764885	0.01868742
Computer Science	buckshot	18	0.486111437	0.016400311
Computer Science	buckshot	19	0.484334919	0.016669157
Computer Science	buckshot	20	0.510121881	0.01618469
Multimedia Database	kmeans	3	0.063365742	0.106159129
Multimedia Database	kmeans	4	0.090578843	0.079377791
Multimedia Database	kmeans	5	0.12146068	0.062794753
Multimedia Database	kmeans	6	0.137336404	0.055861023
Multimedia Database	kmeans	7	0.176699231	0.047756034
Multimedia Database	kmeans	8	0.183370234	0.042952012
Multimedia Database	kmeans	9	0.201051369	0.036385103
Multimedia Database	kmeans	10	0.215093703	0.03595474
Multimedia Database	kmeans	11	0.234390607	0.033837044
Multimedia Database	kmeans	12	0.273948161	0.03075323
Multimedia Database	kmeans	13	0.280573892	0.029791184
Multimedia Database	kmeans	14	0.278150363	0.029341718
Multimedia Database	kmeans	15	0.323254647	0.024899285
Multimedia Database	kmeans	16	0.32404667	0.025819797
Multimedia Database	kmeans	17	0.353356311	0.023677651
Multimedia Database	kmeans	18	0.371845234	0.023166923
Multimedia Database	kmeans	19	0.386112594	0.021295114
Multimedia Database	kmeans	20	0.400086562	0.020967098
Multimedia Database	buckshot	3	0.103166357	0.058756045
Multimedia Database	buckshot	4	0.141584193	0.044444886
Multimedia Database	buckshot	5	0.146180037	0.046816551

Multimedia Database	buckshot	6	0.178122635	0.036469611
Multimedia Database	buckshot	7	0.180647839	0.0360094
Multimedia Database	buckshot	8	0.225780343	0.028816655
Multimedia Database	buckshot	9	0.249318774	0.026653139
Multimedia Database	buckshot	10	0.262048419	0.02403192
Multimedia Database	buckshot	11	0.282280656	0.023087414
Multimedia Database	buckshot	12	0.286650572	0.021269459
Multimedia Database	buckshot	13	0.313962132	0.020141669
Multimedia Database	buckshot	14	0.324798481	0.018635096
Multimedia Database	buckshot	15	0.344719252	0.017970208
Multimedia Database	buckshot	16	0.355139533	0.016746197
Multimedia Database	buckshot	17	0.379336928	0.015758784
Multimedia Database	buckshot	18	0.391163011	0.015797745
Multimedia Database	buckshot	19	0.41791312	0.014116583
Multimedia Database	buckshot	20	0.431750259	0.013825877
Software Engineering	kmeans	3	0.082697982	0.122103826
Software Engineering	kmeans	4	0.104449688	0.091838249
Software Engineering	kmeans	5	0.127381453	0.08109389
Software Engineering	kmeans	6	0.162191439	0.064070565
Software Engineering	kmeans	7	0.198561414	0.054363453
Software Engineering	kmeans	8	0.205975853	0.051851955
Software Engineering	kmeans	9	0.221853122	0.04788921
Software Engineering	kmeans	10	0.263162671	0.040926231
Software Engineering	kmeans	11	0.27950921	0.039605424
Software Engineering	kmeans	12	0.295003235	0.040144066
Software Engineering	kmeans	13	0.334557718	0.037233524
Software Engineering	kmeans	14	0.339770586	0.03395572
Software Engineering	kmeans	15	0.352013088	0.036030108
Software Engineering	kmeans	16	0.410635903	0.031496925
Software Engineering	kmeans	17	0.421644056	0.0323243
Software Engineering	kmeans	18	0.414989775	0.029096202
Software Engineering	kmeans	19	0.452049146	0.03004354
Software Engineering	kmeans	20	0.487314326	0.027533927
Software Engineering	buckshot	3	0.118065011	0.086082548
Software Engineering	buckshot	4	0.134651003	0.069770841
Software Engineering	buckshot	5	0.149196013	0.061758706
Software Engineering	buckshot	6	0.15403538	0.055626534
Software Engineering	buckshot	7	0.197160571	0.046312307
Software Engineering	buckshot	8	0.209064016	0.04026252
Software Engineering	buckshot	9	0.25191877	0.036245597
Software Engineering	buckshot	10	0.275324855	0.031825641
Software Engineering	buckshot	11	0.303798835	0.029627109
Software Engineering	buckshot	12	0.329868362	0.027663581
Software Engineering	buckshot	13	0.339549947	0.025643422
Software Engineering	buckshot	14	0.36989783	0.024056276
Software Engineering	buckshot	15	0.389654744	0.022586128
Software Engineering	buckshot	16	0.407023227	0.021811004
Software Engineering	buckshot	17	0.422449166	0.020130702

Software Engineering	buckshot	18	0.451003114	0.018823251
Software Engineering	buckshot	19	0.439614085	0.019353625
Software Engineering	buckshot	20	0.468331153	0.018449561
Parking Decal	kmeans	3	0.150501206	0.156590524
Parking Decal	kmeans	4	0.191126599	0.153675355
Parking Decal	kmeans	5	0.211650958	0.13752367
Parking Decal	kmeans	6	0.275798823	0.141292963
Parking Decal	kmeans	7	0.300637214	0.13137641
Parking Decal	kmeans	8	0.315630355	0.121798382
Parking Decal	kmeans	9	0.344289153	0.110910809
Parking Decal	kmeans	10	0.39188546	0.114357903
Parking Decal	kmeans	11	0.37710006	0.103420407
Parking Decal	kmeans	12	0.410871123	0.102625159
Parking Decal	kmeans	13	0.434821192	0.099595029
Parking Decal	kmeans	14	0.429744606	0.108125148
Parking Decal	kmeans	15	0.448002737	0.106956998
Parking Decal	kmeans	16	0.463360118	0.107198019
Parking Decal	kmeans	17	0.486493203	0.100451986
Parking Decal	kmeans	18	0.515285527	0.093424894
Parking Decal	kmeans	19	0.513005571	0.094313568
Parking Decal	kmeans	20	0.513381745	0.100158425
Parking Decal	buckshot	3	0.182999385	0.123920206
Parking Decal	buckshot	4	0.175878455	0.119213774
Parking Decal	buckshot	5	0.198186645	0.124836621
Parking Decal	buckshot	6	0.252003593	0.111457138
Parking Decal	buckshot	7	0.254079282	0.102762842
Parking Decal	buckshot	8	0.274766987	0.094977761
Parking Decal	buckshot	9	0.302742889	0.086533944
Parking Decal	buckshot	10	0.341490007	0.070597482
Parking Decal	buckshot	11	0.36872349	0.068039585
Parking Decal	buckshot	12	0.379253646	0.07071564
Parking Decal	buckshot	13	0.414563051	0.059955854
Parking Decal	buckshot	14	0.430690043	0.062866675
Parking Decal	buckshot	15	0.444896205	0.060275807
Parking Decal	buckshot	16	0.456404584	0.057234862
Parking Decal	buckshot	17	0.495643096	0.049171587
Parking Decal	buckshot	18	0.512201342	0.049159548
Parking Decal	buckshot	19	0.522390358	0.049779434
Parking Decal	buckshot	20	0.539075822	0.050746038
Average	kmeans	3	0.105528831	0.106153756
Average	kmeans	4	0.145883741	0.089385039
Average	kmeans	5	0.168965931	0.077344485
Average	kmeans	6	0.214167205	0.069372713
Average	kmeans	7	0.241873699	0.061954169
Average	kmeans	8	0.264719439	0.057288405
Average	kmeans	9	0.287524435	0.051926808
Average	kmeans	10	0.317891396	0.050345463
Average	kmeans	11	0.332741943	0.047015072

Average	kmeans	12	0.36476009	0.045951678
Average	kmeans	13	0.390536933	0.044853697
Average	kmeans	14	0.400889096	0.044125268
Average	kmeans	15	0.414685314	0.044539975
Average	kmeans	16	0.442460992	0.042896822
Average	kmeans	17	0.466002385	0.04137674
Average	kmeans	18	0.481677213	0.038041677
Average	kmeans	19	0.496329037	0.038375741
Average	kmeans	20	0.518771536	0.03976242
Average	buckshot	3	0.158438236	0.070363545
Average	buckshot	4	0.179728761	0.062551082
Average	buckshot	5	0.207068453	0.058896365
Average	buckshot	6	0.229386483	0.051437612
Average	buckshot	7	0.257849029	0.046390214
Average	buckshot	8	0.294225748	0.041423761
Average	buckshot	9	0.322730689	0.037758047
Average	buckshot	10	0.340703199	0.033024492
Average	buckshot	11	0.36755005	0.031108142
Average	buckshot	12	0.388440225	0.030373479
Average	buckshot	13	0.405359919	0.027434904
Average	buckshot	14	0.4325032	0.026850656
Average	buckshot	15	0.452612811	0.025512628
Average	buckshot	16	0.469816654	0.024441717
Average	buckshot	17	0.490833336	0.022218317
Average	buckshot	18	0.511846214	0.021444349
Average	buckshot	19	0.520318217	0.021361947
Average	buckshot	20	0.543706018	0.021172363
Predicted	kmeans	3	0.102539672	0.1084
Predicted	kmeans	4	0.143	0.0879
Predicted	kmeans	5	0.178646265	0.0756
Predicted	kmeans	6	0.210872951	0.0674
Predicted	kmeans	7	0.240508448	0.061542857
Predicted	kmeans	8	0.268092496	0.05715
Predicted	kmeans	9	0.294	0.053733333
Predicted	kmeans	10	0.318503927	0.051
Predicted	kmeans	11	0.341810343	0.048763636
Predicted	kmeans	12	0.364079344	0.0469
Predicted	kmeans	13	0.385438243	0.045323077
Predicted	kmeans	14	0.405990265	0.043971429
Predicted	kmeans	15	0.425820485	0.0428
Predicted	kmeans	16	0.445	0.041775
Predicted	kmeans	17	0.463588949	0.040870588
Predicted	kmeans	18	0.481638744	0.040066667
Predicted	kmeans	19	0.49919374	0.039347368
Predicted	kmeans	20	0.516292529	0.0387
Predicted	buckshot	3	0.137611469	0.074534591
Predicted	buckshot	4	0.177	0.0627
Predicted	buckshot	5	0.211701993	0.05462368

Predicted	buckshot	6	0.243074992	0.048661988
Predicted	buckshot	7	0.271925443	0.044028564
Predicted	buckshot	8	0.298778787	0.040293669
Predicted	buckshot	9	0.324	0.0372
Predicted	buckshot	10	0.347854816	0.034582848
Predicted	buckshot	11	0.370543844	0.032331236
Predicted	buckshot	12	0.392222937	0.030367296
Predicted	buckshot	13	0.413016037	0.028634565
Predicted	buckshot	14	0.433023636	0.02709097
Predicted	buckshot	15	0.452328552	0.02570443
Predicted	buckshot	16	0.471	0.02445
Predicted	buckshot	17	0.489096527	0.023307951
Predicted	buckshot	18	0.506668181	0.022262446
Predicted	buckshot	19	0.523758145	0.021300607
Predicted	buckshot	20	0.540403985	0.02041184

Appendix B

Query	Algorithm	K	Avg Ratio	Avg Score
parking decal	kmeans	3	0.55555556	2.1111111
parking decal	kmeans	5	0.6	2.7333333
parking decal	kmeans	7	0.57142857	2.4761905
computer science	kmeans	4	0.58333333	2.75
computer science	kmeans	7	0.66666667	2.4761905
computer science	kmeans	10	0.76666667	3.1666667
parking decal	buckshot	3	1	3.7777778
parking decal	buckshot	5	0.93333333	4.1333333
parking decal	buckshot	7	0.95238095	3.952381
computer science	buckshot	4	1	3.6666667
computer science	buckshot	7	0.85714286	3.4285714
computer science	buckshot	10	0.86666667	3.2666667