Statistical Pattern Recognition and Decision Making Methods - protocol

Implementing the K-Means method

Lukáš Čejka

February 6, 2022

Contents

1 Introduction

When it comes to statistical pattern recognition, one of the basic methods that can be used is the K-Means method. This protocol aims to describe the theory behind it and present its implementation. The overall goal of K-Means is to group observations in clusters based on a mean value - often Euclidean distance on a 2D plane.
The first section will specify the problem further and describe basic theory behind the K-Means method. Next, the second section will solely focus on the implementation. Then, section four will present results on different datasets. Finally, the last section will summarize the contents of this protocol.

2 Theory

The K-Means method consists of partitioning the set of n observations into k clusters. Moreover, in the partition is done in such a manner where an observation is assigned to a cluster with the closest centroid. In other words each observation will have a nearest centroid and thus it will be assigned to this centroid's cluster. A centroid is the middle point of a cluster, specifically, its position is the mean position of all node positions belonging to that cluster.
The K-Means method is iterative, meaning that in every iteration nodes find the nearest centroid and are assigned to that centroid's cluster and then the centroids are moved to the new mean position of that cluster. This process repeats itself until it converges to a solution: the centroids stop moving.
Formally, the problem is described with a set of n observations $\{x_1, x_2, \ldots, x_n\}$. In this protocol, we assume that each observation is a 2-dimensional vector consisting of an x and a y coordinate. This set is then partitioned into

k sets called clusters $\{S_1, S_2, \ldots, S_k\}$ in such a way that minimizes the distance between a nodes and centroids, i.e. minimize the within-cluster sum of squares [1]. In other words, find:

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 = \underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} |S_i| \operatorname{Var} S_i \tag{1}$$

2.1 Algorithm

Turning theory into practically applicable instructions will yield the following algorithm - referred to by many as *Naïve k-means*:

1. Obtain a set of k initial means $m_1^{(1)}, \ldots, m_k^{(1)}$ (the exponent denotes the iteration) as random observations.
2. Assign each observation to the cluster with the nearest centroid according to Euclidean distance:

$$S_i^{(t)} = \left\{ x_p : \left\| x_p - m_i^{(t)} \right\|^2 \leq \left\| x_p - m_j^{(t)} \right\|^2 \ \forall j, 1 \leq j \leq k \right\}, \tag{2}$$

3. Compute the new position of each centroid as the mean position of all observations in its cluster.

$$m_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j \tag{3}$$

4. If the positions of centroids has changed since with respect to the previous iteration, go to step 2, otherwise end the algorithm.

3 Implementation

In this section, the implementation of the K-Means algorithm will be presented. The implementing language was chosen to be *Python* (version 3.9.6[1]) as it is widely used for operations with data and its analysis, furthermore, it allows for straightforward visualization using *Matplotlib*[2].

Core code The core algorithm can be seen in listing 1 and functions used in it can be seen in listings 2 and 3.

Listing 1: Core function of the K-Means algorithm - `self` refers to the K-Means class.

```python
def update_clusters(self):
  """Assigns nodes to clusters with the nearest centroids and updates the centroids."""
  # Reset node assignment to current clusters
  for cluster in self.clusters:
    cluster.nodes = []

  # Assign nodes to clusters - minimum distance to that cluster's centroid
  for node in self.nodes:
    node.assign_to_cluster_by_nearest_centroid(self.clusters)

  # Update centroids of each cluster
  for cluster in self.clusters:
    cluster.update_centroid()
```

[1]Available at: https://www.python.org/downloads/release/python-396/
[2]More info: https://en.wikipedia.org/wiki/Matplotlib

Listing 2: Functions of the `Node` class that are used for deciding what cluster a node will be assigned to.

```python
def assign_to_cluster_by_nearest_centroid(self, clusters: list):
  """Assigns this node to a cluster with the nearest centroid."""
  if not len(clusters) > 0:
    raise AssertionError("Input list must not be empty!")

  cluster = self.get_cluster_by_nearest_centroid(clusters)
  self.assign_to_cluster(cluster)

def get_cluster_by_nearest_centroid(self, clusters) -> Cluster:
  """Returns the cluster whose centroid is closest to this node by position"""
  try:
    distances = [self.distance_to_node(cluster.centroid) for cluster in clusters]
  except AttributeError:
    raise AssertionError("Input variable must be a list of Cluster instances")

def distance_to_node(self, node) -> float:
  """Calculates distance to other node."""
  try:
    # "pos" is an instance of the Position class which contains x and y coordinates
    # along with a function for calculating Euclidean distance
    return self.pos.distance_to_pos(node.pos)
  except AttributeError:
    raise AssertionError(f'Input variable must be an instance of {type(self).__name__}')

def assign_to_cluster(self, cluster):
  """Assigns this node to a cluster"""
  try:
    cluster.nodes.append(self)
  except AttributeError:
    raise AssertionError("Input variable must be an instance of Cluster!")

  return clusters[min_value_index_from_list(distances)[1]]
```

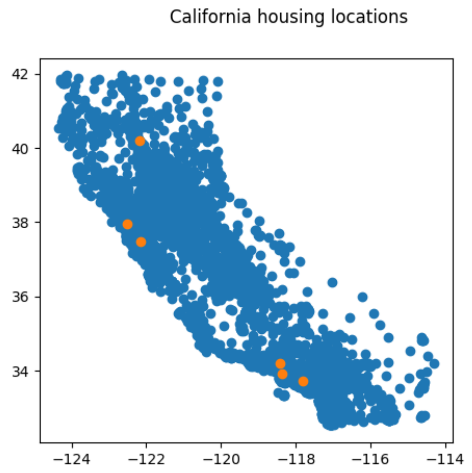
Listing 3: Function that calculates distance between two positions - `self` refers to the `Position` class.

```python
def distance_to_pos(self, pos) -> float:
  """Return Euclidean distance from this position to a given position"""
  try:
    return sqrt(pow((self.x - pos.x), 2) + pow((self.y - pos.y), 2))
  except (AttributeError, TypeError):
    raise AssertionError(f'Input variable must be an instance of {type(self).__name__}')
```

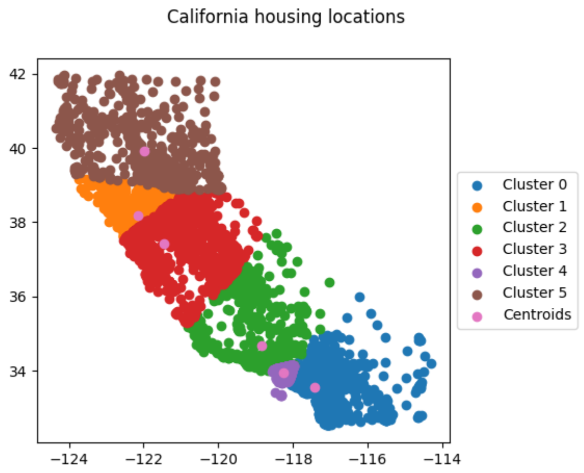
4 Results

For the presentation of results, the *California Housing Prices* dataset[2]. The initial housing locations with randomly selected centroids among them can be seen in figure 1a. The 1st iteration where the nodes have been assigned to the closest centroid and the positions of centroids have been recalculated as the mean position of the nodes of that cluster can be seen in figure 1b.

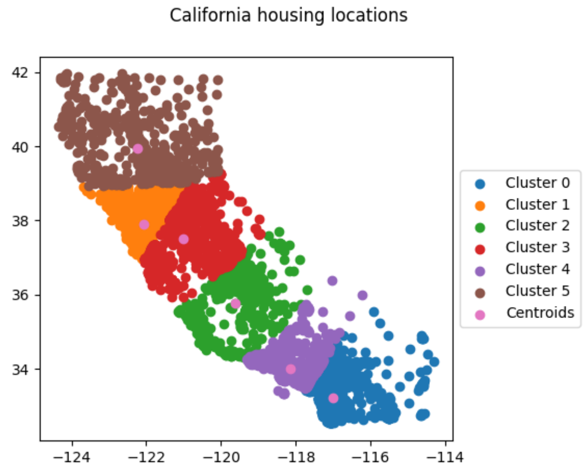
Looking at figure 1f, it is interesting to note that the majority of centroids are in the vicinity of major cities in California, USA. For example, the centroid of Cluster 4 is near Los Angeles, the centroid for Cluster 2 is near



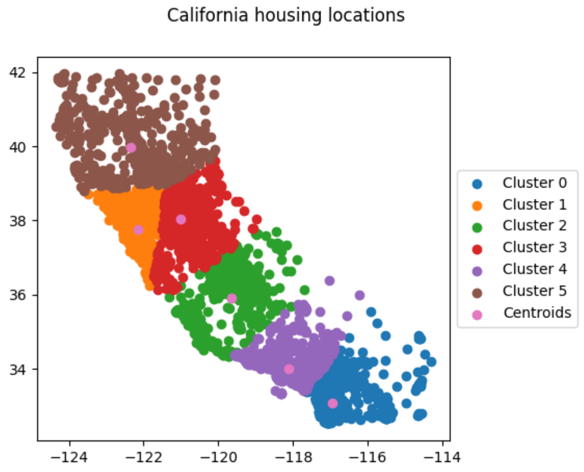
(a) Initial centroids



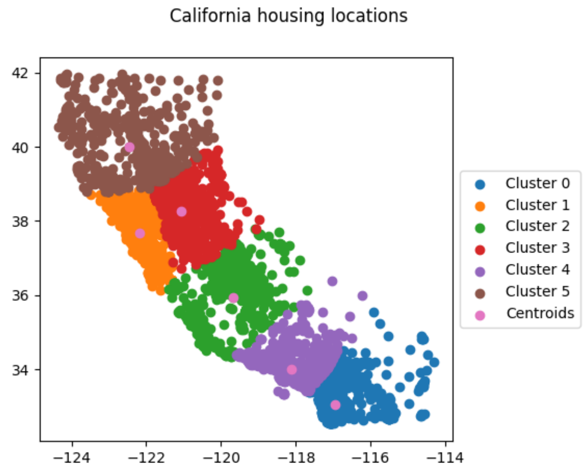
(b) 1st iteration



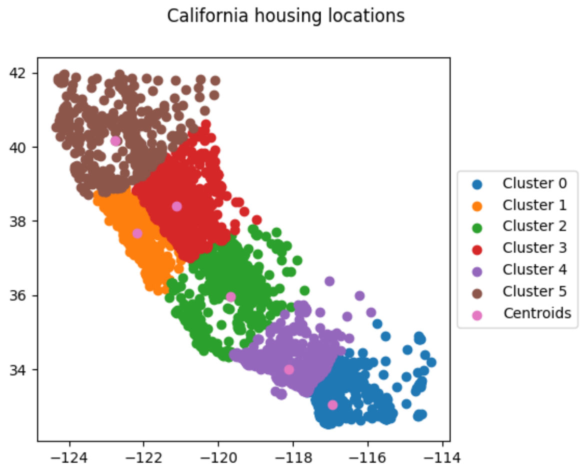
(c) 2nd iteration



(d) 3rd iteration



(e) 4th iteration



(f) Final iteration

Figure 1: K-Means algorithm used on housing locations from the *California Housing Prices* dataset[2].

Bakersfield, the centroid for Cluster 1 is near San Francisco, etc.

5 Conclusion

In summary, the K-Means algorithm was implemented in Python based on different sources[3][1] for the purpose of this assignment project for the *Statistical Pattern Recognition and Decision Making Methods* subject. The main objective of this work was to implement the method and use it on dataset. While the results overall seem to correlate with reality, there are discrepancies which the author was not able to remove or justify - perhaps more data (not just summary housing, but all housing) would be required for a more accurate result.

References

[1] (2001-) K-means clustering. San Francisco (CA). [Online]. Available: https://en.wikipedia.org/wiki/K-means_clustering

[2] C. Nugent. California housing prices. [Online]. Available: https://www.kaggle.com/camnugent/california-housing-prices

[3] P. Sharma. (2019) The most comprehensive guide to k-means clustering you'll ever need. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/