What is Clustering

- Identifying groups of similar instances.
- Identifying the generation mechanisms that produced the instances.
- Clustering Algorithms vary by:
  - Hierarchical vs Non-hierarchical structure
  - Soft vs hard assignment
  - Measures of similarity/distance
  - Linkage versus non-linkage.
Module Outline

• Next 2-3 classes
  – Basic algorithms

• Next week
  – Speeding up clustering using KD-Trees and geometric reasoning
  – Incorporating background knowledge into clustering as well as speeding up algorithms.
Why Do Clustering

• Typical problems
  – Find groups of similar stars, people or transactions.

• Typical uses
  – Compression
  – Outlier detection
  – Classification/prediction
  – Simplify other mining tasks
Clustering Applications - 1

Lots of examples of clustering “traditional” vector data
  Customer demographic data base,
  Transactions

Image Segmentation

Spatial and Feature Space Clustering: Applications in Image Analysis, 6th Int. Conf. on Computer Analysis and Patterns, Prague, Czech Republic
Clustering Applications - 2

More recently applications to non-vector data

Sequences (Web-usage)  Curves/Trajectories (Web-usage)

Trajectory clustering using mixtures of regression models
S. Gaffney and P. Smyth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 1999

Visualization of navigation patterns on a Web site using model-based clustering
Clustering Applications - 3

Alignment Then Clustering

Graphs showing data points and curves for different datasets.
Clustering Graphs

- Social networks
- Enron data
- Finding clusters in graphs
Types of Clustering
Hierarchical vs Non-Hierarchical

**Agglomerative Hierarchical Clustering**
1. Initially, every instance is in its own cluster
2. Compute similarities between each cluster
3. Merge two most **similar** clusters into one.
4. Goto 2
Also divisive hierarchical clustering.

Time complexity?

**Non-Hierarchical Clustering**
Find the “best” grouping of the instances into $k$ clusters/groups/classes
Linkage versus Non-Linkage

• Linkage Clustering
  – Start by linking together the instances.
  – Then identify groups.
  – This identifies the groups and also how the elements within the groups are linked.

• Non-linkage Clustering
  – No connection between instances is recorded, only the cluster prototypes/centroids.
Example
K-Means Clustering Problem

• Input
  – Set of observations \( x_{1..n,1..j} \) and \( k \) (the number of clusters to form).

• Processing
  – Find the “best” set partition that minimizes the VQ error. Problem complexity.

• Output
  – A “description” or exemplar for each cluster
  – For every instance
    • A “distance” to each cluster center
    • The cluster it is assigned to.

• Deriving the K-Means algorithm
Notion of Distance in K-Means

• Typically Use Euclidean Distance

\[ d = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \]

• Other measures of distance, Manhattan, Mahalanobis.

• Distance measure depends on attribute type
  – Continuous attributes
  – Categorical Attributes
  – Binary attributes
K-Means Objective/Loss Function

• Distortion/Vector Quantization Error

\[ E = \frac{1}{2} \sum_{i=1}^{n} D(x_i, C_j), \text{where } x_i \in C_j \]

• What is the solution which provides a zero distortion?

• Only finds a local optima
Upto Here
Hierarchical vs Non-Hierarchical

**Agglomerative Hierarchical Clustering**
1. Initially, every instance is in its own cluster
2. Compute similarities between each cluster
3. Merge two most similar clusters into one.
4. Goto 2

Time complexity, space complexity

Also divisive hierarchical clustering.
Examples - 2

Rice-2b
Rice-2a
Maize-2
Wheat-2
Sorghum-2

Barley-1
Wheat-1
Maize-1
Sorghum-1

Arabidopsis
Many Ways to Define “Closest”

• Distance between centroids
Example: $n=6$, $k=3$, closest pair of centroids

Centroid after first step.

Centroid after second step.
Single Link Agglomerative Clustering

• Use maximum similarity of pairs:
  \[ sim(c_i, c_j) = \max_{x \in c_i, y \in c_j} sim(x, y) \]

• Can result in “straggly” (long and thin) clusters due to chaining effect.
  – Appropriate in some domains, such as clustering islands: “Hawai’i clusters”

• After merging \( c_i \) and \( c_j \), the similarity of the resulting cluster to another cluster, \( c_k \), is:
  \[ sim((c_i \cup c_j), c_k) = \max(sim(c_i, c_k), sim(c_j, c_k)) \]
Single Link Example
Complete Link Agglomerative Clustering

• Use minimum similarity of pairs:
\[ \text{sim}(c_i, c_j) = \min_{x \in c_i, y \in c_j} \text{sim}(x, y) \]

• Makes “tighter,” spherical clusters that are typically preferable.

• After merging \( c_i \) and \( c_j \), the similarity of the resulting cluster to another cluster, \( c_k \), is:
\[ \text{sim}((c_i \cup c_j), c_k) = \min(\text{sim}(c_i, c_k), \text{sim}(c_j, c_k)) \]
Complete Link Example
Computational Complexity

• In the first iteration, all HAC methods need to compute similarity of all pairs of $n$ individual instances which is $O(n^2)$.

• In each of the subsequent $n-2$ merging iterations, it must compute the distance between the most recently created cluster and all other existing clusters.
  – Since we can just store unchanged similarities

• In order to maintain an overall $O(n^2)$ performance, computing similarity to each other cluster must be done in constant time.
  – or $O(n^3)$ if done naively
Group Average Agglomerative Clustering

- Use average similarity across all pairs within the merged cluster to measure the similarity of two clusters.

\[
sim(c_i, c_j) = \frac{1}{|c_i \cup c_j||c_i \cup c_j| - 1} \sum_{\mathcal{r} \in (c_i \cup c_j)} \sum_{\mathcal{p} \in (c_i \cup c_j), \mathcal{p} \neq \mathcal{r}} \text{sim}(\mathcal{r}, \mathcal{p})
\]

- Compromise between single and complete link.
- Two options:
  - Averaged across all ordered pairs in the merged cluster
  - Averaged over all pairs between the two original clusters
- Some previous work has used one of these options; some the other. No clear difference in efficacy
Computing Group Average Similarity

- Assume cosine similarity and normalized vectors with unit length.
- Always maintain sum of vectors in each cluster.
  \[ p_j(c_j) = \sum_{x \in c_j} x \]
- Compute similarity of clusters in constant time:
  \[
  \text{sim}(c_i, c_j) = \frac{(p'(c_i) + p'(c_j)) \cdot (p'(c_i) + p'(c_j)) - (|c_i| + |c_j|)}{(|c_i| + |c_j|)(|c_i| + |c_j| - 1)}
  \]
Efficiency: Medoid As Cluster Representative

• The centroid does not have to be a document.
• Medoid: A cluster representative that is one of the documents
• For example: the document closest to the centroid
• One reason this is useful
  – Consider the representative of a large cluster (>1000 documents)
  – The centroid of this cluster will be a dense vector
  – The medoid of this cluster will be a sparse vector

• Compare: mean/centroid vs. median/medoid
K-Means Algorithm

1. Randomly assign each instance to a cluster.
2. Calculate the centroids for each cluster
3. For each instance
   1. Calculate the distance to each cluster center
   2. Assign the instance to the closest cluster
4. Goto 2 until distortion is small
$K = 2$ how will the point be divided, where will the clusters be

$K = 3$ "

" " " " " "
\( c_1 = 3, 6, \quad c_2 = 8, 5 \)

\[(x-3)^2 + (y-6)^2 = (x-8)^2 + (y-5)^2 \]

\[x^2 - 6x + 9 + y^2 - 12y + 36 = x^2 - 16x + 64 + y^2 - 10y + 25 \]

\[10x - 2y - 44 = 0\]
K = 3

c1 = 2.5, 2.5, 2.5

C1 vs C2: 
\[-5x -5y + 12.5 = -10x -15y + 81.25\]
\[5x + 10y - 68.75 = 0\]
Clustering Non-Vector Data

• Strings?
• Sequences?
• Curves
• Matrices
Strings

- 010010101010
- 101010101010
Sequences

- How do I measure distances

Table 1: An example of individuals characterized by Web navigation sequences.

<table>
<thead>
<tr>
<th>User 1</th>
<th>Session 1</th>
<th>2 3 2 2 3 3 3 3 1 1 1 3 1 3 3 3 3 3 3</th>
<th>1 2 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Session 2</td>
<td>3 3 3 1 1 1</td>
<td></td>
</tr>
<tr>
<td>User 2</td>
<td>Session 1</td>
<td>7 7 7 7 7 7 7 7 7 7 7 7</td>
<td></td>
</tr>
<tr>
<td>User 3</td>
<td>Session 1</td>
<td>1 5 1 1 5 1 5 1 1 1 1 1 1 1 1 1 1 1 1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Session 1</td>
<td>5 1 1 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Session 3</td>
<td>1 3 3 1 5 1 1 1 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>
Sequences -1

• How do I measure distances

Figure 1: Three joint-transition probability matrices for the Web page-request data. Each square represents joint probability $p(s_i, s_j)$, where $s_i$ and $s_j$ are all combinations of the 10 available states. Lighter squares represent higher probability.
Curves

A typical gene expression data set contains a set of sequences whose values measure the level of response for a certain set of genes over time. For example, a data set might contain 1000 sequences measuring the responses of 1000 different genes over a 24 hour period. This data set might contain 20 response measurements in each sequence, yielding 20,000 measurements in total.

Figure 2: This picture shows the actual gene expression data. Note that only one-fifth of the data set is displayed.
Curves - 2
Properties of K-Means Algorithm - 1

1) The Effect Of Exclusive Assignment
   Can not model overlapping classes, Outliers

2) The Inconsistency of the learning algorithm
   \[
   \lim_{n \to \infty} P(\theta_{\text{TRUE}}) = 1
   \]
   where  \( n \) is the number of observations

3) The algorithm is not invariant to non-linear transformations

4) The algorithm is not invariant to scale transformations
Properties of K-Means Algorithm - 2

5) The algorithm finds the local minima of its loss function, which is the vector quantization error (distortion).

6) The algorithm provides biased class parameter estimates.
Properties of K-Means Algorithm - 3

7) The algorithm requires the a-priori specification of the number of classes

8) Euclidean distance measures can unequally weight attributes

9) Non-parametric modeling of continuous attributes
Distance - Continuous Attributes

- Need to rescale to prevent bias
  \[ Z = \frac{X - \mu}{\sigma} \]
- What about correlations between attributes
Measuring the Quality of Clusters (Stop here)

• Intrinsic properties
  – Maximum diameters
  – Density etc.

• Extrinsic properties
  – Measure homogeneity on extrinsic label values
  – Correlation/Mutual information
  – Rand index
K-Means Clustering Problem

• **Input**
  – Set of observations $x_{1..n,1..j}$ and $k$ (the number of clusters to form).

• **Processing**
  – Find the “best” set partition that minimizes the VQ error. Problem complexity.

• **Output**
  – A “description” or exempllar for each cluster
  – For every instance
    • A “distance” to each cluster center
    • The cluster it is assigned to.

• **Deriving the K-Means algorithm**
K-Means Algorithm

1. Randomly assign each instance to a cluster.
2. Calculate the centroids for each cluster.
3. For each instance
   1. Calculate the distance to each cluster center
   2. Assign the instance to the closest cluster
4. Goto 2 until distortion is small
5-Means Example

From http://www-2.cs.cmu.edu/~dpelleg/kmeans.html
5-Means Example

From http://www-2.cs.cmu.edu/~dpelleg/kmeans.html
Notion of Distance in K-Means

• Typically Use Euclidean Distance

\[ d = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \]

• Other measures of distance, Manhattan, Mahalanobis.

• Distance measure depends on attribute type
  – Continuous attributes
  – Categorical Attributes
  – Binary attributes
K-Means Objective/Loss Function

• Distortion/Vector Quantization Error

\[ E = \frac{1}{2} \sum_{i=1}^{n} D(x_i, C_j), \text{where } x_i \in C_j \]

• What is the solution which provides a zero distortion?

• Only finds a local optima
K-Means Representational Bias, Search Bias and Loss Function

• Represent clusters by their centroids
• Search using gradient descent
• Distortion/Vector Quantization Error
  \[ E = \frac{1}{2} \sum_{i=1}^{n} D(x_i, C_j), \text{where } x_i \in C_j \]
• Minimize distortion must manage information-modeling trade-off
  \[ E_{x \in Q}[\chi(x)] = \omega_0 KL(Q_0 \| P_0) + \omega_1 KL(Q_1 \| P_1) + I(Q \| F) \]
1) **The Effect Of Exclusive Assignment**
   Can not model overlapping classes, Outliers

2) **The Inconsistency of the learning algorithm**
   \[ \lim_{n \to \infty} P(\theta_{\text{TRUE}}) = 1 \]
   where \( n \) is the number of observations

3) **The learning algorithm is not invariant to non-linear transformations**

4) **The learning algorithm is not invariant to scale transformations**
Properties of K-Means Algorithm - 2

5) The learning algorithm finds the local minima of its loss function, which is the vector quantization error (distortion).

6) The learning algorithm provides biased class parameter estimates.
Properties of K-Means Algorithm - 3

7) The learning algorithm requires the a-priori specification of the number of classes
8) Euclidean distance measures can unequally weight attributes
9) Non-parametric modeling of continuous attributes
Distance - Continuous Attributes

- Need to rescale to prevent bias
  \[ Z = \frac{X - \mu}{\sigma} \]
- What about correlations between attributes
Up-to Here
Example: $n=6, k=3$, closest pair of centroids

Centroid after first step.

Centroid after second step.
Single Link Example
Single Link Agglomerative Clustering

- Use minimum distance b/w pairs:
  \[ D(c_i, c_j) = \min_{x \in c_i, y \in c_j} D(x, y) \]

- Can result in “straggly” (long and thin) clusters due to chaining effect.
  - Appropriate in some domains, such as clustering islands: “Hawai’i clusters”

- After merging \( c_i \) and \( c_j \), the similarity of the resulting cluster to another cluster, \( c_k \), is:
  \[ D((c_i \cup c_j), c_k) = \min(D(c_i, c_k), D(c_j, c_k)) \]
Graph Theoretic Interpretation

- Let $d'_n$ now be the distance between the clusters merged at step $n$ (counting from bottom).
- Let $G(d'_n)$ be a graph that has edges between all data points within distance $d'_n$ of each other. Then the clusters after step $n$ are the connected components of $G(d'_n)$. Why?
- Explains potentially long straggly clusters.
- Assumptions?
Single Linkage Analysis and Algorithm

• The time complexity of single-link clustering is $O(n^2)$. Why is this interesting?

• How?
  – First compute all pair-wise distances in $O(n^2)$.
    • We can find the smallest distances and candidate for each instance while doing this. Store in best-merge array
  – In each of the $n-1$ join steps: find the smallest distance in the next-best-merge array.
    • Then merge the two clusters, and update the distance matrix in $O(n)$ (how???)
    • Finally, update the best-merge array in $O(n)$ in each step. (how???)
Complete Link Example
Complete Link Agglomerative Clustering

• Use maximum distance b/w pairs:
  \[ D(c_i, c_j) = \max_{x \in c_i, y \in c_j} D(x, y) \]

• Makes “tighter,” spherical clusters that are typically preferable.

• After merging \( c_i \) and \( c_j \), the similarity of the resulting cluster to another cluster, \( c_k \), is:
  \[ D((c_i \cup c_j), c_k) = \max(D(c_i, c_k), D(c_j, c_k)) \]
Graph Theoretic Interpretation

• Assume each pair of points has a unique distance. Let $d_n$ be the diameter of the cluster formed at step $n$.

• Let $G(d_n)$ be a graph that has edges between all data points within $d_n$ distance from one another.

• Then the clusters after step $n$ are *cliques* of $G(d_n)$. Why?
Complete Linkage Analysis and Algorithm

• Complexity is at worst $O(n^2 \log n)$. One such algorithm follows:
  – For each point (n)
    • Calculate distance to every other point (n)
    • Sort these n points. (nlogn)
  – After join operation, update the distance matrix in O(n).
    How?
  – Merge next pair of clusters by finding smallest distance. O(n)

• So complete-link clustering is harder than single-link clustering. Why what property causes this?
Complete Linkage Analysis and Algorithm

- Complexity is at worst $O(n^2 \log n)$. One such algorithm follows:
  - For each point (n)
    - Calculate distance to every other point (n)
    - Sort these n points. (nlogn)
  - After join operation, update the distance matrix in $O(n)$. How?
  - Merge next pair of clusters by finding smallest distance. $O(n)$
- So complete-link clustering is harder than single-link clustering. Why?
- If best merge partner for $k$ before joining $i$ and $j$ was either $i$ or $j$, then after joining $i$ and $j$ the best merge partner for $k$ after the join need not be $i$ or $j$.
- Why?
- For single-link: distance of the two closest members is a local property that is not affected by merging
- For complete-link distance defined as the diameter of a cluster is a non-local property that can change during merging.
Properties of K-Means Algorithm - 1

1) The Effect Of Exclusive Assignment
   Can not model overlapping classes, Outliers

2) The Inconsistency of the algorithm
   \[ \lim_{n \to \infty} P(\theta_{\text{TRUE}}) = 1 \]
   where \( n \) is the number of observations

3) The algorithm is not invariant to non-linear transformations

4) The algorithm is not invariant to scale transformations
Properties of K-Means Algorithm - 2

5) The algorithm finds the local minima of its loss function, which is the vector quantization error (distortion).

6) The algorithm provides biased class parameter estimates
Properties of K-Means Algorithm - 3

7) The algorithm requires the a-priori specification of the number of classes
8) Euclidean distance measures can unequally weight attributes
9) Non-parametric modeling of continuous attributes
K-Means Algorithm

1. Randomly assign each instance to a cluster.
2. Calculate the centroids for each cluster
3. For each instance
   1. Calculate the distance to each cluster center
   2. Assign the instance to the closest cluster
4. Goto 2 until distortion is small

Complexity?
K-Means Algorithm

1. For each point $x$, find the center in $C^{(i)}$ which is closest to $x$. Associate $x$ with this center.

2. Compute $C^{(i+1)}$ by taking, for each center, the center of mass of points associated with this center.

Minimizes vector quantization error

$$\text{distortion}_\phi = \frac{1}{R} \cdot \sum_x d^2(x, \phi(x))$$

Complexity is $O(knmi)$
Lots of “redundancy” in iterations w.r.t instance assignments
Basic Ideas for Speed Up

• Use geometry to reason.
• Batch updates of instances with notion of owners of hyper-rectangles.
Very Brief Introduction to KD-Trees as Used in this Work

- Binary
- Each node represents a hyper-rectangle of the instance space
  - Root node is the entire instance space
  - The “union” of the hyper-rectangles of the children form the parent’s hyper-rectangle.
- Each node contains:
  - \( h_{\text{max}} \), \( h_{\text{min}} \), number of points, centroid, sum of Euclidean Norms for all points falling within the hyper-rectangle
  - These are sufficient statistics for updating centroids!
- Internal nodes define a split (like a decision tree)
- Leaf nodes store actual points
The Algorithm - 1

We define the distance \( d(x, h) \) between a point \( x \) and a hyper-rectangle \( h \) to be \( d(x, \text{closest}(x, h)) \). For a hyper-rectangle \( h \) we denote by \( \text{width}(h) \) the vector \( h^{\text{max}} - h^{\text{min}} \).

• The basic idea: do batch updating. How???

Definition 1 Given a set of centers \( C \) and a hyper-rectangle \( h \), we define by \( \text{owner}_C(h) \) a center \( c \in C \) such that any point in \( h \) is closer to \( c \) than to any other center in \( C \), if such a center exists.

Theorem 2 Let \( C \) be a set of centers, and \( h \) a hyper-rectangle. Let \( c \in C \) be \( \text{owner}_C(h) \). Then:

\[
d(c, h) = \min_{c' \in C} d(c', h).
\]

Proof by contradiction
The Algorithm - 2

- Ok. We have the definition of an owner and a state there can only be one…now what … exhaustive testing for ownership is time consuming. Naïve way?

**Definition 3** Given a hyper-rectangle $h$, and two centers $c^1$ and $c^2$ such that $d(c^1, h) < d(c^2, h)$, we say that $c^1$ dominates $c^2$ with respect to $h$ if every point in $h$ is closer to $c^1$ than it is to $c^2$.

Isn’t this a redundant definition of #1???

**Lemma 4** Given two centers $c^1, c^2$, and a hyper-rectangle $h$ such that $d(c^1, h) < d(c^2, h)$, the decision problem “does $c^1$ dominate $c^2$ with respect to $h$?” can be answered in $O(M)$ time.

- Relationship between ownership and domination
- Keep on comparing cluster centroids to determine if one centroid dominates all others for a hyper-rectangle
Visualizing Centroid Domination

Figure 1: Domination with respect to a hyper-rectangle.
$L_{12}$ is the decision line between centers $c^1$ and $c^2$. Similarly, $L_{13}$ is the decision line between $c^1$ and $c^3$. $p_{12}$ is the extreme point in $h$ in the direction $c^2 - c^1$, and $p_{13}$ is the extreme point in $h$ in the direction $c^3 - c^1$. Since $p_{12}$ is on the same side of $L_{12}$ as $c^1$, $c^1$ dominates $c^2$ with respect to the hyper-rectangle $h$. Since $p_{13}$ is not on the same side of $L_{13}$ as $c^1$, $c^1$ does not dominate $c^3$. 
The Algorithm

Update($h, C$):

1. If $h$ is a leaf:
   
   (a) For each data point in $h$, find the closest center to it and update that center’s counters.
   
   (b) Return.

2. Compute $d(c, h)$ for all centers $c$. If there exists one center $c$ with shortest distance:

   If for all other centers $c'$, $c$ dominates $c'$ with respect to $h$ (so we have established $c = \text{owner}(h)$):
   
   (a) Update $c$’s counters using the data in $h$.
   
   (b) Return.

3. Call Update($h_l, C$).

4. Call Update($h_r, C$).
How to do the Batch Update

• Just use hyper-rectangle centers and number of encompassed points for L1 distance for L2 distance use Euclidean norms.
• Like a pseudo-point with non-unit weight
When the Algorithm Will Work?

Figure 2: Visualization of the hyper-rectangles owned by centers. The entire two-dimensional dataset is drawn as points in the plane. All points that “belong” to a specific center are colored the same color (here, K=2). The rectangles for which it was possible to prove that belong to specific centers are also drawn. Points outside of rectangles had to be determined in the slow method (by scanning each center). Points within rectangles were not considered by the algorithm. Instead, their number and center of mass are stored together with the rectangle and are used to update the center coordinates.
Open Issues

• Worst case analysis of improved algorithm is still $O(knmi)!$ Can we determine a bound on the chance of success?
• Other limitations
Extensions to the Basic Algorithm

• Black-listing
  – If a centroid $a$ dominates centroid $b$ for $h$, then it will dominate for all descendents of $h$
  – Remove $b$ from the list of centroids to check when decomposing $h$
Experimental Results - 1

<table>
<thead>
<tr>
<th>points</th>
<th>blacklisting</th>
<th>naive</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>50000</td>
<td>2.02</td>
<td>52.22</td>
<td>25.9</td>
</tr>
<tr>
<td>100000</td>
<td>2.16</td>
<td>134.82</td>
<td>62.3</td>
</tr>
<tr>
<td>200000</td>
<td>2.97</td>
<td>223.84</td>
<td>75.3</td>
</tr>
<tr>
<td>300000</td>
<td>1.87</td>
<td>328.80</td>
<td>176.3</td>
</tr>
<tr>
<td>433208</td>
<td>3.41</td>
<td>465.24</td>
<td>136.6</td>
</tr>
</tbody>
</table>

Table 1: Comparative results on real data.
Run-times of the naive and blacklisting algorithm, in seconds per iteration. Run-times of the naive algorithms also shown as their ratio to the running time of the blacklisting algorithm, and as a function of number of points. Results were obtained on random samples from the 2-D “petro” file using 5000 centers.
Sensitivity to Different Data Sets

• How will the running time change for the basic (non-KD-Tree) algorithm as a function of: number of clusters, instances, attributes?

# points effect is linear
Results for Black-Listing Algorithm

# dimensions effect is super-linear

# number of centers effect is linear
Approximate Clustering – Not Everyone’s A Winner

Descend down the tree until the ratio of the hyper-rectangle volume to the instance space volume is “small”. Then evenly divide these points amongst all competitors (The ones that haven’t been ruled out by blacklisting).

Our pruning criterion is:

\[ n \cdot \sum_{j=1}^{M} \left( \frac{\text{width}(h)_j}{\text{width}(U)_j} \right)^2 \leq d^i \]

where \( n \) denotes the number of points in \( h \), \( U \) is the “universal” hyper-rectangle bounding all of the input points, \( i \) is the iteration number, and \( d \) is a constant, typically set to 0.8.
Approximate Clustering - Results

Note, no longer equivalent to standard k-means
As expected, run time is better, but distortion is higher.

Figure 7: Runtime of approximate clustering. Running time, in seconds per iteration, is shown as the number of points varies. Each line stands for a different algorithm.

Figure 8: Distortion of approximate clustering. Distortion for approximate and exact clustering. Each line stands for a different algorithm.