In the name of God

Unsupervised Machine Learning, K-means & DBSCAN

Table of Contents

Researched by:

Kiyan Shafiee

Mohammadmahdi Kazemaini

Arash behnamfar

Supervisor:Dr.Iranidoust

Abstract

In this article, the types of artificial intelligence learning methods are first examined.

Then we provide a summary of our assumptions. These assumptions are not the main purpose of the article but help us as a road map to reach the main title of the article.

In the main part of the article, Unsupervised Learning in AI[1] is explained by explaining K-means and DB Scan algorithms.

In conclusion, a complete research will be presented to you by describing the strengths and weaknesses of the algorithms.

Defaults

In the realm of machine learning and data science, predictive modeling plays a crucial role in deriving insights from data. Among the various algorithms available, Linear Regression, Decision Trees, and Random Forests stand out due to their effectiveness and versatility. Each of these methods has unique characteristics, strengths, and applications, making them suitable for different types of problems.

Introduction

IDK

[1] Artificial intelligence

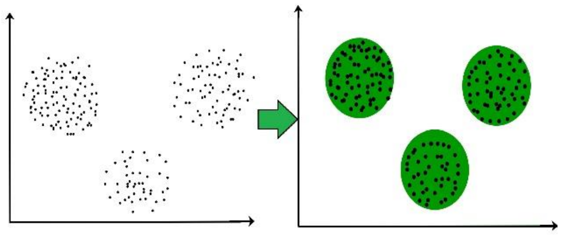
Unsupervised learning

1. Unsupervised Learning in Artificial Intelligence

Unsupervised learning is a critical branch of machine learning where an AI system learns to identify patterns, relationships, and structures in data without the need for labeled input. Unlike supervised learning, where the model is trained on a dataset with input-output pairs (i.e., labels), unsupervised learning deals with data that does not have any associated labels. The primary goal of unsupervised learning is to infer the natural structure present in a set of data points.

2. Key Concepts in Unsupervised Learning

2.1 **Clustering**: Clustering is one of the most common techniques in unsupervised learning. It involves grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups. Popular clustering algorithms include K-means and DBSCAN.



2.2 **Dimensionality Reduction**: This involves reducing the number of random variables under consideration. It is especially useful when dealing with high-dimensional data. Techniques such as Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) help in simplifying the data while retaining its essential characteristics.

2.3 **Anomaly Detection**: Anomaly detection, or outlier detection, involves identifying rare items, events, or observations that raise suspicions by differing significantly from the majority of the data. This is particularly useful in fraud detection, network security, and other areas where the identification of unusual patterns is crucial.

3. Applications of Unsupervised Learning

Customer Segmentation, Image and Speech Recognition, Recommender Systems, Genomics and Healthcare

4. Transition to Discussing K-means, DBSCAN, and Segmentation

In the realm of unsupervised learning, several powerful algorithms and techniques stand out for their ability to uncover hidden patterns and structures within data. Among these, clustering methods such as K-means and DBSCAN are particularly noteworthy. These algorithms excel at grouping data points based on their inherent similarities, enabling a deeper understanding of the underlying data distribution. This process of grouping or clustering is fundamental in tasks such as data segmentation, where the goal is to divide a dataset into distinct and meaningful subgroups. Understanding how K-means, DBSCAN, and segmentation operate not only illustrates the core principles of unsupervised learning but also highlights their practical applications across various domains, from customer segmentation to anomaly detection.

Understanding the K-means Algorithm: A Comprehensive Guide

1. How K-means Works

The K-means algorithm aims to divide a dataset into K clusters, where each data point to the cluster with the nearest center of clusters, known as the cluster centroid. The algorithm operates iteratively and consists of the following steps:

1.1 **Initialization**: The algorithm begins by selecting K initial centroids randomly from the dataset. These centroids represent the initial guess for the centers of the clusters. Sometimes if we select some special points to getter then result may become unacceptable so there are lots of algorithms that try to choose better that random as k-means++. We explain it later.

1.2 **Assignment Step**: Each data point is assigned to the nearest centroid based on the Euclidean distance. This step effectively divides the dataset into K clusters.

1.3 **Update Step** The algorithm recalculates the centroid of each cluster by taking the mean of all data points assigned to that cluster. These new centroids are then used in the next iteration.

1.4 **End of the process**: The assignment and update steps are repeated until the centroids no longer change significantly, indicating that the algorithm has converged. At this point, the clusters are stable, and each data point belongs to the nearest centroid's cluster.

- **Convergence**: The assignment and update steps are repeated until the centroids no longer change significantly, indicating that the algorithm has converged. At this point, the clusters are stable, and each data point belongs to the nearest centroid's cluster.
- **Number of iterations**: We limit number of iterations so if it converges sooner than limit that’s our result and if we reach to iteration limit Everything that has been achieved so far is our result.

2. Key Concepts and Considerations

- **Choosing K points**: One of the most critical aspects of K-means is determining the appropriate number of clusters, K. This choice often requires domain knowledge or experimentation, as an inappropriate K can lead to poor clustering results. Techniques such as the Elbow Method, where the sum of squared distances from each point to its assigned centroid is plotted against different values of K, can help in selecting an optimal number of clusters.
- **Distance Metric**: While Euclidean distance is the most commonly used metric in K-means, other distance measures (like Manhattan distance) can be employed depending on the nature of the data. The choice of distance metric can significantly affect the clustering results.
- **Cluster Shape**: K-means assumes that clusters are roughly spherical and equally sized. It may not perform well if the data contains clusters of varying shapes and sizes, or if the clusters are not linearly separable.
- **Scalability**: K-means is computationally efficient and scales well with large datasets. However, it can be sensitive to the initialization of centroids. To mitigate this, methods such as K-means++, Forgy Method, etc. k-mean++ just change random selecting. It still select first randomly, then select all rest random too. So what’s fair? In the second stage, the calculations are weighted by distance of nearest centroid to each point. So if one point is near a centroid then it has smaller chance to select and if we select first centroid better we converge sooner.

3. Applications of K-means

Customer Segmentation, Image Compression, Document Clustering.

4. Strengths and Limitations

Strengths:

- Simplicity: K-means is easy to understand and implement, making it a popular choice for clustering tasks.
- Efficiency: The algorithm is computationally efficient, especially with large datasets.
- Versatility: K-means can be adapted to different types of data by modifying the distance metric or using techniques like K-means++.

Limitations:

- **Sensitivity to Initial Centroids**: Poor initialization can lead to suboptimal clusters or slower convergence. K-means++ helps mitigate this issue, but it doesn't eliminate it entirely.
- **Difficulty with Complex Clusters**: K-means may struggle with clusters that are not spherical or have different sizes. It also assumes clusters are linearly separable, which may not always be the case.
- **Determining K**: Choosing the right number of clusters can be challenging and often requires experimentation.

5. How to implement K-means in this research

Understanding the DBSCAN Algorithm: A Comprehensive Guide

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a powerful and versatile clustering algorithm, particularly well-suited for identifying clusters of arbitrary shapes in large datasets and dealing with noise or outliers. Unlike K-means, which assumes a fixed number of clusters and tends to work best with spherical cluster shapes, DBSCAN focuses on the density of data points to find clusters. This detailed guide will explore the workings of DBSCAN, its advantages, limitations, and its wide-ranging applications.

1. How DBSCAN Works

DBSCAN clusters data points based on their density. The key idea is that a cluster is a dense region of data points, which is separated by lower-density regions from other clusters. DBSCAN requires two main parameters: ***epsilon (ε)*** and ***minPts***.

- **Epsilon (ε)**: This is the radius that defines the neighborhood around a data point. In simpler terms, it's the maximum distance between two points for them to be considered as part of the same cluster.
- **minPts**: This is the minimum number of points required within the ε-radius to consider a point as a core point of a cluster.

2. The DBSCAN Algorithm Steps:

2.1 **Core Points**: Points that have at least minPts within their ε-radius are considered core points. These points are the backbone of clusters.

2.2 **Directly Density-Reachable Points**: A point p is directly density-reachable from point q if p is within the ε-radius of q and q is a core point.

2.3 **Density-Reachable Points**: A point p is density-reachable from q if there is a chain of points p_1 , p_2 , … , p_n where each $p_{i+}1$ is directly density-reachable from p_i starting from q.

2.4 **Border Points**: Points that are within the ε-radius of a core point but do not have enough neighboring points to be considered core points themselves.

2.5 **Noise Points**: Points that are not within the ε-radius of any core points and cannot be linked to any cluster. These are considered outliers.

3. Clustering Process

- The algorithm starts with an arbitrary point. If it has at least minPts within its ε-radius, it becomes a core point, and a cluster is initiated.
- The cluster is then expanded by recursively including all points density-reachable from the core point.
- If a point cannot be density-reached from any other point, it is labeled as noise.
- The process continues until all points are processed.

4. Advantages of DBSCAN

4.1 **Ability to Find Arbitrary Shaped Clusters**: DBSCAN can identify clusters of various shapes, including those that are not linearly separable. Unlike K-means, which assumes spherical clusters, DBSCAN's density-based approach allows it to handle complex cluster structures.

4.2 **Handling of Noise and Outliers**: DBSCAN effectively identifies and isolates noise points (outliers) that do not belong to any cluster. This makes it particularly useful in real-world datasets where noise is common.

4.3 **No Need to Specify the Number of Clusters**: DBSCAN does not require the user to specify the number of clusters in advance. The algorithm determines the number of clusters based on the data distribution and the chosen ε and minPts parameters.

4.4 **Scalability with Large Datasets**: DBSCAN is efficient for large datasets and can scale well, especially with indexing techniques like k-d trees that optimize neighborhood searches.

5. Limitations of DBSCAN

5.1 **Choice of Parameters**: The effectiveness of DBSCAN heavily depends on the choice of ε and minPts. An inappropriate choice of these parameters can lead to poor clustering results. For example, a very small ε might treat too many points as noise, while a very large ε could merge distinct clusters.

5.2 **Difficulty with Varying Densities**: DBSCAN assumes that the clusters have similar densities. If the dataset contains clusters of varying densities, DBSCAN might struggle to separate them correctly, either splitting a cluster into several or merging different clusters.

5.3 **Computational Complexity**: Although DBSCAN is scalable, its performance can degrade on very high-dimensional data where distance calculations become more complex. In such cases, dimensionality reduction techniques like PCA might be necessary before applying DBSCAN.

6. Applications of DBSCAN

Geospatial Data Analysis, Image Processing, Anomaly Detection, Market Segmentation.

7. Best Practices for Using DBSCAN

7.1 **Parameter Tuning**: Experiment with different values of ε and minPts to find the optimal clustering. The selection of ε can be guided by techniques like the k-distance graph, where the optimal ε is chosen at the “elbow” point.

7.2 **Normalization**: Ensure that the data is normalized, especially if different features have different scales. This is important because DBSCAN relies on distance measures, which can be biased by the scale of the features.

7.3 **Dimensionality Reduction**: For high-dimensional data, consider reducing the dimensionality using methods like PCA or t-SNE before applying DBSCAN. This can improve the performance and accuracy of the clustering.

7.4 **Combine with Other Methods**: Sometimes, combining DBSCAN with other clustering methods (like K-means) can yield better results, especially when dealing with complex datasets that contain both well-separated and dense regions.

8. How to implement K-means in this research

Now that we are quite familiar with DBSCAN,

How can we implement it? How to utilize it's power to find outliers (noise pixels) in our images?

8.1 First step: understanding user inputs

Unlike K-means, we need to input two inputs:

8.1.1 Epsilon
8.1.2 min points

So what does they mean? Why is number of clusters not an input like k-means?

As pinpointed before, DBSCAN needs 2 user inputs:

- **Epsilon**: minimum distance between two points in order to consider them "neighbors."
- **minPts**: minimum number of neighbors a point must have in order to be called core-point.

The expansion of a cluster continues until the reach the last radius of core-points. Then their non-core-point neighbors are added and the expansion stops.

So we may conclude that the number of clusters is not user-defined. And it is true!

The epsilon and minPts determines the number of clusters. Also unlike k-means, we have outliers or noise pixels. They are pixels that do not belong to any cluster.

But what does distance mean in a picture with pixels? The pixels have a consistent distance?

Redefinition of "Distance"

Computer works with numbers right? So we need to see colors as bunch of numbers; as they actually are! They are just bunches of matrices with several dimensions based on their color mode. For example RGB has 3 parameters, grayscale has only 1 parameter, CMYK has 4 parameters. So it is safe to say that we can consider these dimensions as parameters of distance because the amount of difference between color parameters makes the colors more different and distinguishable.

So if we're dealing with an RGB image, we have 5 parameters that contribute to distance:

X, Y, R, G, B .

So a pixel can be next to another pixel but be considered noise because of distant R, G, B parameters.

How the code works?

After traditional steps like putting the image into an array, we defined a DBSCAN class that calculated the distance of each pixel from each other.

The path of optimization!

In *legacy_dbscan.ipynb*, we used a nested loop which had the time complexity of O(n^2). It limited our algorithm to only process small, grayscale and low-resolution images within a reasonable time. So it was time we changed our approach from a simple nested loop to a more advanced move. We became acquaint with KDtree, which is binary tree including matrix nodes. It reduced the time complexity to O(n*log(n)) which changes the computation time drastically in large images.

So it's important to maintain an optimized approach because DBSCAN takes much larger time to calculate in comparison to k-means.

8.2 Selection of eps and minPts:

It depends on the sensitivity of noise detection.

The more sensitive you tune it, the more noises it marks (can be with black color).

We set noise cluster IDs to -1 and other clusters to other IDs, so outliers are distinguishable.

8.3 Expanding it one step further:

One if we wanted to fix the noise? In other words, is morphing it into a near cluster possible?

In *fixer_dbscan.ipynb*, all the pixels with -1 will obtain the id of a nearby cluster. So they are fixed! Morphed into a cluster.