

Homework 6: Clustering

Jef Harkay

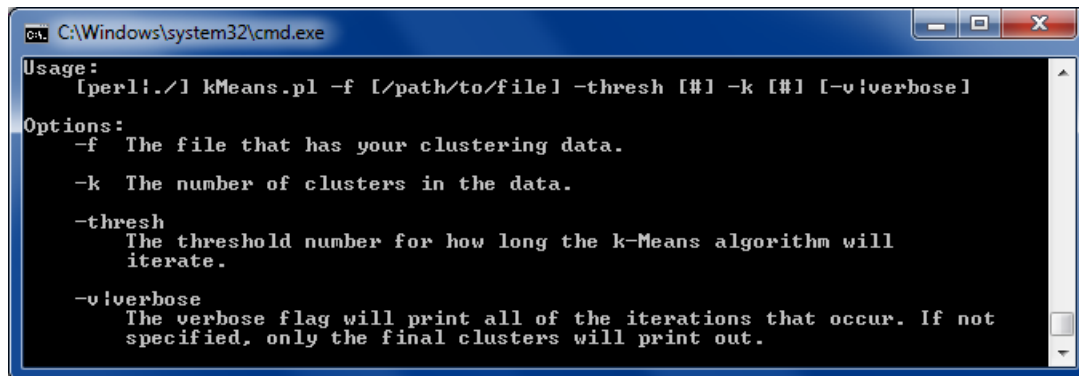
December 3, 2011

1 k-Means

For k-Means, I implemented the algorithm in a perl script.

1.1 Usage

To check the usage statement, just run the perl script without any flags. The result is displayed in Figure 1.



```
C:\Windows\system32\cmd.exe
Usage:
[perl!./] kMeans.pl -f [/path/to/file] -thresh [#] -k [#] [-v!verbose]

Options:
-f The file that has your clustering data.
-k The number of clusters in the data.
-thresh
  The threshold number for how long the k-Means algorithm will
  iterate.
-v!verbose
  The verbose flag will print all of the iterations that occur. If not
  specified, only the final clusters will print out.
```

Figure 1: kMeans.pl usage statement

The script requires that you specify a clustering data file (-f), the number of clusters you think there are (-k), and a threshold value for how long the algorithm iterates (-thresh). The verbose (-v—verbose) option will print out each iteration’s cluster and mean values.

In the algorithm, the initial mean values are chosen by randomly picking points from the clustering data. If you choose a k value of 3, then your initial means will be 3 random points from the clustering data. To prevent the same points from being used, a system of checking the chosen random points was implemented.

1.2 Results

In this section, we will discuss what threshold value is used and the results from playing around with different cluster values.

1.2.1 Threshold Value

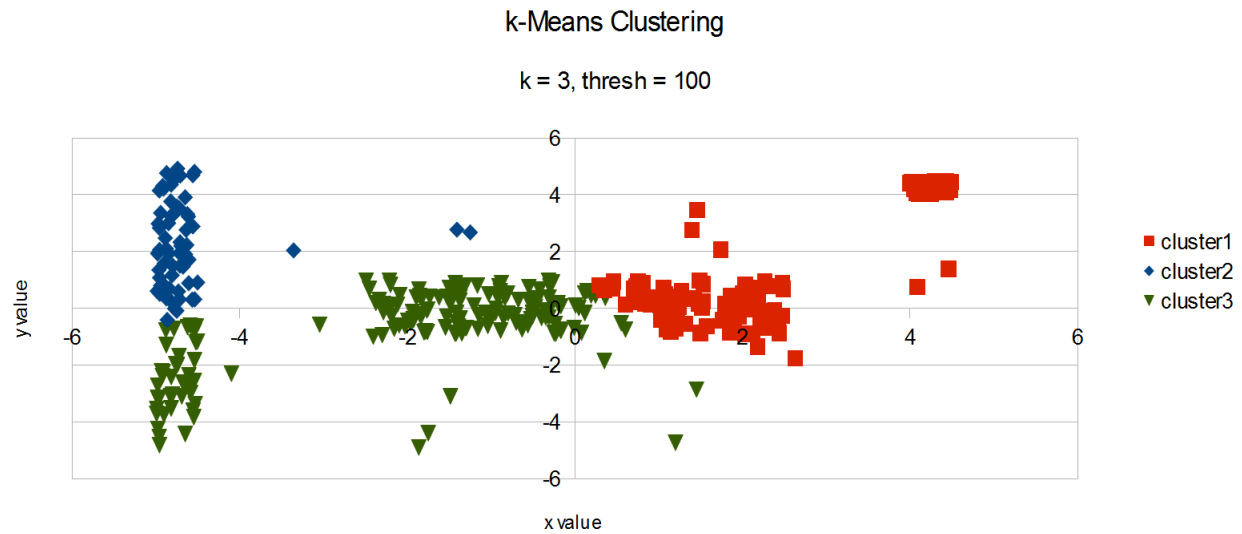


Figure 2: ‘perl kMeans.pl -f clustering_data.txt -k 3 -thresh 100’

Figure 2 shows the final clusters when the threshold of the algorithm is set to 100 iterations. Looking at the results, the algorithm looks like it could run for more iterations to get better clusters because we know those aren't the right clusters. After playing around with the threshold value, it was decided that a value of 1000 gave us pretty good results, so for Figures 3 and 4, that's the value that was used.

1.2.2 Different k Values

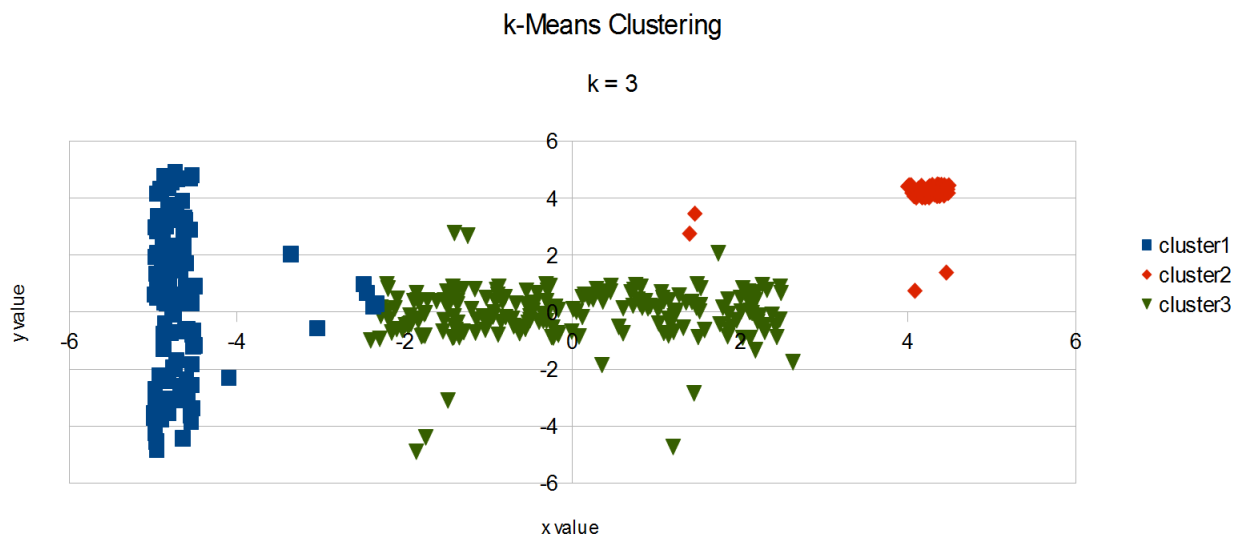


Figure 3: ‘perl kMeans.pl -f clustering_data.txt -k 3 -thresh 1000’

Figure 3 shows the final clusters when the cluster value is 3. Judging by the actual labelled clusters vs. the plotted clusters, it looks like we get pretty close to the right clusters. I realize that 3 clusters isn't the right amount (we actually have 4 clusters when including the noise), but if 4 clusters were chosen, we get Figure 4.

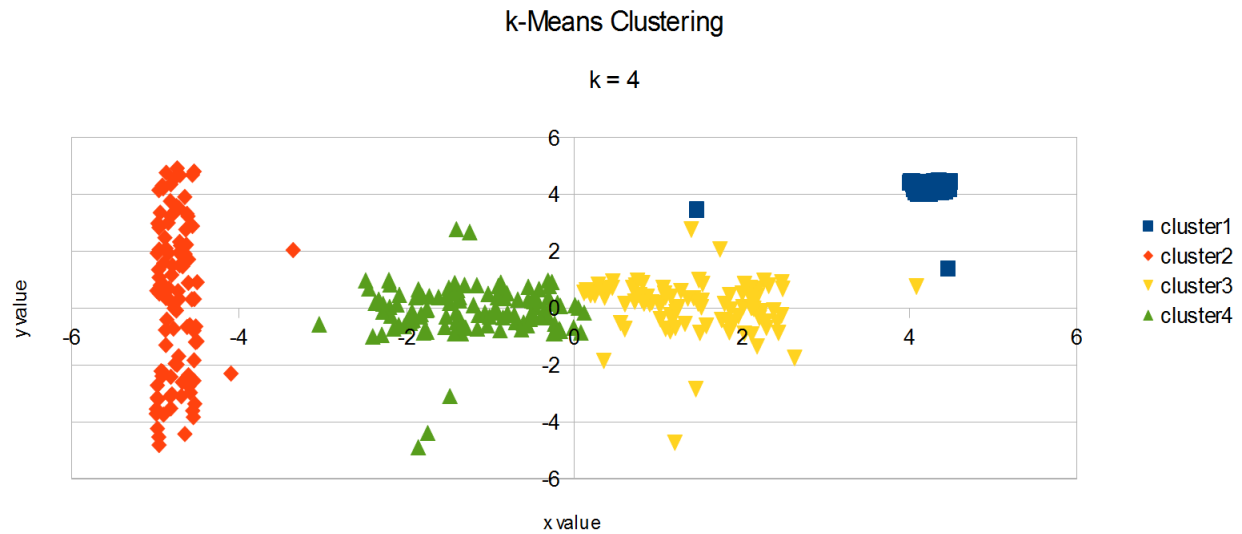


Figure 4: 'perl kMeans.pl -f clustering_data.txt -k 4 -thresh 1000'

Figure 4 shows that k-Means doesn't do good job of clustering the noise in one group. It makes sense that k-Means doesn't do a good job because it's only calculating distances, and nothing more.

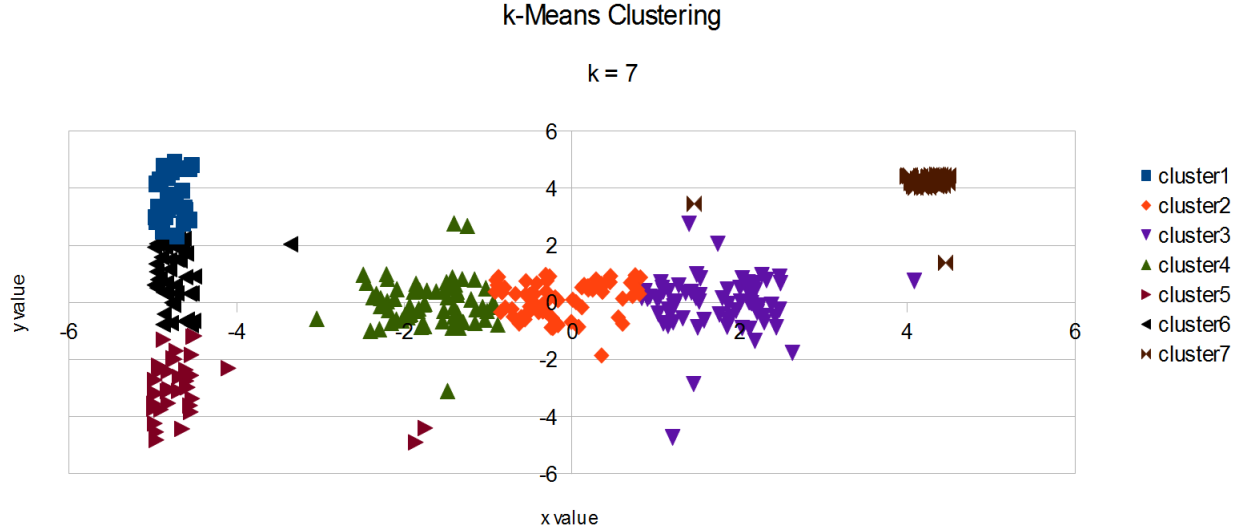


Figure 5: ‘perl kMeans.pl -f clustering_data.txt -k 7 -thresh 1000’

Figure 5 shows the final clusters when the cluster value is 7. We know there aren’t 7 clusters in the original cluster data set, so this is a bad run. The final clusters do have closely related points together, but it doesn’t represent the correct clusters.

2 Hierarchical Clustering

For a Hierarchical Clustering algorithm, I made use of Sergio Gómez’s software “MultiDendrograms.” Mr. Gómez implemented an Agglomerative Hierarchical Clustering approach in Java. Agglomerative starts with all the points as the clusters, then it finds all sets of two points that are closest together, and continues to do so until it finds one final cluster. Basically, you start with all points and converge to one final cluster.

Before using Mr. Gómez’s program, I had to calculate the Euclidean distance between every point. The distance.pl script starts with $point_1$ and calculates its distance from $point_2$, $point_3$, ..., $point_n$. Then the algorithm goes to $point_2$ and calculates the distance from $point_1$, $point_3$, ..., $point_n$, and so on and so forth. Using the output from distance.pl, I created distance.txt, which included all of these Euclidean distances. I then loaded distance.txt into the “MultiDendrograms” program to create the proper dendrograms.

2.1 Linkage Types

There are two linkage types that I used when running the Hierarchical Clustering algorithm: **Complete-linkage** and **Single-linkage**. Complete-linkage comes up with the maximum distance between all the points in cluster 1 against cluster 2. Single-linkage calculates the shortest distance between all the points in cluster 1 against cluster 2. To help clarify this point, Figure 6 was created.

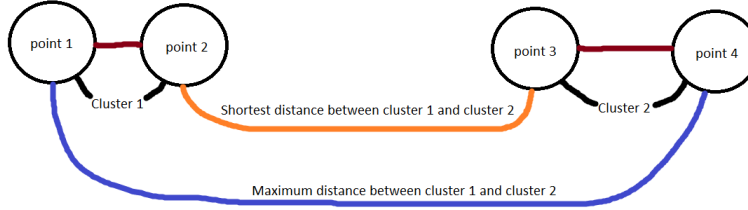


Figure 6: Linkage Types

In Figure 6, the orange line from *point*₂ to *point*₃ shows the shortest distance, or Single-linkage, between the two clusters. The blue line from *point*₁ to *point*₄ shows the maximum distance, or Complete-linkage, between the two clusters. The problem with Single-linkage is that it only looks at the two closest points. Figure 7 shows this problem.

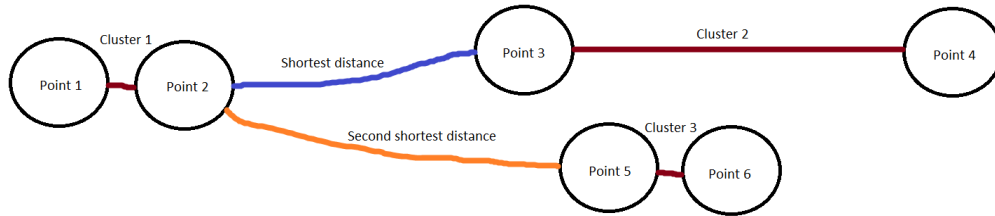


Figure 7: Linkage example

In Figure 7, let's pretend *cluster*₂ and *cluster*₃ are closer to *cluster*₁ than they are each other. Let's also say *point*₃ is closer to *point*₂ than *point*₅ to *point*₂, *point*₆ is closer to *point*₁ than *point*₄ to *point*₁, and the distance between *point*₃ and *point*₄ is much greater than *point*₅ to *point*₆.

If all of that's true, Single-linkage would pick *cluster*₂ over *cluster*₃, even though the distance from *point*₃ to *point*₄ is much greater than *point*₅ to *point*₆. This is where Single-linkage fails. Complete-linkage would have chosen *cluster*₃ because the distance from *point*₆ to *point*₁ is less than *point*₄ to *point*₁, which means *cluster*₃ has a much smaller distance (more similarity) between its points than *cluster*₂.

2.2 Results

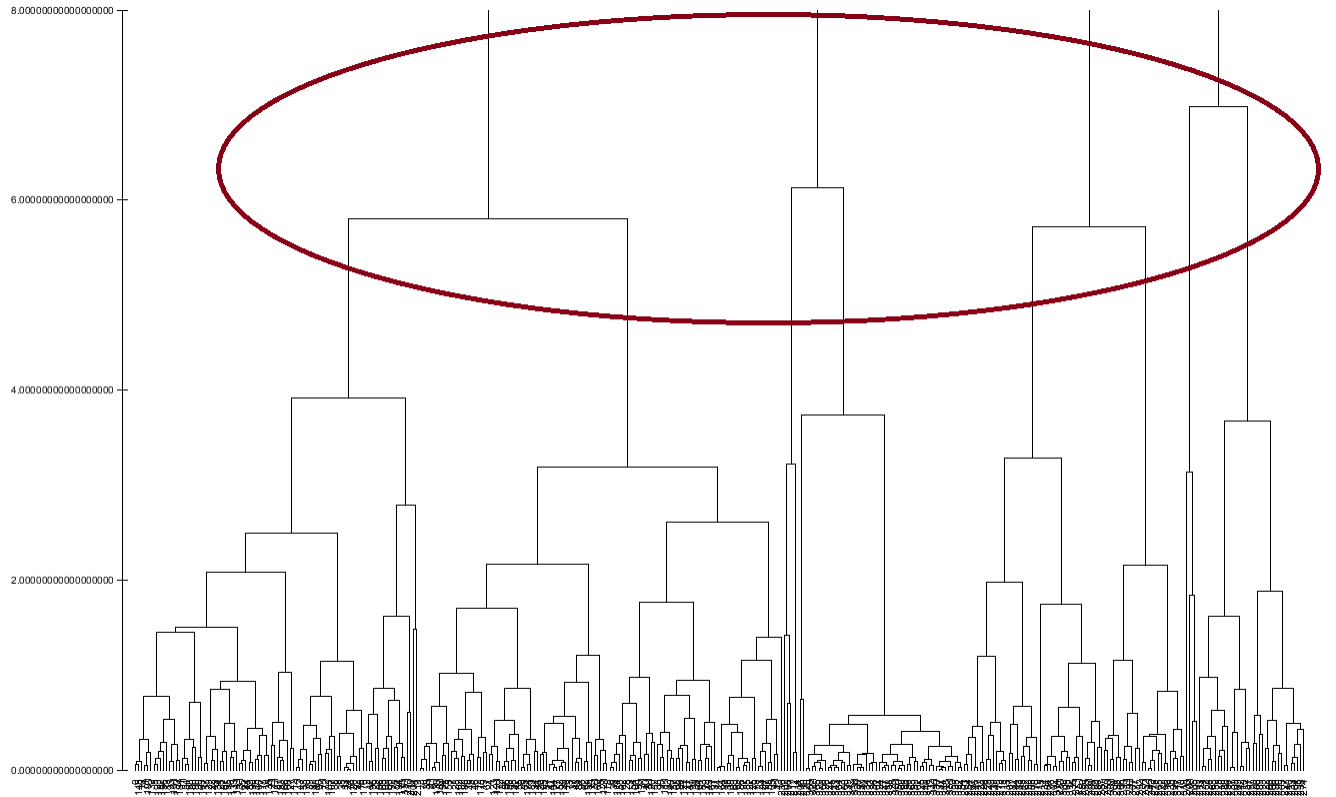


Figure 8: Complete-linkage run

In Figure 8, the dark red circle is showing the top 4 clusters from using complete linkage with the Hierarchical Clustering algorithm. Starting from the left in the red circle, the large cluster is most likely the CENTER cluster, followed by the NOISE cluster, then the LEFT cluster, ending with the TOPRIGHT cluster.

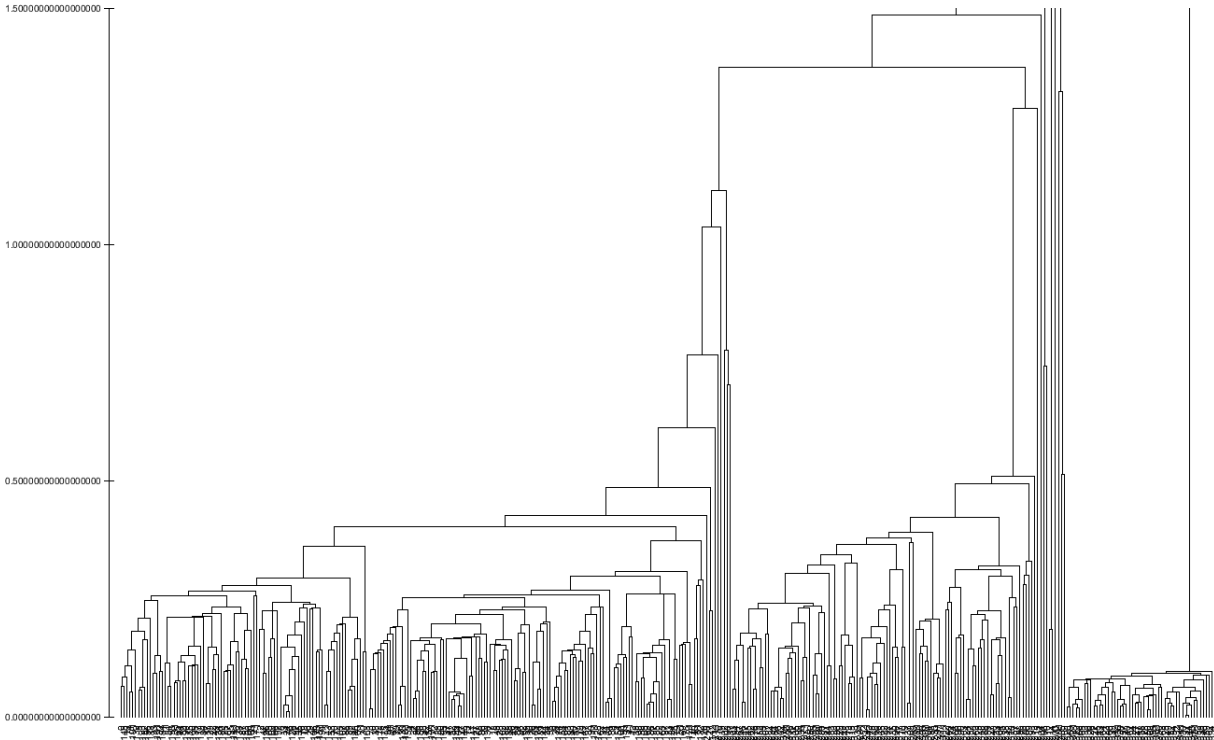


Figure 9: Single-linkage run

Figure 9 is pretty awful. The clustering seems to be all over the place, and there doesn't seem to be any sign of our 4 clusters.

3 Conclusion

k-Means seems to be a pretty solid algorithm if you know the proper threshold and number of clusters. However, if you don't know those values, then there is much trial and error to be done. Also, another drawback for k-Means is that it can't handle noisy data very well.

Hierarchical Clustering is pretty neat because you feed it data and it decides what your clusters are. It even looks like it groups noisy data properly. The drawback with this algorithm is computation/complexity. Hierarchical Clustering has basically been abandoned because it's extremely time consuming to run, especially on larger datasets. It's unfortunate because, to me, this seems like a truly unsupervised algorithm—all you have to do is send it data and not worry about things like a threshold or k value. Granted, the Agglomerative has to "know" when to stop, so you have the right iteration level, but if a Divisive algorithm was used, it would have stopped at the proper amount of clusters.

4 References

Sergio Gómez, "MultiDendrograms: A Hierarchical Clustering Tool", <http://deim.urv.cat/~sgomez/-multidendrograms.php>.

5 kMeans.pl Code

```
use warnings;
use strict;
use Pod::Usage;
use Getopt::Long;
use List::Util 'shuffle';

my $file;
my $thresh;
my $k;
my $verbose;

GetOptions( "f=s" => \$file,
            "thresh=i" => \$thresh,
            "k=i" => \$k,
            "v|verbose" => \$verbose);
pod2usage(1) unless $file and $thresh and $k;

open my $fh, '<', $file or die "Can't open $file! $!\n";
my @x = ();
my @means = ();
my @clusters = ();

while (my $line = <$fh>) {
    chomp $line;
    my @split = split(/\s+/, $line);
    push @x, [@split];
}
@x = shuffle(@x);
my $num_points = (scalar @x[0]) - 1;
my %randos = ();

# Picking a random point in the dataset as our starting means.
my $range = scalar @x;
for (my $i = 0; $i < $k; $i++) {
    my $rand = int(rand($range));
    while (1) {
        if ($randos{$rand}) {
            $rand = int(rand($range));
        }
    }
}
```

```

    }
    else {
        $randos{$rand} = 1;
        last;
    }
}
push @means, $x[$rand];
}

my $count = 0;
while ($count++ < $thresh) {
    @clusters = clusterize();
    @means = remean(\@clusters);

    if ($verbose) {
        for (my $i = 0; $i < scalar @clusters; $i++) {
            for (my $j = 0; $j < scalar @{$clusters[$i]}; $j++) {
                for (my $k = 0; $k < scalar @{$clusters[$i][$j]} - 1; $k++) {
                    if ($k == 0) {
                        print $clusters[$i][$j][$k];
                    }
                    else {
                        print ", ", $clusters[$i][$j][$k];
                    }
                }
                print ", ", "cluster", $i + 1, "\n";
            }
        }
        for (my $i = 0; $i < scalar @means; $i++) {
            for (my $j = 0; $j < scalar @{$means[$i]}; $j++) {
                if ($j == 0) {
                    print $means[$i][$j];
                }
                else {
                    print ", ", $means[$i][$j];
                }
            }
            print "\n";
        }
        print "\n";
    }
    elsif ($count == $thresh - 1) {
        for (my $i = 0; $i < scalar @clusters; $i++) {
            for (my $j = 0; $j < scalar @{$clusters[$i]}; $j++) {
                for (my $k = 0; $k < scalar @{$clusters[$i][$j]} - 1; $k++) {

```

```

        if ($k == 0) {
            print $clusters[$i][$j][$k];
        }
        else {
            print ", ", $clusters[$i][$j][$k];
        }
    }
    print ", ", "cluster", $i + 1, "\n";
}
}
}

sub remean {
    my $clusters = $_[0];
    my @means = ();
    for (my $i = 0; $i < scalar @{$clusters}; $i++) {
        my @mean = ();
        my $cluster_size = scalar @{$clusters->[$i]};
        my $num_points = (scalar @{$clusters->[$i][0]}) - 1;
        for (my $k = 0; $k < $num_points; $k++) {
            for (my $j = 0; $j < $cluster_size; $j++) {
                $mean[$k] += $clusters->[$i][$j][$k];
            }
            $mean[$k] /= $cluster_size;
        }
        push @means, [@mean];
    }
    return @means;
}

sub clusterize {
    my @clusters;
    for (my $i = 0; $i < scalar @x; $i++) {
        my $cluster = distance(\@{$x[$i]}, \@means);
        push @{$clusters[$cluster]}, $x[$i];
    }
    return @clusters;
}

sub distance {
    my ($ref_x, $ref_means) = @_;
    my $cluster;
    my $fin_dist = 0;
    for (my $i = 0; $i < $k; $i++) {

```

```

my $dist = 0;
for (my $j = 0; $j < $num_points; $j++) {
    $dist += abs($ref_x->[$j] - $ref_means->[$i][$j]) ** 2;
}
$dist = sqrt($dist);
if ($fin_dist == 0 || $dist < $fin_dist) {
    $fin_dist = $dist;
    $cluster = $i;
}
}
return $cluster;
}

```

--END--

=head1 SYNOPSIS

```
[perl | ./] kMeans.pl -f [/path/to/file] -thresh [#] -k [#] [-v|verbose]
```

=head1 OPTIONS

=over 4

=item B<-f>

The file that has your clustering data.

=item B<-k>

The number of clusters in the data.

=item B<-thresh>

The threshold number for how long the k-Means algorithm will iterate.

=item B<-v|verbose>

The verbose flag will print all of the iterations that occur. If not specified, only the final clusters will print out.

=back

=head1 DESCRIPTION

This program is an implementation of the k-Means algorithm.

=cut

6 distance.pl Code

```
use strict;
use warnings;
use Pod::Usage;
use Getopt::Long;

my $file;

GetOptions( "f=s" => \$file );
pod2usage(1) unless $file;

open my $fh, '<', $file or die "Can't open file $file! $!\n";
my @x = ();
my @final = ();

while (my $line = <$fh>) {
    chomp $line;
    my @split = split(/\s+/, $line);
    push @x, [$split[0], $split[1]];
}

for (my $i = 0; $i < scalar @x; $i++) {
    for (my $j = 0; $j < scalar @x; $j++) {
        next if $j == $i;
        print "$i $j ", sqrt(($x[$i][0] - $x[$j][0]) ** 2 + ($x[$i][1] - $x[$j][1]) ** 2);
    }
}

__END__

=head1 SYNOPSIS

[perl|./] kMeans.pl -f [/path/to/file]

=head1 OPTIONS

=over 4

=item B<-f>
```

The file that has your clustering data.

=back

=head1 DESCRIPTION

This program is calculating the euclidean distance **for** every point. So it first looks at point 0 with every other point, then point 1 with every other point, and so on and so forth.

=cut