Topics

- Large-scale Analytics 2. Clustering: K-Means Clustering Using MapReduce

Clustering: Core concept

- Set of N-dimensional vectors
  - Can be in the order of millions
- Group (or cluster) them based on their proximity (or similarity) to each other in an N-dimensional space
  - Vectors or objects in a cluster (or group) are more similar to each other than in any other group
**k-Means Clustering**

- **Unlabeled dataset**
- Aims to partition \( m \) observations into \( k \) clusters
- Each observation belongs to the cluster with the nearest mean

**Example: Clustering movies**

- Represent each movie by the set of users
- Each movie is a vector \( \{x_1, x_2, \ldots, x_i\} \), where \( x_i \) is the rating provided by user \( i \)
- Similar movies have similar ratings from the same sets of users

**Example: Protein Sequences**

- Objects are sequences of \( \{C, A, T, G\} \)
- Distance between two sequences is the edit distance, or the minimum number of inserts and deletes needed to change one sequence to another
- Clusters correspond to proteins with similar sequences

**k-Means algorithm**

1. **Input**
   - \( k \) (number of clusters)
   - Training set \( \{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\} \)
   - \( x^{(i)} \in \mathbb{R}^n \)
   - (drop \( x^{(0)} = 1 \) convention)

2. **Randomly initialize \( k \) cluster centroids \( \mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n \)**

   ```
   \begin{align*}
   \text{repeat} & \\
   \text{for } i = 1 \text{ to } m & \\
   c^{(i)} & = \text{index (from 1 to } K) \text{ of cluster centroid closest to } x^{(i)} \\
   \text{for } k = 1 \text{ to } K & \\
   \mu_k & = \text{average (mean) of points assigned to cluster } k \\
   \end{align*}
   ```
Assign centroids

Reassign Cluster

Recompute Centroids

Reassign Clusters

Recompute Centroids – Done!

Questions

• What is a good value for K?
• Does K-means always terminate?
• How should we choose initial cluster centers?
k-Means for non-separated clusters

- Separated clusters
- Non-Separated clusters

How to choose the number of clusters

- Value $k$ in the algorithm

Choosing the value $K$

- Elbow Method
- Correct value for $k$

K-Means terminates

- Consider the objective function.
- There are finitely many possible clusterings ($K^n$)
- Each time we reassign a point to a nearer cluster, the objective decreases.
- Every time we recompute the centroids, the objective either stays the same or decreases.
- Therefore the algorithm has to terminate.

Distance Measures

- Euclidean Distance
- Manhattan Distance
- Cosine Distance
- Hamming Distance
- Jaccard Dissimilarity
- Edit Distance
- Smith Waterman Similarity
- Image Distance
- Etc.
### Distance Measures

- Euclidean Distance
- Manhattan Distance
- Cosine Distance
- Hamming Distance
- Jaccard Dissimilarity
- Edit Distance
- Smith Waterman Similarity
- Etc.

**Jaccard Dissimilarity**

\[ \text{Jaccard Similarity} = 1 - \frac{\text{Jaccard Dissimilarity}}{2} \]

**Cosine Similarity**

\[ \text{Cosine Similarity} = 1 - \frac{\text{Cosine Dissimilarity}}{2} \]

**Hamming Distance**

- Between two binary vectors, the number of positions at which the corresponding bits are different.

\[ \text{Hamming Distance} \text{ between } 100 \text{ and } 011 \text{ is } 3 \]

### Edit distance

- **Levenshtein distance**
  - A minimal edit script that transforms the former into the latter.

- "kitten" vs. "sitting"
  - Delete k at 0.
  - Insert s at 0.
  - Delete e at 4.
  - Insert i at 4.
  - Insert g at 6.
  - The total cost is 5 operations.

### Wagner-Fischer algorithm

\[
\begin{array}{cccccc}
 & k & l & t & e & n \\
k & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
l & 1 & 1 & 2 & 3 & 4 & 5 & 6 \\
t & 2 & 2 & 1 & 2 & 3 & 4 & 5 \\
e & 3 & 3 & 2 & 1 & 2 & 3 & 4 \\
\end{array}
\]

\[
\begin{align*}
\text{Wagner-Fischer algorithm} \\
\text{char s[1..m] and char t[1..n]} \\
\text{for } i \text{ in } [0..m] \text{ do } d[i,0] \leftarrow i \text{; } \text{for } j \text{ in } [0..n] \text{ do } d[0,j] \leftarrow j \text{; } \text{for } i \text{ in } [1..m] \text{ do } \text{for } j \text{ in } [1..n] \text{ do if } s[i] \text{ == } t[j] \text{ then } d[i,j] \leftarrow d[i-1,j-1] \text{; else } d[i,j] \leftarrow \text{min}(d[i-1,j]+1, d[i,j-1]+1, d[i-1,j-1]+1) \text{; return } d[m,n] \\
\end{align*}
\]

### Implementing k-Means using MapReduce

- Apache Mahout

### k-Means using MapReduce

- Computing the Euclidean distance between the sample vectors and the centroids can be parallelized.
  - By splitting the data into individual subgroups and clustering samples in each subgroup separately.
  - By the mapper.

- Recalculating new centroid vectors.
  - Divide the sample vectors into subgroups.
  - Compute the sum of vectors in each subgroup in parallel.
  - Reduce will add up the partial sums and compute the new centroids.

### Questions

- How much of data should be transferred for this steps?
- How do we effectively parallelize this process?

### Canopy clustering algorithm

- **Unsupervised pre-clustering algorithm**
  - Defines the proximity regions.
  - Instead of starting with random points.
  - Often used as preprocessing step for k-Means.

- Major goal of this algorithm is to speed up clustering operations on large datasets.
General Canopy Clustering Algorithm

- Using two thresholds $T_1$ (the loose distance) and $T_2$ (the tight distance), where $T_1 > T_2$

1. Begin with the set of data points to be clustered.
2. Remove a point from the set, beginning a new "canopy".
3. For each point left in the set, assign it to the new canopy if the distance is less than the loose distance $T_1$.
4. If the distance of the point is less than the tight distance $T_2$, remove it from the original set.
5. Repeat steps 2-4, until there are no more data points in the set to cluster.

Generating Input data (1/2)

Generating Input data (2/2)

Generating Canopy centers (Red)

Generating Canopy centers (Green)

Collecting Canopy Centers (Reducer)
Perform Canopy Clustering  (Reducer)

Final Canopy centers

Using **Canopy Clustering Algorithm** to Perform k-means using MapReduce

**Step 1. Canopy generation phase**

- **Mapper**
  - Each mapper processes a subset of the total points and applies the chosen distance measure and thresholds.
  - Input: a subset of the total points
  - Functionality: run the canopy algorithm over a subset
  - Output: `<a_constant_key, canopy_centroid>`
    - we will use only one reducer

- **Reducer**
  - A single reducer
  - Input: `<a_constant_key, [a list of canopy_centroids]>`
  - Functionality: run the canopy algorithm over the local canopy centroids
  - Output: Final set of canopy centroids

**Step 2. Clustering phase (k-Means)**

- **Mapper**
  - Each mapper reads the Canopies produced by the first phase
  - Input: a subset of the total points (one or more data points)
  - Functionality: find the closest centroid to this point
  - Output: `<canopy_centroid_ID, a_point_info>`

- **Reducer**
  - Input: `<canopy_centroid_ID, [a list of points in this cluster]>`
  - Functionality: calculate new centroid for this cluster
  - Output: a new centroid

**Repeat Step 2 with the new set of centroids (from reducer)**