03 - Kmeans

Ethan Graham

20th June 2023

Unsupervised Learning

the training set is **not** annotated, and the system must also learn the classes. K-means is an example of this.

K-Means clustering

Given a set of input samples, group the samples into K-clusters *(we assume k to be known/given)* A cluster k is formed by a set of points, and has mean μ_k.

Formally, our cost function is the sum of distances from points to their associated cluster center. We seek to minimize this. This is difficult, however, as we don't know what points belong to what cluster. We dn't know the center of gravity/mean of the clusters.

Our solution is to **initialize the k clusters randomly**. We then assign every point x_i to its nearest center. We then **update all the centers to be the mean of all points associated to them** and repeat this procedure until convergence. We note that each point is assigned to a single cluster. It is possible in theory that a point lie exactly on the boundary between two clusters, but in practice this never happens.

Stopping criterion

We either set a fixed number of iterations (however this is arbitrary and a bad choice could lead to bad results). Another criterion is, since the algorithm convergences to a stable solution, is to stop the procedure as soon as the difference in assignents between two iterations is small.

Python Implementation

Here is an implementation of kmeans using `numpy`

```
def kmeans(X, k, max_iters=100):
    # Randomly initialize centroids
    centroids = X[np.random.choice(range(X.shape[0]), k, replace=False)]

    for _ in range(max_iters):
        # Assign each data point to the nearest centroid
        distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=-1)
        labels = np.argmin(distances, axis=1)

        # Update centroids
        new_centroids = np.array([X[labels == i].mean(axis=0) for i in range(k)])

        # Check for convergence
        if np.all(centroids == new_centroids):
            break

        centroids = new_centroids

    return labels, centroids
```

Even though the algorithm always coverges to a solution, it isn't necessarily the best. It is dependent on the initialization of cluster centers.

The result is dependent on the initialization

Superpixels

Run k-means algorithm with regularly spaced seeds/means on a grid and using a distance weighted sum of distances in image space and in gray level/color space forms superpixels (large pixels where all associated pixels are of the same color).

Inhomogeneous Data

Euclidean distance seems most appropriate, but in practice data dimenions may have different magnitudes and encode different types of information. This can mess with our distance metrics a lot. **Solutions:** - scale each dimension by subtracting smallest value and scaling the result to be between 0 and 1 - use a different metric like manhattan distance - normalize data (ensures data is distributed in $[0, 1]$)

Clustering compactness

K-means clustering exploits the notion of compactness of clusters (i.e. points of the same class are close together)

Spectral Clustering

If data isn't compact or is shaped weirdly (like donut shaped) when we can use spectral clustering which is graph-based connectivity. We group the points based on edges in a graph. Strong connections indicate points that should be clusters and weaker ones suggest that the graph can be partitioned in these locations.

We can define a metric for computing similarity between two x_i and x_j as follows

$w_{ij} = exp(\frac{-||x_i - x_j||^2}{\sigma \check{s}})$ where σ is a hyperparameter

If we think of the cost of a cut as being the sum of weights connecting the two groups (how similar the groups on either side of the cut are using the metric we just defined) then intuitively we want to cluster in a way that minimizes the cost of a partition.

We can normalize the cut to avoid favouring unbalanced partitions.

Minimizing the normalized cut can be approximated as a generalized eigenvalue problem $(D - W)y = \lambda D$ where W is the similarity matrix between all pairs of points and D is the diagonal degree matrix.were W is the similarity matrix between the solution is obtained by the eigenvector with the second smallest eigenvalue.