Clustering by Gaussian Mixture Model and Light Gradient Boosting Machine

by

Feihan Yang

A Master's Project Submitted in Partial Fulfillment of the
Requirements for the Degree of

Masters of Engineering

in the Department of Electrical and Computer Engineering

Clustering by Gaussian Mixture Model and Light Gradient Boosting Machine

by

Feihan Yang
University of Victoria

Supervisory Committee

Dr.Xiao-Dai Dong, Supervisor
Department of Electrical and Computer Engineering

Dr.Hong-Chuan Yang, Departmental Member
Department of Electrical and Computer Engineering

ABSTRACT

This project studies clustering by Gaussian mixture model (GMM) and Bayesian Gaussian mixture model (BGMM) combined with light gradient boosting machine (LightGBM) respectively. One common unsupervised learning method for clustering, K-means, serves as the baseline for comparison. LightGBM is an ensemble supervised learning method that combines a number of weak learners to form a strong learner. In this project, LightGBM is combined with BGMM and GMM to improve the clustering performance. A Kaggle competition dataset is used to test these different learning algorithms. Performance evaluation is based on rand index that assesses the similarity between the ground truth clusters and predicted clusters. Moreover, intracluster distances and intercluster distances that indicate the aggregation of the clusters and the separation between different clusters respectively are calculated to generate other performance metrics. In particular, an intercluster distance named multi-cluster average centroid linkage distance is proposed to simplify the distance computation with high precision. The evaluation results reveal that LightGBM with BGMM consistently outperforms the other methods making it a preferred classification approach for the dataset.

Contents

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

my parents for supporting me , understanding and encouraging me in the low moments.

Professor Dong for mentoring, support and patience.

The way to get started is to quit talking and begin doing.
Walt Disney

DEDICATION

I dedicate this project to the coming to an end of my master's career. Hoping to be good at myself in the next stage of life.

Chapter 1

Introduction

Clustering algorithms exploit the underlying structure of the data distribution and define rules for grouping the data with similar characteristics [1]. In today's era of data abundance, the ability to extract meaningful insights from vast and complex datasets is crucial for informed decision-making and problem-solving across various domains. Some of this knowledge can be obtained by observations of mean, variance, and covariance of sensory messages, and perhaps also by a method called "minimum entropy coding" [2]. Clustering, a fundamental technique in unsupervised machine learning, plays a critical role in uncovering hidden patterns, grouping similar data points, and facilitating data exploration and analysis.

This study focuses on addressing an unsupervised classification problem within the field of machine learning. Unsupervised classification named clustering, is a machine learning technique used to group data points into clusters based on similarities or patterns in the data without the use of labels of data points. There are vast landscape of unsupervised machine learning methodologies, each has its unique set of advantages and disadvantages. When choosing unsupervised machine learning methodologies for a problem, we should consider factors such as scalability, interpretability, and computational efficiency. Our choice also should base on the nature of the problem and the characteristics of the data.

Sometimes it becomes necessary to experiment with multiple classification methodologies to ascertain the most suitable approach for the dataset. In this project, five distinct classification methodologies are explored: K-means, Gaussian mixture model (GMM), Bayesian Gaussian mixture model (BGMM) and light gradient-boosting machine (LightGBM) combined with GMM and BGMM separately. Gradient boosting decision tree (GBDT) is a popular machine learning algorithm, in the gradient boost-

ing machine family there are quite a few effective implementations such as XGBoost and pGBRT [9]. We apply LightGBM on the classification results of two unsupervised methods: GMM and GBMM respectively to test if adding one supervised algorithm on unsupervised method can improve the classification performance.

Subsequent to executing these three classification methods on the dataset, a variety of evaluation techniques are employed to assess their efficacy. These evaluation methods include rand measure, intracluster distance and intercluster distance. Furthermore, a calculation method for determining intercluster distance is proposed. Ultimately, the evaluation outcomes indicate that LightGBM emerges as the optimal classification approach for addressing the specific problem at hand.

The advent of Kaggle playground prediction competitions has provided data enthusiasts and machine learning practitioners with an unparalleled platform to test their skills and algorithms on diverse datasets spanning multiple domains. These competitions challenge participants to develop innovative solutions to real-world problems, often involving complex data structures and challenging analytical tasks.

In this project, we will apply five unsupervised clustering methods on a dataset sourced from one Kaggle playground prediction competition. This dataset presents a set of challenges, including the absence of ground truth labels and the need to identify an optimal number of clusters. Our objective is to explore and implement five various clustering methodologies to effectively partition the dataset into meaningful groups. The project begins with a comprehensive exploration and analysis of the dataset, both discrete and continuous features are analyzed, and we employ statistical tests such as the Shapiro-Wilk test, poisson dispersion test, and Q-Q plots to assess the distributional characteristics of the features, providing valuable insights into the dataset's structure and potential preprocessing requirements.

Subsequently, we will implement and evaluate the five clustering methodologies: K-means, GMM, BGMM and LightGBM in conjunction with BGMM or GMM. Each methodology offers unique advantages and challenges, requiring careful consideration and evaluation to determine their effectiveness in clustering the dataset. GMMs are commonly used as a parametric model of the probability distribution of continuous measurements or features in a biometric system, such as vocal-tract related spectral features in a speaker recognition system [19]. To evaluate the performance of each methodology, we employ three evaluation metrics such as the rand index, intracluster distance, and intercluster distance. These metrics provide measures of clustering accuracy, compactness, and separation, enabling us to compare and contrast the

efficacy of different clustering approaches. By leveraging advanced techniques and comprehensive evaluation strategies, we aim to uncover hidden patterns and identify meaningful clusters.

In summary, this project is about clustering a complex dataset using advanced clustering techniques to uncover hidden patterns and insights. Through carefully analysis, implementation and evaluation we can have a comprehensive understanding of these methods. Ultimately, our goal is to find the cluster methodology with the best performance for the dataset and see if adding LightGBM on BGMM or GMM will improve the cluster results than individual unsupervised method.

1.1 Agenda

The report is organized as follows:

Chapter 1 provides an introduction to the report, outlining its goals and objectives.

Chapter 2 gives an overview of the dataset, carries out analysis of discrete and continuous features by three statistical tests; Shapiro-Wilk, poission dispersion, Q-Q Plots.

Chapter 3 introduces four machine learning methods: Kmeans, Gaussian mixture model, Bayesian Gaussian mixture model and LightGBM.

Chapter 4 presents implementation details of each methodology.

Chapter 5 gives evaluation results based on three metrics: rand index, intracluster distance and intercluster distance, and comparison of results and methodologies.

Chapter 6 contains the conclusion of the report, summarizing the key findings and discussing their implications.

Chapter 2

Analyzing the Kaggle Dataset

2.1 Prerequisites

Gaussian mixtures was popularised by Duda and Hart in their seminal 1973 text, 'Pattern Classification and Scene Analysis'. However, it was not until 1977 that the expectation-maximisation (EM) was presented by Dempster, Laird and Rubin in their paper. EM algorithm is the learning algorithm behind a Gaussian mixture model, i.e., it solves the inverse problem by finding the parameters of the model.

A Gaussian distribution, also known as a normal distribution, is a continuous probability distribution that is symmetrical around its mean, representing a bell-shaped curve. In a Gaussian distribution, the majority of the data points cluster around the mean, with fewer data points as you move away from the mean in either direction. The normal distribution (or Gaussian distribution) is a continuous probability distribution characterised by its bell shape. It has the probability density function (PDF) given by

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(x-\mu)^2}{2\sigma^2}) \tag{2.1}$$

where μ is mean and σ is variance. A multivariate Gaussian distribution, also known as a multivariate normal distribution, is an extension of the univariate (single-variable) Gaussian distribution to multiple variables. It is a probability distribution that describes the joint probability distribution of a set of random variables, where each variable may have a different mean and variance, and the variables may be correlated. The probability density function of a multivariate Gaussian distribution for

a p-dimensional random vector X is given by

$$f(\boldsymbol{x}|\boldsymbol{\mu}, \sum) = \frac{1}{((2\pi)^{p/2}\sqrt{\det(\sum)})} \exp(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \sum^{-1} (\boldsymbol{x}-\boldsymbol{\mu})) \tag{2.2}$$

where $\boldsymbol{x}$ is a p-dimensional column vector representing the random variable values, $\boldsymbol{\mu}$ is a p-dimensional column vector representing the mean of the distribution, $\sum$ is a $p \times p$ covariance matrix representing the covariances between the variables, $(\boldsymbol{x}-\boldsymbol{\mu})^T$ denotes the transpose of the difference vector, $\det(\sum)$ is the determinant of the covariance matrix and exp() is the exponential function.

The main difference to the univariate case, is that each feature/dimension can be correlerated with the other ones. These correlations are modelled by the covariance matrix, which allows for the density function to be rotated and stretched to model the data as well as possible. Bayes' theorem is a fundamental principle in probability theory that describes how to update probabilities when new evidence becomes available. It is named after the Reverend Thomas Bayes, who introduced the concept in the 18th century. Bayes' theorem is particularly useful in Bayesian statistics and machine learning. The theorem is expressed mathematically as follows

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.3}$$

where $P(A)$ is prior, $P(B|A)$ is the likelihood and $P(A|B)$ is the posterior. Bayes' theorem is often used in the context of hypothesis testing or updating beliefs based on new evidence. The theorem allows us to calculate the probability of a hypothesis A given the observed evidence B by combining prior knowledge $P(A)$ with the likelihood of the evidence given the hypothesis $P(B|A)$ and normalizing by the probability of the evidence $P(B)$.

In the context of machine learning, Bayes' theorem is a key component of Bayesian statistics and Bayesian inference, where it is used to update probabilities and make predictions based on observed data. It plays a crucial role in Bayesian reasoning and decision-making processes. In the case of continuous random variables, the probabilities are replaced by probability density functions, and the summation becomes integration. The formula becomes

$$f_{A|B}(a|b) = \frac{f_{B|A}(b|a)f_A(a)}{f_B(b)} \tag{2.4}$$

where $f_{A|B}(a|b)$ is the conditional probability density function of A given B, $f_{B|A}(b|a)$ is the conditional probability density function of B given A, $f_A(a)$ is the marginal probability density function of A and $f_B(b)$ is the marginal probability density function of B [5].

In Bayesian statistics, we often have a prior distribution $P(\theta)$, representing our beliefs about a parameter θ before observing data. Bayes' theorem is then used to update this prior distribution based on new data. That is

$$P(\theta|data) \sim P(data|\theta) \cdot P(\theta) \tag{2.5}$$

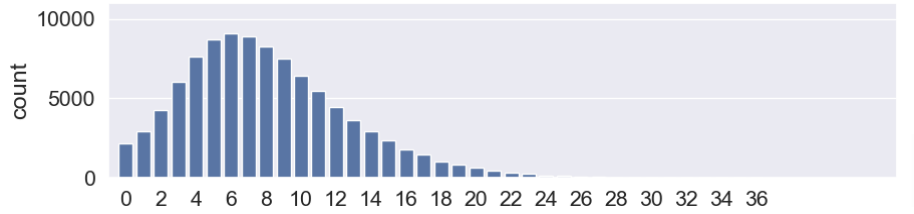
where $P(\theta|data)$ is the posterior distribution, representing our updated beliefs about θ after observing the data, $P(data|\theta)$ is the likelihood, representing the probability of observing the data given a particular value of θ and $P(\theta)$ is the prior distribution, representing our beliefs about θ before observing the data. The posterior distribution is proportional to the product of the likelihood and the prior, and the normalization constant is determined to make the posterior a valid probability distribution.

2.2 Dataset Analysis

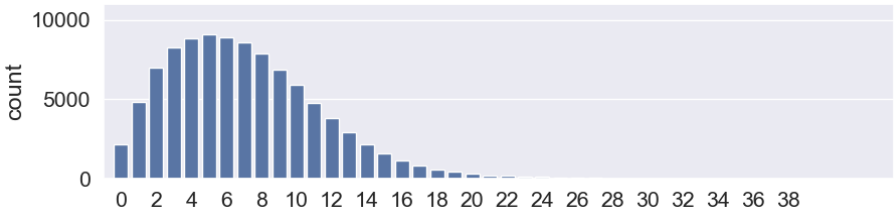
The data set is from one of Kaggle playground prediction competitions. It is the first unsupervised clustering challenge in Kaggle. The dataset is made up of 29 features and 98,000 data points. In this dataset, each row belongs to a particular cluster. Our job is to predict the cluster each row belongs to. We are not given any training data, and we are not told how many clusters there are in the ground truth labels.

Analyzing a dataset allows us to assess the quality of the data. This includes checking for missing values, inconsistencies, outliers, and errors. Addressing these issues early on can prevent biased or inaccurate models. Analyzing the distribution of features and labels helps in understanding the characteristics of the dataset. This information can guide preprocessing steps such as normalization or feature scaling.

In our project, there are seven discrete features: f_07 to f_13. All the values are non-negative. And the distribution of these features are kind of similar, maybe Gaussian or Poisson. There are 22 continuous features: f_00 to f_06 and f_14 to f_28. Distribution are more likely to be Normal, usually with mean 0 and standard deviation 1. The values typically lie between -5 and +5. Fig.2.1 are two figures of distribution of two discrete features.



(a) Feature: 010



(b) Feature: 012

Figure 2.1: Discrete Feature Distribution

Dataset analysis before training is essential for ensuring the quality, fairness, and effectiveness of machine learning models. It provides insights that inform preprocessing steps, model selection, and evaluation strategies, ultimately leading to better performance and more reliable predictions.

2.2.1 Shapiro-Wilk Test

This test is used to test whether a dataset is distributed normally or not. It is one of the most powerful tests for normality. The null hypothesis of the test is that the data are normally distributed. If the W resulting from the test is less than some chosen alpha level (typically 0.05), then the null hypothesis is rejected, indicating that the data do not follow a normal distribution.

This test was published in 1965 by Samuel Shapiro and Martin Wilk and is considered to be one of the most powerful tests for normality testing. Mathematically, it is defined as

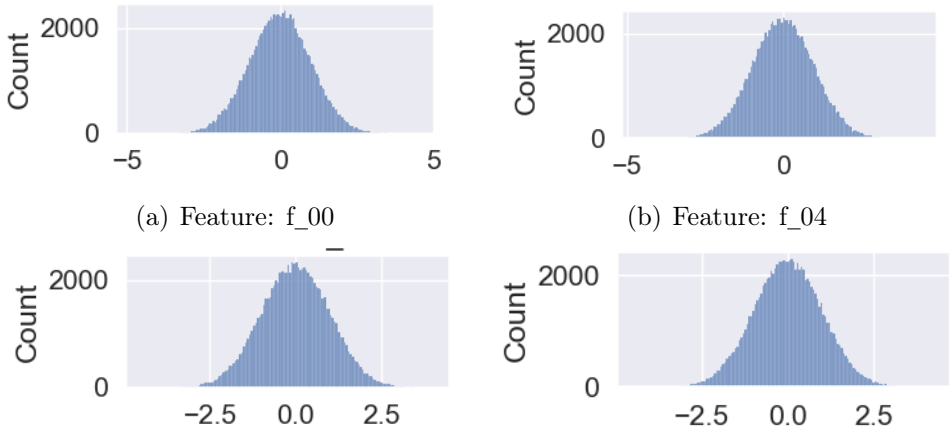
$$W = \frac{(\sum_{i=1}^{n} a_i x_i)^2}{\sum_{i=1}^{n} (x_i - x)^2} \tag{2.6}$$

where x_i is the number from the i-th data point, x is the sample mean and n is the

number of data point in the sample. Variables a_i are calculated via

$$(a_1, ..., a_n) = \frac{m^T V^{-1}}{(m^T V^{-1} V^{-1} m)^{0.5}} \tag{2.7}$$

where $m_1, ..., m_n$ are the mean values of the ordered statistic, of independent and identically distributed random variables, sampled from normal distributions and V denotes the covariance matrix of that order statistic [7]. Fig.2.2 shows four features' distributions. As we can see they look very similar to be normally distributed. And from Shapiro-Wilk test, they are also accepted to be normal distributions.



(a) Feature: f_00

(b) Feature: f_04

(c) Feature: f_15

(d) Feature: f_19

Figure 2.2: Continuous Feature Distribution

The Shapiro-Wilk test statistic (denoted as W) is calculated based on the observed data values and certain coefficients derived from the ordered sample values [12]. The calculated test statistic is compared to the critical value from the Shapiro-Wilk distribution, which depends on the sample size and the chosen significance level.

It is important to note that the Shapiro-Wilk test is sensitive to sample size. For small sample sizes, it has good power to detect departures from normality. However, for larger sample sizes, even minor deviations from normality can lead to rejection of the null hypothesis. In the test, all 29 features have 98,000 data points and hence setting alpha level to be 0.05 for all features would be rational for testing.

Feature: f_00 to Feature: f_07 Hypothesis: Accepted

Feature: f_08
to
Feature: f_13
Hypothesis: Rejected
Feature: f_14
to
Feature: f_21
Hypothesis: Accepted
Feature: f_22
to
Feature: f_28
Hypothesis: Rejected

Above is the results of Shapiro-Wilk test for 29 features of the dataset. The W of f_00 to f_07 and f_14 to f_21 are all bigger than 0.05 and the W of rest features are less than 0.05. About half of features can be treated as normal distributions from our test.

2.2.2 Poisson Dispersion Test

The Poisson dispersion test is a statistical test used to assess whether the variance of count data is equal to its mean, which is a key assumption of the Poisson distribution. In other words, it determines whether the observed variance of count data is consistent with what would be expected under a Poisson distribution.

The null hypothesis of the test is that the variance of the count data is equal to its mean, as predicted by a Poisson distribution. This is the most common test used for verifying a Poisson distribution. The mathematical function is defined as

$$D = \sum_{i=1}^{n} \frac{(x_i - x)^2}{x} \tag{2.8}$$

where x_i is the number from the i-th sample point and x is the sample mean. Note that the expected value of this statistic is

$$E(D) = \frac{(n-1)Var(X)}{E(X)} = \frac{(n-1)\lambda}{\lambda} = n - 1 \tag{2.9}$$

where λ is the rate of the Poisson distribution and X is the variable of the Poisson distribution [8]. Because Poisson's mean and variance are same (both of them are λ), we can delete them together. So the expected value is too far away from the value of $n - 1$, we could just reject the null hypothesis.

More formally, D has a chi-squared distribution with $n - 1$ degrees of freedom under the null hypothesis. We determine the critical values by using a two-tailed test

with significance level. We set the significance level α to be 0.05. From the value of α, we can get the lower band and upper band of the two-tailed test as lower=$\alpha/2$ and upper=$(1-\alpha)/2$. Then using these three values (lower band, upper band and $n-1$) with percentile point function, we can get the critical range. If D of a feature is not in the range, we could reject the hypothesis that this feature is distributed as Poisson.

Feature: f_07
to
Feature: f_13
Hypothesis: Rejected

From the histogram figures of 29 feature distribution, f_07 to f_13 are similar with Poisson distribution. So we would do Poisson dispersion test for these seven features. The result is above. All these seven features are rejected by the test. So there is no Poisson distribution feature in the dataset.

2.2.3 Q-Q plots

Q-Q plots, that is quantile-quantile plots, are used to visually compare how similar two distributions are to each other. They consist of plotting the quantiles (i.e., regular intervals) of the observed distribution against the quantiles of the theoretical distribution [14]. The closer the Q-Q plots are to forming a straight line, the more confident you can be that the observed and theoretical distributions are the same. In this project, we will use normal Q-Q plots to verify if the 29 features are normal distributions. Different colors of lines represent different theoretical distributions being compared to the empirical data distribution. Blue line represents the theoretical normal distribution while red line represents the distribution of data in feature. Most of features in our dataset appear very good positive correlation. For example the Q-Q plot of feature 1 is shown in Fig.2.3.

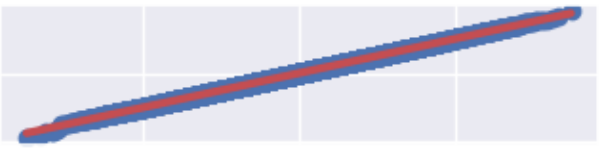


Figure 2.3: Feature 1 Normal Q-Q chart

Even though the features f_22 to f_28 failed the Shapiro-Wilk test, they still

appear to be quite close to being normally distributed. This behaviour could be because these features are made up of a mixture of normal distributions. There is not an easy way to verify this however. At the mean time some features (f_08 to f_13) fail the normal test because the two lines of these features fail matching each other very well. Fig2.4 is the Q-Q chart of feature 10.

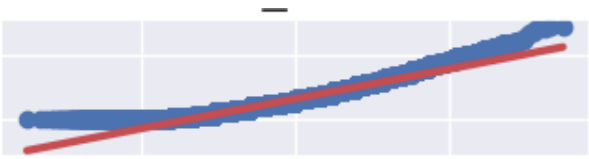


Figure 2.4: Feature 10 Normal Q-Q chart

From the three tests, we have a simple understanding of this dataset ,that is, for the 29 features, most of them can be treated as normal distributions. Features f_00 to f_07 and f_14 to f_21 could be made up by one normal distribution while f_22 to f_28 could be seen made up by multiple normal distributions. Features f_08 to f_13 failed both Shapiro-Wilk test and Q-Q normal plot test so they can not be seen as normal distributions and they failed Poisson dispersion test so they can not be treated as Poisson distributions neither.

Chapter 3

Unsupervised Clustering for the Dataset

In this project, we use five different unsupervised learning algorithms to be applied on the dataset: K-means, Gaussian mixture model and Bayesian Gaussian mixture model, Gaussian mixture model, conjunction of GMM with LightGBM and conjunction of BGMM with LightGBM. Our goal is to find out which algorithm is most suitable for our dataset.

K-means, Gaussian mixture models, and LightGBM are all techniques used in machine learning and data analysis, but they have several differences and have distinct characteristics. K-means is primarily used for partitioning data into a predetermined number of clusters based on minimizing the within-cluster sum of squares. It is simple and fast. GMM assumes that the data is generated from a mixture of several Gaussian distributions. It models the data as a weighted sum of Gaussian distributions and is more flexible than K-means, as it can capture elliptical clusters and accommodate different cluster shapes. GMM is often used when the data is believed to be generated from multiple underlying distributions or when the assumption of spherical clusters in K-means is not appropriate. LightGBM, is a gradient boosting framework that is designed for distributed and efficient training of machine learning models.

3.1 K-means

K-means is an unsupervised machine learning algorithm used to find groups of observations. This algorithm follows an easy or simple way to classify a given data set

through a predetermined number of clusters. It works iteratively to assign each data point to one of K groups based on the features that are provided. And the data points are clustered based on feature similarity.

3.1.1 How K-means works

K-means is based on centroid-based clustering. A centroid is a data point at the centre of a cluster. In centroid-based clustering, clusters are represented by a centroid. It is an iterative algorithm in which the notion of similarity is derived by how close a data point is to the centroid of the cluster. K-means clustering works as follows. The K-means clusering algorithm uses an iterative procedure to deliver a final result. The algorithm requires the number of clusters K and the data set as input. The data set is a collection of features for each data point. The algorithm starts with initial estimates for the K centroids [1]. Normally, the initial centroids will be decided randomly. The algorithm then iterates between two steps.

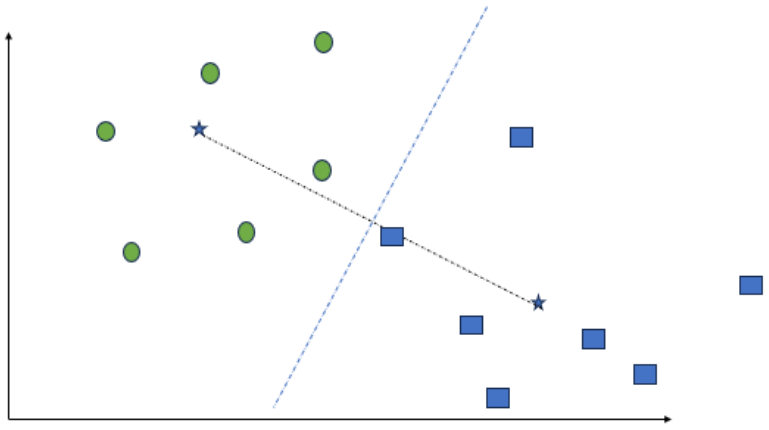


Figure 3.1: Clusters' Centroid

1) Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, which is based on the squared Euclidean distance. So, if one set is the collection of centroids, then each data point is assigned to a cluster based on minimum Euclidean distance.

2) In this step, the centroids are recomputed and updated. This is done by taking the mean of all data points assigned to that centroid's cluster.

The algorithm then iterates between step 1 and step 2 until a stopping criteria is met. Stopping criteria means no data points change the clusters, the sum of the distances is minimized or some maximum number of iterations is reached. This algorithm is guaranteed to converge to a result. Some experimental analysis divulged that there is no universal solution for the problems of the K-means algorithm [1]. The result may be a local optimum meaning that executing more than one run of the algorithm with randomized starting centroids may give a better outcome.

3.1.2 Implementation of K-means

In this project, we use K-means class which is in sklearn package to complete kmeans model. This class is a part of the 'sklearn.cluster' module. K-means requires the number of clusters to be specified, and it initializes cluster centroids randomly by default. We can also specify inital centroids if needed.

The algorithm works by iteratively assigning each data point to the nearest centroid and then updating the centroids based on the mean of the points assigned to each cluster. The K-means class in sklearn allows us to specify various parameters such as the number of clusters, initialization method, maximum number of iterations, and more.

```
# sklearn import
from sklearn.datasets import make_classification
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

```
kmeans = KMeans(n_clusters=8,
                init='random',
                random_state=SEED)
kmeans.fit(df_model)
labels = kmeans.labels_
```

Figure 3.2: Code of using KMeans class

3.2 Gaussian Mixture Model and Bayesian Gaussian Mixture Model

A Gaussian mixture is simply a combination (or mixture) of Gaussian distributions. It is a common distribution that appears in clustering problems where the data is sampled randomly from several groups that each follow a normal distribution. Gaussian mixture model and Bayesian Gaussian mixture model are both Gaussian mixture models used for clustering tasks, but they differ in some aspects. In GMM, the parameters are estimated using the expectation-maximization algorithm while in BGMM, Bayesian methods are used for parameter estimation. Bayesian methods involve the use of Bayesian inference techniques such as variational inference.

3.2.1 GMM

In a GMM, it is assumed that the data is generated by a combination of several Gaussian distributions, each characterized by its mean and covariance. GMM parameters are estimated from training data using the iterative expectation-maximization algorithm or maximum a posteriori (MAP) estimation from a well-trained prior model [19]. The model parameters include the weights of each Gaussian component, representing the contribution of each component to the overall distribution. Additionally, GMMs have parameters for the mean and covariance of each Gaussian component.

The probability density function of a GMM for a given data point is a weighted sum of the PDFs of its individual Gaussian components. Mathematically, the probability density function of a GMM can be expressed as

$$f(x) = \sum_{i=1}^{K} \pi_i \cdot \mathcal{N}(x|\mu_i, \Sigma_i) \tag{3.1}$$

where $f(x)$ is the probability density of the data point x, K is the number of Gaussian components in the mixture, π_i is the weight of the i-th Gaussian component and $\mathcal{N}(x|\mu_i, \sum_i)$ is the Gaussian distribution function with mean μ_i and covariance matrix $\sum_i$ [13].

Then each Gaussian is responsible that a single data spot is placed where it is. One Gaussian may be more responsible than the others but they are all working in a mixture to explain the data. During learning all of these ellipsoid clusters are moving in space and varying their shape trying to match perfectly the distributed

data $p(x)$. The learning procedure for models with latent variables is the expectation maximization. This algorithm works in two consecutive steps that are repeated until all Gaussians only have found their place and would only show slight changes in further steps.

Why we use EM algorithm rather than the other algorithms? Because the expectation-maximization algorithm offers several advantages, making it a widely used method in various statistical and machine learning applications. EM has good robustness. EM is particularly useful when dealing with incomplete or missing data. It is designed to estimate the parameters of a model even when some data points have missing values. EM is an unsupervised learning method. EM is often applied to unsupervised learning problems, such as clustering and density estimation. It does not require labeled data and can be used to discover hidden patterns in the data. EM is well-suited for problems involving mixture models, such as Gaussian mixture models. It can effectively estimate the parameters of multiple distributions contributing to the overall data distribution. Unlike hard clustering algorithms that assign data points to a specific cluster, EM provides soft assignments. It assigns probabilities to data points belonging to different clusters, allowing for a more nuanced representation of uncertainty in the data.

EM is known to converge to a local maximum of the likelihood function, ensuring that the estimated parameters are relatively stable and consistent. EM is a versatile algorithm that can be applied to a wide range of models and distributions. It is not limited to specific types of probability distributions and can be adapted for different applications. EM exhibits guaranteed monotonic improvement in likelihood during each iteration, which contributes to its stability and reliability. However, it may converge to a local maximum, and the choice of initial parameters can influence the results. EM can be employed for complex models, where closed-form solutions for maximum likelihood estimation are not readily available. It is particularly useful in situations where direct optimization is computationally infeasible.

The expectation-maximization algorithm consists of two main steps: E-step and M-step. In the E-step, the algorithm computes the expected value of the missing data or latent variables while in the M-step, the algorithm updates the parameters of the model to maximize the likelihood of the observed data.

E-Step (expectation):

At the start of the model, Gaussians are placed and shaped randomly (or pre-trained by K-means locations). In this step, we need to calculate the expected values

of responsibilities given the current model parameters. For each Gaussian, that represents one cluster, the model calculates how responsible the cluster z_k is for one data spot x_n as

$$\gamma_{nk} = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \sum_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \sum_j)} \tag{3.2}$$

where $\mathcal{N}$ is normal distribution. These responsibilities sum up to 1 over all clusters for one data spot. Here we can read out the winner at the end of the learning procedure that is assigned as the predicted cluster for that data spot x_n.

M-Step (maximization):

During the maximization step, the expected values of the latent variables (calculated in the E-step) are used and the parameters of the model are updated to maximize the likelihood of the observed data given the current estimates of the latent variables. In GMM, to maximize the likelihood, the locations given by the cluster center and the shapes covered by the covariance matrices are updated as

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} x_n \tag{3.3}$$

$$\sum\nolimits_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} (x_n - \mu_k)(x_n - \mu_k)^T \tag{3.4}$$

where N is the number of data points in the dataset and N_k is the number of data points whose γ_{nk} is not zero.

We can see that we obtain each cluster location μ_k by looking and summing over all data spots. This is a big difference between K-means and GMM as the former only calculates the cluster centers using cluster members. But in GMM, as already told, each Gaussian is responsibile for all data points. Hence the new location is calculated by using them all, but weighting with the responsibility. In this way points that could be well explained by this cluster contribute much more to the new location (mean) than the other data points. The same holds for the cluster shapes, e.g., covariances.

3.2.2 BGMM

In Bayesian Gaussian mixture models, compared to EM in GMM, variational inference is the technique used to approximate complex probabilistic models. To find

the point estimates of the latent variables which account for uncertainty, an entire posterior distribution over the latent variables and parameters is approximated. For approximation of posterior distribution which involves intractable integrals variational inference methods are applied for making the calculation tractable.

In Bayesian Gaussian mixture model, we assume that the observed data $X = \{x_1, x_2, ..., x_n\}$ are generated from a mixture of K Gaussian distributions. The latent variables include the cluster assignments $Z = \{z_1, z_2, ..., z_n\}$, where z_i denotes the cluster assignment of each data point, and parameters of the Gaussian distributions involves mean μ_k, precision matrix Λ_k and mixing coefficients π_k. Our goal is to infer the posterior distribution over the parameters and the latent variables. First, we need to define some prior on the unknown parameters for mixing coefficients, mean vectors, precision matrix and cluster assignment for each of the observation present in the data

$$\pi \sim \text{SymDir}(K, \alpha_0) \tag{3.5}$$

$$\Lambda_{i=1...K} \sim \mathcal{W}(W_0, v_0) \tag{3.6}$$

$$\mu_{i=1...K} \sim \mathcal{N}(\mu_0, (\beta\Lambda_i)^{-1}) \tag{3.7}$$

$$Z \sim \text{Mult}(1, \pi) \tag{3.8}$$

$$X \sim \mathcal{N}(\mu_{z_i}, \Lambda_{z_i}^{-1}) \tag{3.9}$$

where SymDir is the symmetric Dirichlet distribution as prior for mixing coefficients with hyper-parameter α_0, $\mathcal{W}$ is Wishart distribution as prior for precision matrix with hyper-parameter (W_0, v_0), Mult is multivariate normal distribution with hyper-parameter (μ_0, β_0) [17]. The joint probability distribution of the variables can be written as

$$p(X, Z, \pi, \mu, \Lambda) = p(X|Z, \mu, \Lambda)p(Z|\pi)p(\pi)p(\mu|\Lambda)p(\Lambda) \tag{3.10}$$

The joint probability distribution can be divided into individual distributions given by

$$p(X|Z, \mu, \Lambda) = \prod_{n=1}^{N}\prod_{k=1}^{K}\mathcal{N}(x_n|\mu_k, \Lambda_k^{-1})^{Z_{nk}} \tag{3.11}$$

$$p(Z|\pi) = \prod_{n=1}^{N}\prod_{k=1}^{K}\pi_k^{Z_{nk}} \tag{3.12}$$

$$p(\pi) = \frac{\Gamma(K\alpha_0)}{\Gamma(\alpha_0)^K} \prod_{k=1}^{K} \pi_k^{\alpha_0 - 1} \tag{3.13}$$

$$p(\mu|\Lambda) = \prod_{k=1}^{K} \mathcal{N}(\mu_k|\mu_0, (\beta_0\Lambda_k)^{-1}) \tag{3.14}$$

$$p(\Lambda) = \prod_{k=1}^{K} \mathcal{W}(\Lambda_k|W_0, v_0) \tag{3.15}$$

The third equation above is using Dirichlet distribution to get the prior probability distribution of π. Dirichlet distribution is a multivariate probability distribution that represents the distribution of probabilities for outcomes in a multinomial distribution. The mixing proportions, π_k, are given a symmetric Dirichlet distribution prior with parameter α_0/K

$$p(\pi_1, ..., \pi_k|\alpha_0) \sim \text{Dirichlet}(\alpha_0/K, ..., \alpha_0/K) \tag{3.16}$$

The above family distributions is called the variational distribution. Our goal is to find the best variational distribution that approximates the true posterior $p(\theta|X)$, where θ means all the model parameters, by minimizing the Kullback-Leibler (KL)

$$\lambda^* = \arg\min_\lambda \text{KL}(q(\theta|\lambda)||p(\theta|X)) \tag{3.17}$$

Then we can use θ to find the number of clusters and each Gaussian distribution's mean and variance.

3.3 Light Gradient-Boosting Machine

LightGBM, short for light gradient-boosting machine, is a free and open-source distributed gradient-boosting framework for machine learning, originally developed by Microsoft. It is based on decision tree algorithms and used for ranking, classification and other machine learning tasks. The development focus is on performance and scalability. In our project, this algorithm is applied after BGM classification as it is a supervised classification algorithm.

The size of dataset is increasing rapidly. It becomes very difficult for traditional data science algorithms to give accurate results. LightGBM is prefixed as light because of its high speed. LightGBM can handle the large size of data and takes less

memory to run. LightGBM is a relatively new algorithm and lightGBM focuses on accuracy of results. It also supports GPU learning and thus data scienctists are widely using GBM for data science application development. However, it also has some disadvantages, for example lightGBM is sensitive to overfitting and can easily overfit small data. Gradient boosting is a machine learning technique used for regression and classification problems. It builds an ensemble of weak learners sequentially, with each new learner correcting errors made by the existing ensemble. LightGBM is an advanced GBDT algorithm, which contains two techniques: gradient-based one-side sampling and exclusive feature bundling to deal with large number of data instances and large number of features respectively [9].

3.3.1 LightGBM intuition

LightGBM grows tree vertically while other tree based learning algorithms grow trees horizontally. It means that LightGBM grows tree leaf-wise while other algorithms grow level-wise. It will choose the leaf with max delta loss which is the change in the loss function that occurs when a particular leaf node is split further during the tree-growing process to grow. When growing the same leaf, leaf-wise algorithm can reduce more loss than a level-wise algorithm. We need to understand the distinction between leaf-wise tree growth and level-wise tree growth.

Leaf-wise tree growth

Leaf-wise tree growth is a strategy used in some gradient boosting algorithms. Traditionally, decision trees are grown level-wise, meaning that nodes at each level of the tree are expanded before moving to the next level. In contrast, leaf-wise tree growth expands the node that reduces the loss the most, regardless of its depth in the tree.

In leaf-wise tree growth, instead of splitting nodes level by level, the algorithm selects the node with the largest gain in terms of reducing the loss function (e.g., mean squared error for regression, log loss for classification) [3]. The algorithm considers all possible split points for each feature and chooses the one that results in the greatest reduction in the loss function. It can be explained with the illustration in Fig. 3.3.

Level-wise tree growth

Level-wise tree growth, also known as depth-wise tree growth or breadth-first tree growth, is a traditional strategy used in decision tree algorithms for growing trees. In level-wise growth, nodes at each level of the tree are expanded before moving to the

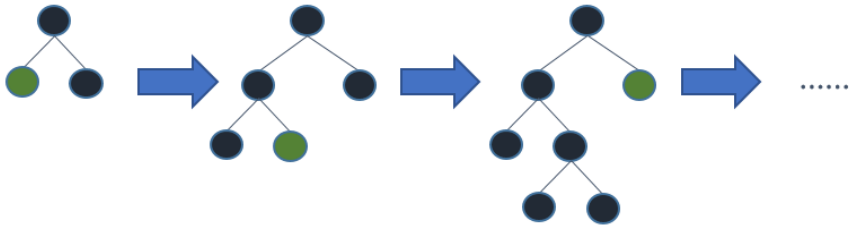


Figure 3.3: Leaf-wise tree growth

next level. This means that the tree is grown horizontally, with all nodes at the same level being expanded simultaneously. Most decision tree learning algorithm grow tree by this method.

In level-wise tree growth, all nodes at a particular level of the tree are expanded before any nodes at the next level are considered. At each level, the algorithm evaluates all possible split points for each feature and selects the best split point based on a criterion such as maximizing information gain (for decision trees) or minimizing impurity (for classification trees). It can be explained with the illustration in Fig. 3.4.

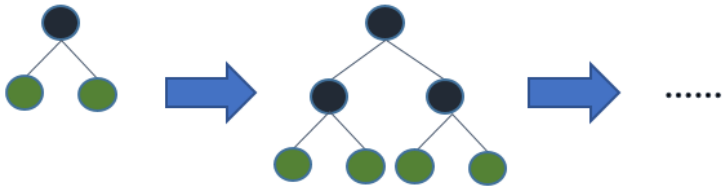


Figure 3.4: Level-wise tree growth

In the context of gradient boosting, decision trees serve as 'verdict trees', approximating negative gradients of the loss function with respect to the model's output. These trees are constructed iteratively using a training set comprising instances $x_1, x_2, ..., x_n$. We assume all the negative gradients of a loss function with respect towards the output model are $g_1, g_2, ..., g_n$. The decision tree nodes are split based on features to maximize variance reduction, aiming to capture the most revealing features leading to the largest evidence gain [18]. The model formula involves a combination of a base model and the contributions from sequentially added decision trees, each scaled by a learning rate. Variance gain, a measure of data improvement

through node splitting, quantifies the reduction in variance achieved by each split. The variance gain of dividing measure j at a point d for a node is defined as

$$V_j(d) = \frac{1}{n}\left(\frac{(\sum_{x_{ij}<d} g_i)^2}{n_l^j(d)} + \frac{(\sum_{x_{ij}>d} g_i)^2}{n_r^j(d)}\right) \tag{3.18}$$

where n is the number of data points in the dataset, $n_l^j(d)$ is the number of data points that satisfy $x_{ij} < d$ and $n_r^j(d)$ is the number of data points that satisfy $x_{ij} > d$ [10]. This process facilitates the transformation of the input space X towards the gradient space G, ultimately enhancing the model's predictive performance.

3.3.2 Implementation of LightGBM

In this classification method, we combine two classification models. One is unsupervised learning: Bayesian Gaussian mixture; another is supervised learning: LightGBM. Because LightGBM is supervised learning so we need an way to label our dataset by BGMM then we can apply LightGBM to get a more advanced model. We apply LightGBM algorithm on the result of BGMM applied on the dataset. Here is a basic example of how to use Bayesian Gaussian mixture in scikit-learn:

```
from sklearn.mixture import BayesianGaussianMixture
import numpy as np

# Generate some sample data
X = np.random.randn(100, 2)

# Create a BayesianGaussianMixture instance
bgm = BayesianGaussianMixture(n_components=10)

# Fit the model to the data
bgm.fit(X)

# Predict the cluster labels
labels = bgm.predict(X)

# Get the estimated number of components
n_components = bgm.n_components_
```

Figure 3.5: Framework of BGMM

At first, fit a Bayesian Gaussian mixture model to the dataset. Predict cluster labels and compute the maximum probability score for each data point based on the

predicted cluster probabilities. Select data points with a predicted probability score above a certain threshold (0.7) for LightBGM training. 'BayesianGaussianMiture' is a class in scikit-learn's 'sklearn.mixture' module, which implements a Bayesian version of Gaussian mixture models.

The key difference between 'BayesianGaussianMixture' and the traditional 'GaussianMixture' is that the former incorporates a Bayesian framework for model selection. Bayesian Gaussian mixture model typically involves criterion based on model selection techniques like Bayesian information criterion or Akaike information criterion. So unlike Gaussian mixture, you do not need to specify the number of components explicitly when using Bayesian Gaussian mixture. Instead, the algorithm infers the optimal number of components from the data.

LightGBM classifier is one of the most successful machine learning algorithms, which has fast, distributed, high performace gradient boosting framework based on decision tree algorithm. It is used for ranking, classification and many other machine learning tasks.

Chapter 4

Experiments

4.1 Programming Environment

This project is implemented on python and matlab. Python has a powerful function library that can help us implement our model more conveniently. Matlab is a high-level programming language and environment that is widely used in various fields. It is easy to use and provides the flexibility to solve problems at any level of details.

PyCharm is an integrated development environment (IDE) for Python. PyCharm is developed by JetBrains and provides features such as code analysis, a graphical debugger, an integrated unit tester, integration with version control systems, and support for web development with Django or Flask, among others. PyCharm is widely used by Python developers for its robust set of tools and features that streamline the development process.

4.2 Feature Engineering

Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, improving model performance. The choice of feature engineering techniques can vary based on the characteristics of the dataset and the specific requirements of different classification methods.

Now, we introduce how to do feature engineering before training model of these three classification methods in turn. The feature engineering of three projects are different because different classification methods have different requirements and assumptions about the data.

When we apply kmeans classification method on our dataset, we need to performe exploratory data analysis (EDA) and data engineering on this dataframe first. The code will iterate through each column in the dataframe and prints various statistics and information about each column.

Loading the Data:

- The first line of code loads the data from the CSV file into a pandas dataframe called `df`.

Printing DataFrame Information:

- `print(df.shape)`: This prints the shape of the dataframe, which gives the number of rows and columns in the dataset.
- `df.head()`: This prints the first few rows of the dataframe, giving a glimpse of what the data looks like.

```
if df[c].dtype == int or df[c].dtype == float:
    s = "- Statistics:\n"

    me = np.round(df[c].mean(), 2)
    st = np.round(df[c].std(), 2)
    s += f"-- Mean (std): {me} ({st})\n"

    q1 = np.round(df[c].quantile(0.25), 2)
    q2 = np.round(df[c].quantile(0.5), 2)
    q3 = np.round(df[c].quantile(0.75), 2)
    s += f"-- Quantiles: q1={q1}, q2={q2}, q3={q3}\n"
    s += f"-- Min {df[c].min()}\n"
    s += f"-- Max {df[c].max()}"
```

Figure 4.1: Code of Getting Statistics information

Iterating Through Columns:

- The code iterates through each column in the dataframe.
- For each column (`c`), it prints various pieces of information about that column:
  - Index and name of the column.
  - Number of unique elements in the column.
  - Sample of unique elements (up to the first 20).

  - Data type of the column.
  - Number of missing values in the column and the percentage of missing values relative to the total number of rows.

Additional Statistics for Numeric Columns:

- If the column's data type is either integer or float, it calculates and prints additional statistics:
  - Mean and standard deviation.
  - Quantiles (25th, 50th, and 75th percentiles).
  - Minimum and maximum values.

Formatting Output:

- The output is formatted neatly using string formatting techniques (using f-strings).

Overall, this code provides a detailed overview of the dataset, including basic statistics and information about each column, which is crucial for understanding the data and making informed decisions during the data analysis process. Because K-means is a pretty high-intelligence classification function, we do not have to do too much work on data engineering.

For GMM preprocessing, the data is scaled by using a power transformation, setting column names, and dropping specific columns before using it as input for machine learning models. Here is the detailed process.

- **Scaling the Data:**
  - `PowerTransformer().fit.transform(data)`: This assigns the column names of the original data (`data`) to the scaled data (`scaled_data`). This ensures that the scaled dataframe has the same column names as the original dataframe.
  - `pd.DataFrame(...)`: This part creates a dataframe from the scaled data.
- **Setting Column Names:**

  - `scaled_data.columns = data.columns`: This assigns the column names of the original data (`data`) to the scaled data (`scaled_data`). This ensures that the scaled dataframe has the same column names as the original dataframe.

- **Dropping Features:**

  - `drop_feats`: Some features in the dataset may contain noise or irrelevant information that can negatively impact the performance of the model. Removing these noisy features can help improve the quality of the learned patterns. We use feature importance scores to identify features that contribute the most to the predictive power of the model. Features with low importance scores will be removed. This variable contains a list of column names to be dropped from the dataset. These columns are identified by their names, such as 'f_01', 'f_02', ..., 'f_06', and 'f_14', 'f_15', ..., 'f_21'.
  - `X = scaled_data.drop(drop_feats, axis=1).values`: This line drops the columns specified in `drop_feats` from the scaled dataframe (`scaled_data`). It drops these columns along the columns axis (`axis=1`) and assigns the resulting dataframe to `X`. Finally, `.values` converts the dataframe into a NumPy array, which is commonly used as input for machine learning models. In addition, reducing the number of features increases comprehensibility and ameliorates the problem that some unsupervised learning algorithms break down with high dimensional data [4].

Table 1:Part of Original Dataset

Dataset	feature 1	feature 2	feature 3
data point 1	-0.389419	-0.912791	0.648951
data point 2	-0.689249	-0.4539535	0.6541745
data point 3	0.809078	0.324567	-1.170602

Table 2:Part of Preprocessed Dataset

Dataset	feature 1	feature 2	feature 3
data point 1	-0.389230	-0.917652	0.647948
data point 2	-0.688368	-0.458647	0.653182
data point 3	0.805709	0.319397	-1.166935

The two tables above show how the values in dataset change after preprocessing. (The first table is the first three values of 'f_01','f_02','f_03'; and the second table is that nine values after preprocessing.)

In our LightGBM classification project, except scaling the columns by using PowerTransformer, it also applies the interquartile range (IQR) method to identify and handle outliers in the float columns in preprocessing.

The interquartile range method is a statistical technique used to identify and handle outliers in a dataset. It is based on the quartiles of a distribution, specifically the first quartile (Q1) and the third quartile (Q3). First, The dataset is divided into four equal parts, with Q1 representing the value below which 25 % of the data fall, and Q3 representing the value below which 75 % of the data fall. The IQR is calculated as the difference between the third quartile (Q3) and the first quartile (Q1). It represents the spread or variability of the middle 50 % of the data. Any data points that fall below $Q1 - 2 * IQR$ or above $Q3 + 2 * IQR$ are considered outliers. Outliers can be handled in various ways, such as removing them from the dataset, replacing them with a central value (e.g., mean or median), or capping them to a certain threshold. This method is commonly used in exploratory data analysis and data preprocessing to clean datasets before further analysis or modeling.

```
def iqr_outliers(df,col_list):
   for col in col_list:
      q1 = df[col].quantile(0.25)
      q3 = df[col].quantile(0.75)
      iqr = q3-q1
      Lower_tail = q1 - 2 * iqr
      Upper_tail = q3 + 2 * iqr
      df.loc[df[col] > Upper_tail,col ]=Upper_tail
      df.loc[df[col] < Lower_tail,col]=Lower_tail
   return df

train=iqr_outliers(train,float_cols)
```

Figure 4.2: Code of IQR method

4.3 Classification Implementation

In this section, we introduce the implementation of the classification methods. As we have showed K-means' implementation in last chapter, we now describe how GMM and LightGBM with BGMM work. For GMM classification, we wrote a class to implement the expectation-maximization algorithm for fitting a Gaussian mixture model to a given dataset. At first, we need to determine the number of cluster. For GMM algorithm, the model can not automatically converge into certain number of clusters. In this project, all the cluster number will be tested from 2 until it presents a parabolic curve.

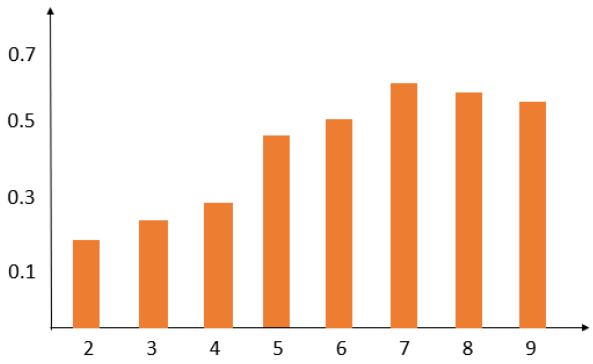


Figure 4.3: Rand measure score of different cluster number in GMM

We can see that in GMM when we set cluster number to 7, we can get the best performance for classification result. The similar test for the other four algorithms shows the same results that 7 is the best cluster number for our problem. The classification is divided into six parts: initialization, expectation(E-step), maximization(M-step), fitting the model, prediction. The core part of the class is E-M algorithm.

1. **Expectation Step**
   In the E-step method, the responsibilities of each data point for each Gaussian component are updated based on the current parameters. This step calculates the probabilities of each data point belonging to each Gaussian component using the current means and covariances. The responsibilities are stored in the '**responsibilities**' attribute. The code is shown below

   self.responsibilities = self.predict_ proba(X)

2. **Maximization Step**
   In the M-step method, the parameters of the GMM (mixing coefficients, means, and covariances) are updated based on the current responsibilities. This step updates the mixing coefficients ('**pi**'), means ('**mu**'), and covariances ('**sigma**') using the current responsibilities and data. The code is shown below

   self.pi = self.responsibilities.mean(axis=0)
   for j in range(self.k):
   { r_column = self.responsibilities[:, j]
   total_responsibility = r_column.sum()
   self.mu[j] = (X * r_column[:, np.newaxis]).sum(axis=0) / total_responsibility
   self.sigma[j] = np.cov(X.T, aweights=(r_column / total_responsibility).flatten(),
   bias=True) }

At the end, the probabilities of each data point belonging to each Gaussian component are calculated in the last loop. Here is how this part works.

Initialize Likelihood Matrix

- A matrix likelihood of shape '**(self.n, self.k)**' is initialized to store the likelihoods of each data point belonging to each Gaussian component. Here, self.n is the number of data points, and self.k is the number of Gaussian components.

Compute Likelihoods for Each Gaussian Component

- For each Gaussian component '**j**': A multivariate normal distribution is instantiated with mean '**self.mu[j]**' and covariance '**self.sigma[j]**'.
- The pdf method of the **multivariate_normal** object is called to compute the probability density function of the data points X under this Gaussian component. These likelihood values are stored in the corresponding column of the '**likelihood**' matrix.

Calculate Numerator of Responsibilities

- The numerator of the responsibilities matrix is computed by element-wise multiplication of the likelihood matrix (likelihood) and the mixing coefficients '**self.pi**'. This represents the unnormalized responsibilities.

Compute Denominator of Responsibilities

- The denominator of the responsibilities matrix is obtained by summing the numerator along each row. This ensures that the responsibilities sum up to 1 for each data point.

Calculate Responsibilities

- The responsibilities matrix is computed by dividing the numerator matrix by the denominator matrix. This yields the normalized responsibilities, indicating the probability of each data point belonging to each Gaussian component.

Return Responsibilities

- The calculated responsibilities matrix is returned as the output of the '**predict_proba**' method.

```
def predict_proba(self, X):
    likelihood = np.zeros((self.n, self.k))
    for j in range(self.k):
        distribution = multivariate_normal(mean=self.mu[j], cov=self.sigma[j])
        likelihood[:, j] = distribution.pdf(X)

    numerator = likelihood * self.pi
    denominator = numerator.sum(axis=1)[:, np.newaxis]
    responsibilities = numerator / denominator
    return responsibilities
```

Figure 4.4: Code of Prediction Part

In our LightGBM classification project, we apply clustering using a Bayesian Gaussian mixture model first like GMM above, then we train a LightGBM classifier on the significant data points identified by the clustering algorithm. Finally, it generates predictions and get the result we want.

First, fit a Bayesian Gaussian mixture model to the dataset. Then, predict cluster labels for each data point using the fitted model and compute the probability score based on the predicted cluster probabilities. Next, select data points with a predicted probability score above a certain threshold (0.7) for training the LightGBM model. These data points are assumed to be more reliable for training the model. Prepares data for training by selecting features '**x**' and target labels '**y**' based on the selected data points. Then apply stratified K-Fold cross-validation for training and validation,

and the number of fold is 10. Stratification ensures that each fold has a similar distribution of target labels. For each fold, it splits the data into training and validation sets. The model is trained on the training set and evaluated on the validation set. Set LightGBM model parameters '**params_lgb**'. For each fold, LightGBM datasets '**tr_dataset**' and '**vl_dataset**' are created and a LightGBM model is trained with early stopping and logging evaluation metrics. Finally, the trained models are stored in a list for later use.

```
gkf = StratifiedKFold(N_FOLDS)
for fold, (train_idx, valid_idx) in enumerate(gkf.split(X, y)):
  tr_dataset = lgb.Dataset(X.iloc[train_idx], y.iloc[train_idx], feature_name=best_cols)
  vl_dataset = lgb.Dataset(X.iloc[valid_idx], y.iloc[valid_idx], feature_name=best_cols)

  model = lgb.train(params=params_lgb,
            train_set=tr_dataset,
            valid_sets=vl_dataset,
            num_boost_round=5000,
            callbacks=[lgb.early_stopping(stopping_rounds=300, verbose=True), lgb.log_evaluation(period=200)])

  model_list.append(model)
```

Figure 4.5: Code of LightGBM Classifier

Training the LightGBM Model

- A list '**model_list**' is initialized to store the trained LightGBM models for each fold.
- A '**StratifiedKFold**' object '**gkf**' is created with '**N_FOLDS**' folds.
- '**train_idx**' and '**valid_idx**' contain the indices of the training and validation sets for the current fold, respectively.
- LightGBM datasets ('**tr_dataset**' for training and '**vl_dataset**' for validation) are created using '**lgb.Dataset**'.
- The LightGBM model is trained using the '**lgb.train**' function:
  - The parameters are specified in '**params=params_lgb**'.
  - Early stopping is applied with '**lgb.early_stopping**' to monitor the validation performance and stop training if it doesn't improve for a specified number of rounds ('**stopping_rounds=300**').

– Evaluation metrics are logged every 200 rounds using '**lgb.log_evaluation**'.

```
for model in model_list:
  lgb_preds+=model.predict(train_scaled[best_cols])

ss=pd.read_csv("sample_submission.csv")
ss['Predicted']=np.argmax(lgb_preds, axis=1)
ss.to_csv("submission.csv",index=False)
```

Figure 4.6: Code of Model Prediction

The predictions from all models are averaged to get the final predictions. The class with the majority vote among individual predictions will be chosen as the final prediction for each data point. Once all the LightGBM models are trained and stored in model_list, predictions are made using these models. For each trained model in the model_list, predictions are generated for all data points in the original dataset using the predict method. This process ensures that predictions from all models are aggregated, which can help improve the overall prediction accuracy and reduce overfitting. After predictions from all models are accumulated, they are combined to obtain the final predictions. Since the task is a classification problem, the final prediction for each data point is determined by selecting the class with the highest predicted probability. This is achieved by using the np.argmax function along the axis of the predicted probabilities.

Chapter 5

Evaluation, Analysis and Comparisons

5.1 Rand Measure

The rand index or rand measure in statistics, and in particular in data clustering, is a measure of the similarity between two data clusterings [15]. In this project, we use this measure to evaluate the outputs. The classification results will be evaluated with the ground truth cluster labels of the data.

5.1.1 Definition

Given a set of n elements $S = o_1, ..., o_n$ and two partitions of S to compare, $X = X_1, ..., X_r$, a partition of S into r subsets, and $Y = Y_1, ..., Y_s$, a partition of S into s subsets, define the following:

- a_i the number of pairs of elements in S that are in the same subset in X and in the same subset in Y.
- b_i the number of pairs of elements in S that are in the different subset in X and in the different subset in Y.
- c_i the number of pairs of elements in S that are in the same subset in X and in the different subset in Y.
- d_i the number of pairs of elements in S that are in the different subset in X and in the same subset in Y.

The rand index, R is given by

$$R = \frac{a+b}{a+b+c+d} = \frac{a+b}{\binom{n}{2}} \tag{5.1}$$

Pair of points belonged to a and b means in these two classification results, the relationship between them is considered identically: a means these pairs of points both belongs to one cluster in two classification result while b means these pairs of points belongs to different cluster. So $a + b$ can be considered as the number of agreements between X and Y and $c + d$ as the number of disagreements between X and Y.

For example, the dataset has five elements: $S = o_1, o_2, o_3, o_4, o_5$ and these elements belong to three clusters: $X = X_1, X_2, X_3$. Our classification result is $S = o_1(X_1), o_2(X_1), o_3(X_1), o_4(X_2), o_5(X_2)$. The truth label is $S = X_1, X_1, X_2, X_2, X_3$. Because there are five elements in the dataset so we have 10 pairs of points totally. Then we can get a, b: $a = 1$,$b = 5$. Using the formula of R, we can get $R = (1+5)/10 = 0.6$. A rand index value close to 1 indicates a high level of agreement between the two clusterings, meaning that they are similar. On the other hand, a rand index value close to 0 suggests little or no agreement between the two clusterings, indicating dissimilarity. A rand index value less than 0 suggests that the agreement between the two clusterings is worse than random chance [16].

The rand index is straightforward to compute and interpret, making it widely used for evaluating clustering algorithms. It considers both concordant and discordant pairs of points, providing a comprehensive assessment of clustering similarity.

5.1.2 Evaluation of Outputs

Rand measure is the most important evaluation method in this project. We will pick the classification method with the highest number of rand measure score as the best classification method for our project. The rand Index provides a simple and intuitive measure of similarity between classification results and true label. It is straightforward and easy for us to understand. We submit our results into Tabular Playground Series Website to get our scores.

1. **K-means**

Figure 5.1: Evaluation score of K-means

2. **Gaussian Mixture Model (GMM)**

Figure 5.2: Evaluation score of GMM

3. **Bayesian Gaussian Mixture Model (BGMM)**

Figure 5.3: Evaluation score of BGMM

Figure 5.3 shows the rand measure scores of three unsupervised methods. We also want to see if unsupervised method combining with LightGBM, which is a supervised method, will perform better than that before combination.

1. **GMM with LightGBM**

Figure 5.4: Evaluation score of GMM with LightGBM

2. **BGMM with LightGBM**

Figure 5.5: Evaluation score of BGMM with LightGBM

There are two scores for each method, one is public score and another is private score. The public score is calculated by using approximately 20% of the test data. The private score is calculated by using approximately 80% of the test data. The two scores is designed for competitors manually changing values and keeping submitting until they getting a perfect submission. From the rand measure results we can see that combining a Gaussian mixture model with a supervised method can outperform using just a single GMM model. However, because our dataset is from a competition and we do not have the truth label. So even if we could know which classification method is best for our problem, we still do not know the reason why one result is better than the other. So in the next, we will use intercluster and intracluster distance to give us more details of the behaviors of the classification method. We can see that LightGBM combining with BGMM method has the highest score while K-means has the lowest score. Next, we will try to get more insight view of these five results and their corresponding clusters.

5.2 Cluster Analysis

In this part of evaluation, we use distance to evaluate our classification results. Because in an unsupervised learning dataset, we normally consider that the distances between the points in different clusters are longer than those in the same cluster. So we will discuss about the distance between the objects of the different clusters and the objects of the same clusters. There are two types of distance – intercluster distance and intracluster distance.

The aim of the clustering process is to discover overall distribution patterns and interesting correlations among the data attributes. It is the task of grouping a set of objects in such a way that objects in the same group are more similar to each other than to those in other groups. Cluster analysis itself is not one specific algorithm,

but the general task to be solved. It can be achieved by various algorithms that differ significantly in their understanding of what constitutes a cluster and how to efficiently find them.

5.2.1 Intracluster Distance

Intracluster distance is a crucial metric in cluster analysis. We can consider it as the diameter of one cluster. It provides insights into the compactness or dispersion of data points within individual clusters. It refers to the average distance between all pairs of points within the same cluster. Normally, the less the mean intracluster distance of all clusters the better the convergence of the problem.

The calculation of intracluster distance typically involves measuring the distances between all pairs of data points within a cluster and then averaging these distances. Various distance metrics can be utilized for this purpose, including Euclidean distance, Manhattan distance, and cosine similarity, depending on the nature of the data and the requirements of the analysis.

1. **Complete Diameter Distance**
   The complete diameter distance is the distance between two most remote objects belonging to the same cluster. This distance metric provides valuable information about the spread or dispersion of data points within a cluster. Unlike other intracluster distance, which calculates the average distance between all pairs of points within a cluster, complete diameter distance focuses on the furthest distance between two points. The biggest advantage of this method is easy to calculate as
   $$d_1(S) = max\left\{d(x, y)\right\}. \tag{5.2}$$

2. **Average Diameter Distance**
   The average diameter distance is the average distance between all the objects belonging to the same cluster. Unlike other intracluster distance calculation methods, which calculates the average distance of each point to the centroid or the center of the cluster, average diameter distance considers the distances between all possible pairs of points within the cluster. This metric provides a comprehensive measure of the overall spread or dispersion of data points within

the cluster, given by

$$d_2(S) = \frac{1}{|S|\,(|S|-1)} \sum_{x,y} \{d(x,y)\}\,. \tag{5.3}$$

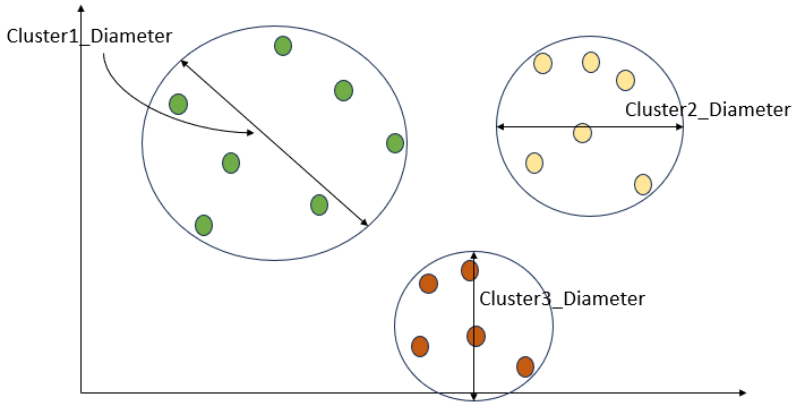


Figure 5.6: Intracluster Distance

3. **Centroid Diameter Distance**

   The centroid diameter distance is double average distance between all of the objects and the cluster center of s. Unlike other distance metrics that focus on pairwise distances between points within a cluster, centroid diameter distance provides a measure of the average spread or dispersion of points around the centroid. Centroid diameter distance serves as a measure of the compactness or concentration of data points within the cluster around the centroid. Centroid diameter distance provides valuable insights into the structure and cohesion of clusters in cluster analysis, helping to evaluate clustering results and guide further analysis and interpretation of clustered data [11]. It is given by

$$d_3(S) = 2\left\{\frac{\sum_{x\in S} d(x,v)}{|S|}\right\} \tag{5.4}$$

$$v = \frac{1}{|S|}\sum_{x\in S} x. \tag{5.5}$$

Taking all things into consideration, we pick the third metric to calculate the intracluster distance of the three classification results. If a clustering algorithm makes clusters so that the intracluster distance of a same cluster is less, then we can tell that it is a good clustering algorithm.

```
sum = 0;
for i = 1:7
[a,b]=size(submatrix{i});
Cen = Cen_matrix(i,:);
sum2 = 0;%%%sum2 means one Centroid to other points' distance
for j = 1:7
    if j ~= i
        A = submatrix{j};
        [a2,b2]=size(submatrix{j});
        for o = 1:a2
            sum2 = sum2 + norm(Cen-A(o,:));
        end
    end
end
shu = a*sum2/98000/(98000-a)
sum = sum + shu;
end
```

Figure 5.7: Code of Each clusters' diameter

The completion of distance calculation is carried out in matlab. First, we need to get all cluster centroid coordinates. For example, in the BGMM with LightGBM result, these seven centroid coordinates in GMM classification are listed in Table 3.

Table 3:Cluster Centroid Coordinates in GMM

GMM	feature1	feature2	feature3	...	feature29
Cluster1	0.0047	0.0170	0.0030	...	0.0297
Cluster2	0.0075	0.0049	-0.0059	...	-0.4225
Cluster3	0.0057	0.0032	0.0074	...	0.0423
Cluster4	-1.0266e-04	-0.0065	0.0058	...	-0.2958
Cluster5	-0.0073	0.0133	-0.0172	...	-0.3051
Cluster6	-0.0077	-1.1532e-04	0.0061	...	-0.3376
Cluster7	0.0064	0.0046	-0.0030	...	-0.3361

Then we get all seven clusters' diameters. To get a mean cluster diameter for us easily comparing the classification performance of five methods, we have to normalize

the diameters. Weights of each cluster is the number of objects in the corresponding cluster divided by the total number of objects. It is given by

$$d_3 = \sum_{i=1}^{K} \frac{d_3(S_i)n_i}{n} \tag{5.6}$$

where $d_3(S_i)$ is the cluster diameter of the i-th cluster and n_i is the number of points in the i-th cluster. We list GMM's result for example, and the other four classifications follow the same process.

Table 4:Cluster Diameters in GMM

GMM	Diameter	Weighted Diameter
Cluster1	20.9845	4.5855
Cluster2	23.3263	3.0934
Cluster3	21.1358	1.6311
Cluster4	22.7659	4.9746
Cluster5	20.0100	3.4489
Cluster6	20.4583	2.8936
Cluster7	20.9055	0.8266

Adding up the seven weighted diameter we can get the mean diameter: 20.8904. Then input the other four method's result to get mean diameter. Compare these five mean diameters we could find which classification method has better convergence ability for a same dataset.

```
%Kmeans
%mean:23.2365
%GMM
%mean:21.4497
%BGMM
%mean:21.2234
%GMM with LightGBM
%mean:21.3996
%BGMM with LightGBM
%mean:21.0125
```

We can find that the classification result of BGMM with LightGBM has the best convergence, followed by BGMM, GMM with LightGBM, GMM. K-means is the worst. It is the same rank as using the rand measure.

5.2.2 Intercluster Distance

Intercluster distance is the distance between two objects belongs to two different clusters. Intercluster distance refers to the measure of the separation or dissimilarity between different clusters in a dataset or within a clustering algorithm.

There are several ways to measure intercluster distance, depending on the specific clustering algorithm being used. There are five types of intercluster distance.

1. **Single Linkage Distance**
   The single linkage distance is the closest distance between two objects belonging to two different clusters. In single linkage clustering, the distance between two clusters is defined as the shortest distance between any two points in the different clusters. In other words, the distance between two clusters is determined by the minimum distance between any point in one cluster and any point in the other cluster. Mathematically, the single linkage distance between two clusters S and T can be expressed as

$$d_1(S,T) = min\left\{d(x,y), x \in S, y \in T\right\}. \tag{5.7}$$

2. **Complete Linkage Distance**
   The complete linkage distance is the distance between two most remote objects belonging to two different clusters. In complete linkage clustering, the distance between two clusters is defined as the maximum distance between any two points in the different clusters. In other words, the distance between two clusters is determined by the farthest distance between any point in one cluster and any point in the other cluster. Mathematically, the complete linkage distance between two clusters S and T can be expressed as

$$d_2(S,T) = max\left\{d(x,y), x \in S, y \in T\right\}. \tag{5.8}$$

3. **Average Linkage Distance**
   The average linkage distance is the average distance between all the objetcts belonging to two different clusters. It is expressed as

$$d_3(S,T) = \frac{1}{|S|\,|T|}\sum_{x\in S, y\in T} d(x,y). \tag{5.9}$$

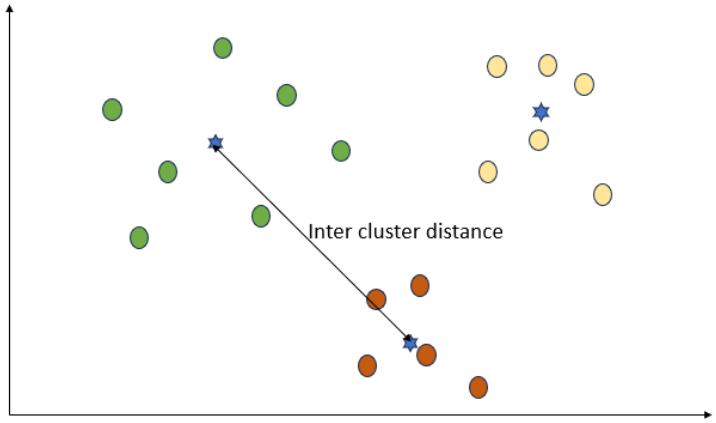


Figure 5.8: Intercluster Distance

Average linkage clustering tends to produce clusters that are more balanced and less susceptible to outliers compared to single linkage clustering. It is computationally efficient and often results in well-separated clusters. However, it can still suffer from the problem of "chaining" if clusters are elongated or contain outliers.

4. **Centroid Linkage Distance**
   The centroid linkage distance is the distance between the centers v_s and v_t of two clusters S and T respectively. It is given by

$$d_4(S,T) = d(v_s, v_t) \tag{5.10}$$

$$v_s = \frac{1}{|S|}\sum_{x\in S} x, v_t = \frac{1}{|T|}\sum_{y\in T} y. \tag{5.11}$$

   Centroid linkage distance method measures the distance between centroids. It is less sensitive to outliers compared to single linkage clustering but can be sensitive to clusters of varying sizes. This method is often used when the clusters' shapes are not well-defined or when the data distribution is not uniform.

5. **Average Centroid Linkage Distance**
   The average centroid linkage distance is the distance between the center of a cluster and all the objects belonging to a different cluster [6]. Mathematically, the average centroid linkage distance between two clusters S and T can be

expressed as

$$d_5(S,T) = \frac{1}{|S|+|T|}\left\{\sum_{x\in S} d(x,vt) + \sum_{y\in T} d(y,vs)\right\}. \tag{5.12}$$

The choice of intercluster distance measure can impact the performance and interpretation of clustering algorithms. Generally, smaller intercluster distances indicate better clustering, as it suggests that clusters are more compact and well-separated from each other. In our project, because we have more than two clusters to be evaluated, if we apply one of above intercluster distance methods, we have to calculate $\binom{K}{2}$ pairs of distance and essentially it is hard to be normalized. So a new method named multi-cluster average centroid linkage distance is proposed as

$$Inter\_Cluster\_distance = \sum_{i=1}^{K} \frac{m_i}{M} \frac{\sum_{x\notin S_i} d(x,V_i)}{M-m_i} \tag{5.13}$$

where K is the number of cluster, v_i is the centroid of the i-th cluster, S_i is the i-th cluster, m_i is the number of points in the i-th cluster and M is the number of total points.

In this new intercluster distance method, we will calculate all seven centroid average distance corresponding to seven clusters. For each calculation process, we will consider other six clusters as a whole cluster. After the seven distance calculations, we will normalize that by the number of points in each. So we can get the mean intercluster distance which we could use to evaluate our result.

Table 5:Intercluster Distance in BGMM with LightGBM

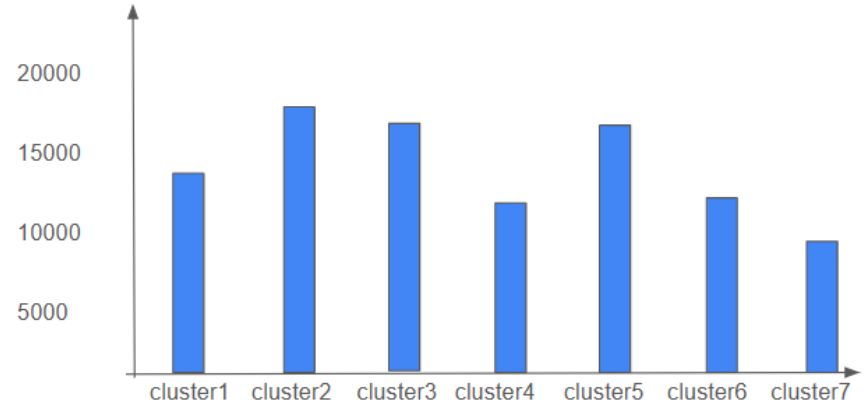
BGMM with LightGBM	Distance	Weighted Distance
cluster1 to noncluster1	17.7195	2.3590
cluster2 to noncluster2	14.1723	2.4855
cluster3 to noncluster3	14.7012	2.4124
cluster4 to noncluster4	15.2375	1.7511
cluster5 to noncluster5	14.7214	2.4370
cluster6 to noncluster6	15.0945	2.1459
cluster7 to noncluster7	14.2379	1.4918

Table 5 shows that for BGMM with LightGBM classification average distances of each cluster to the other six clusters and their weighted values. Adding up seven

weighted distance we can get the mean distance for this classification result. The mean distance is 15.0826.

Let us take an overview of the five results which correspond to five different classification methods. As we have known, the result of LightGBM with BGMM classification method has the highest rank measure score, and we find the distribution of the number of points in seven clusters in LightGBM classification result is more even than the other two results as Fig. 5.9 shows

(a) GMM



(b) BGMM with LightGBM

Figure 5.9: Number of data points in each cluster

However, we cannot conclude that BGMM with LightGBM classification method is better than the others just with its objects numbers of each cluster are more even.

But we could use intercluster distance to compare the results. In this evaluation method, we still need to get all seven clusters' centroid coordinates, and we also need to get the centroid coordinates of all points in the other six clusters. Then we can get the distance between cluster centroid and the points out of this cluster, which represents how clearly this cluster is distinguished from the others. For example, A cluster is one of the seven clusters in the GMM classification result. After we get the distance between A and S_{-A}, we let the distance multiply the ratio of the number of points in cluster A divided by the number of all points.

If a clustering algorithm makes the intercluster distance between different clusters further, then it is considered a good clustering algorithm. The multi-cluster average centroid linkage distance of K-means classification method is **13.0542**; That distance for GMM classification mehod is **14.9903**; That distance for BGMM classification mehod is **15.0310**; That distance for GMM with LightGBM classification mehod is **15.0123** and that distance for BGMM with LightGBM classification mehod is **15.0826**.

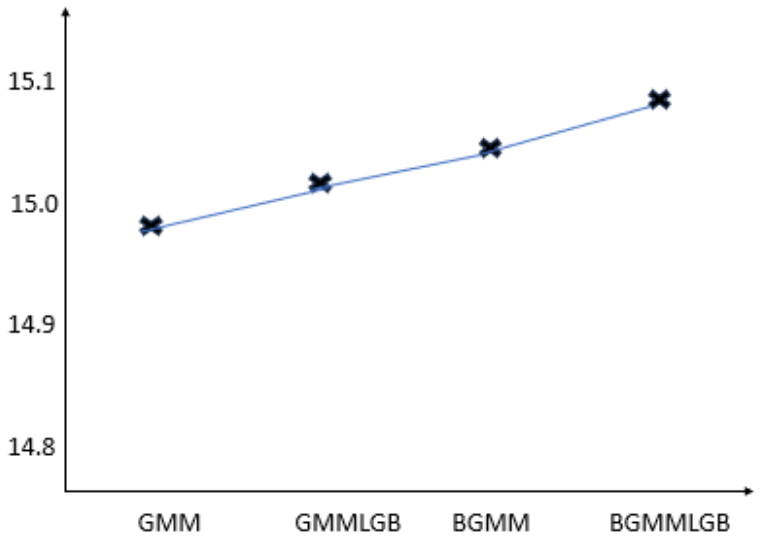


Figure 5.10: The intercluster distances of four classifications

Fig. 5.10 plots the multi-cluster average centroid linkage distances of four methods. From Fig. 5.10 we can see that the interclsuter distance of BGMM with LightGBM is biggest. The second is BGMM. The ranking is in agreement with the rand measure scores. So we get another evidence proving the best classification method

in five methods for this dataset is BGMM with LGB. But for efficacy, BGMM with LightGBM is not the best algorithm in all five methods. On the contrary, BGMM with LightGBM cost the longest runtime compared with other four methods. On my personal Laptop, the model needs 40 minutes to complete the training, 7 minutes for GMM, 9 minutes for BGMM and 3 minutes for K-means. So if we take time consumption into our consideration, BGMM with LightGBM may be not our best choice.

Based on the evaluation results, it becomes evident that integrating a Gaussian mixture model with a supervised method yields superior performance compared to employing solely a GMM model. This is because it can enhance representation. GMM captures the underlying distribution of the data. By combining it with a supervised method, we essentially enrich the representation of the data. The supervised method can further refine this representation based on high-responsibility labeled data, making it more informative.

Chapter 6

Conclusions and Future Work

In this project, we have explored the application of various classification methodologies, K-means, Gaussian mixture model, Bayesian Gaussian mixture model and LightGBM based on BGMM and GMM separately, on a challenging dataset from a Kaggle playground prediction competition. The objective was to cluster the dataset effectively with unsupervised learning.

First, a comprehensive analysis of the dataset has been conducted, including the examination of discrete and continuous feature distributions using Shapiro-Wilk tests, Poisson dispersion tests, and Q-Q plots. This analysis has provided valuable insights into the dataset, highlighting potential challenges and guiding preprocessing steps.

Subsequently, each classification method has been implemented and evaluated using three evaluation metrics: the rand index, intracluster distance and intercluster distance. The rand index serves as a measure of clustering similarity between predicted and ground truth labels, while intracluster and intercluster distances provided insights into the compactness and separation of clusters, respectively. The intercluster distance calculation method used in this project is based on a proposed formula.

The results have revealed that LightGBM combined with BGMM consistently outperforms the other methods in terms of clustering accuracy and cluster distinctiveness. When we applied LightGBM on the results of BGMM or GMM, we could see their performances both improved. So compared to single unsupervised method application, combining unsupervised method with some supervised learning may turn out better. This hybrid approach have demonstrated superior performance, making it the preferred classification method for this dataset.

In conclusion, this project have showed different classification methods lead to different results and the combination of unsupervised method and supervised method

may be a reliable way when we try to find some approach to improve the performance. This project also emphasized the importance of evaluating multiple methods to select an appropriate classification methodology for the dataset. By selecting advanced techniques and thorough evaluation, we can extract valuable insights from complex datasets, ultimately facilitating decision-making and advancing the field of clustering analysis. Further research may explore additional classification algorithms and innovative evaluation techniques to enhance clustering accuracy in diverse data-driven applications.

For future work, while this study has explored five approaches to clustering the dataset, future investigations could focus on a broader choices of unsupervised classification techniques. By evaluating additional classification metrics, we may uncover alternative methodologies that offer improved performance than that introduced in this report.

Since dataset preprocessing plays a significant role in shaping clustering results, our future research will aim to examine a broader range of preprocessing methods to understand how they might improve clustering effectiveness.

Additionally, we might improve the performance of specific classification methods by selectively removing certain features from the dataset before training the model. Therefore, future research efforts could focus on using advanced analytical techniques to identify the underlying factors that make these features disruptive to the clustering process.

Furthermore, since our study lacked true labels, we had to explore alternative methods for evaluating cluster results. In the future, researchers could work on developing and implementing new evaluation methods that go beyond traditional methods like intracluster and intercluster distances.

Appendix A

Additional Information

Main code of Kmeans classification Project

```
    SEED = 42
np.random.seed = SEED
## Input
df = pd.read_csv('data.csv')
print(df.shape)
df.head()

### Let's make a little exploratory data analysis + data engineering
count = 1
for c in df.columns:
print(f'{count} - c')
print(f'- # of unique elements: {df[c].nunique()}')
print(f'- Sample: {df[c].unique()[0:20]}')
print(f'- Dtype: {df[c].dtype}')
print(f'- # of missing values: {df[c].isnull().sum()} of {df.shape[0]}')
print(f'- % of missing values: {np.round(df[c].isnull().sum() / df.shape[0], 3)}')

if df[c].dtype == int or df[c].dtype == float:
s = "- Statistics:"

me = np.round(df[c].mean(), 2)
st = np.round(df[c].std(), 2)
```

```
23 s += f"– Mean (std): {me} ({st})"
24
25 q1 = np.round(df[c].quantile(0.25), 2)
26 q2 = np.round(df[c].quantile(0.5), 2)
27 q3 = np.round(df[c].quantile(0.75), 2)
28 s += f"– Quantiles: q1={q1}, q2={q2}, q3={q3}"
29 s += f"– Min {df[c].min()}"
30 s += f"– Max {df[c].max()}"
31 print(s)
32
33 print('=' * 30)
34 count += 1
35
36
37 # # # Applying K-means
38 df_model = df.drop(columns=['id'])
39
40 kmeans = KMeans(n_clusters=8, init='random', random_state=SEED)
41 kmeans.fit(df_model)
42 labels = kmeans.labels_
43
```

Main code of GMM classification Project

```
1     class GMM:
2 def __init__(self, k, max_iter=100, random_state = 0):
3 self.k = k
4 self.max_iter = max_iter
5 self.random_state = random_state
6
7 def initialise(self, X):
8 self.shape = X.shape
9 self.n, self.d = self.shape
10
11 self.pi = np.full(shape=self.k, fill_value=1 / self.k)
```

```
12 self.responsibilities = np.full(shape=self.shape, fill_value=1 / self.k)
13
14 np.random.seed(self.random_state)
15 random_row = np.random.randint(low=0, high=self.n, size=self.k)
16 self.mu = [X[row_index, :] for row_index in random_row]
17 self.sigma = [np.cov(X.T) for _ in range(self.k)]
18
19 def E_step(self, X):
20 # E-Step: update the responsibilities by holding mu and sigma constant
21 self.responsibilities = self.predict_proba(X)
22
23 def M_step(self, X):
24 # M-Step: update pi, mu and sigma by holding responsibilities constant
25 self.pi = self.responsibilities.mean(axis=0)
26 for j in range(self.k):
27 r_column = self.responsibilities[:, j]
28 total_responsibility = r_column.sum()
29 self.mu[j] = (X * r_column[:, np.newaxis]).sum(axis=0) / total_responsibility
30 self.sigma[j] = np.cov(X.T, aweights=(r_column / total_responsibility).flatten(),
31 bias=True)
32
33 def fit(self, X):
34 self.initialise(X)
35
36 for iteration in range(self.max_iter):
37 self.E_step(X)
38 self.M_step(X)
39
40 def predict_proba(self, X):
41 likelihood = np.zeros((self.n, self.k))
42 for j in range(self.k):
43 distribution = multivariate_normal(mean=self.mu[j], cov=self.sigma[j])
44 likelihood[:, j] = distribution.pdf(X)
45
46 numerator = likelihood * self.pi
```

```
denominator = numerator.sum(axis=1)[:, np.newaxis]
responsibilities = numerator / denominator
return responsibilities

def predict(self, X):
responsibilities = self.predict_proba(X)
return np.argmax(responsibilities, axis=1)

def fit_predict(self, X):
self.fit(X)
predictions = self.predict(X)
return predictions

# Preprocess data
scaled_data = pd.DataFrame(PowerTransformer().fit_transform(data))
scaled_data.columns = data.columns
drop_feats = [f'f_0i' for i in range(7)]
drop_feats = drop_feats + [f'f_i' for i in range(14,22)]
X = scaled_data.drop(drop_feats, axis=1).values


# Gaussian Mixture Model
gmm = GMM(k=7, max_iter=100)
preds = gmm.fit_predict(X)

```

Main code of LightGBM with BGMM classification Project

```
    def iqr_outliers(df,col_list):
for col in col_list:
q1 = df[col].quantile(0.25)
q3 = df[col].quantile(0.75)
iqr = q3-q1
Lower_tail = q1 - 2 * iqr
Upper_tail = q3 + 2 * iqr
```

```
df.loc[df[col]   Upper_tail,col ]=Upper_tail
df.loc[df[col] < Lower_tail,col]=Lower_tail
return df


train=iqr_outliers(train,float_cols)

train_scaled=train.copy()
train_scaled[best_cols]= PowerTransformer().fit_transform(train_scaled[best_cols])
train_scaled[best_cols]= MinMaxScaler().fit_transform(train_scaled[best_cols])

BGM = BayesianGaussianMixture(n_components=N_cluster,covariance_type='full',
max_iter=300, random_state=1,n_init = 5)
BGM.fit(train_scaled[best_cols])

predict=BGM.predict(train_scaled[best_cols])
proba=BGM.predict_proba(train_scaled[best_cols])

train_scaled['predict']=predict
train_scaled['predict_proba']=np.max(proba, axis=1)

train_index=train_scaled[train_scaled.predict_proba > 0.7].index
print(round(len(train_index)/len(train),3))

X = train_scaled.loc[train_index][best_cols]
y = train_scaled.loc[train_index]['predict']

params_lgb = {'learning_rate': 0.07, 'objective': 'multiclass', 'boosting': 'gbdt',
'verbosity': -1, 'n_jobs': -1, 'num_classes': N_cluster}

model_list = []

gkf = StratifiedKFold(N_FOLDS)
for fold, (train_idx, valid_idx) in enumerate(gkf.split(X, y)):
tr_dataset =
```

```
43 lgb.Dataset(X.iloc[train_idx], y.iloc[train_idx], feature_name=best_cols)
44 vl_dataset=
45 lgb.Dataset(X.iloc[valid_idx], y.iloc[valid_idx], feature_name=best_cols)
46
47 model =
48 lgb.train(params=params_lgb,train_set=tr_dataset,valid_sets=vl_dataset,
49 num_boost_round=5000,       callbacks=[lgb.early_stopping(stopping_rounds=300,
50 verbose=True), lgb.log_evaluation(period=200)])
51
52 model_list.append(model)
53
54 lgb_preds=0
55 for model in model_list:
56 lgb_preds+=model.predict(train_scaled[best_cols])
57
```

Bibliography

[1] Mohiuddin Ahmed, Raihan Seraj, and Syed Mohammed Shamsul Islam. The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics*, 9(8), 2020.

[2] H.B. Barlow. Unsupervised Learning. *Neural Computation*, 1(3):295–311, 09 1989.

[3] Michael W Berry, Azlinah Mohamed, and Bee Wah Yap. *Supervised and unsupervised learning for data science*. Springer, 2019.

[4] Jennifer G Dy and Carla E Brodley. Feature selection for unsupervised learning. *Journal of machine learning research*, 5(Aug):845–889, 2004.

[5] Fourth Edition, Athanasios Papoulis, and S Unnikrishna Pillai. *Probability, random variables, and stochastic processes*. McGraw-Hill Europe: New York, NY, USA, 2002.

[6] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38:293–306, 1985.

[7] Zofia Hanusz, Joanna Tarasinska, and Wojciech Zielinski. Shapiro–wilk test with known mean. *REVSTAT-Statistical Journal*, 14(1):89–100, Feb. 2016.

[8] Dimitris Karlis and Evdokia Xekalaki. A Simulation Comparison of Several Procedures for Testing the Poisson Assumption. *Journal of the Royal Statistical Society Series D: The Statistician*, 49(3):355–382, 12 2001.

[9] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision

tree. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[10] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

[11] Vijay Kumar, Jitender Kumar Chhabra, and Dinesh Kumar. Performance evaluation of distance metrics in the clustering algorithms. *INFOCOMP Journal of Computer Science*, 13(1):38–52, 2014.

[12] Quoc V. Le. Building high-level features using large scale unsupervised learning. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8595–8598, 2013.

[13] Geoffrey J McLachlan and Suren Rathnayake. On the number of components in a gaussian mixture model. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(5):341–355, 2014.

[14] Joachim D Pleil. Qq-plots for assessing distributions of biomarker measurements and generating defensible summary statistics. *Journal of Breath Research*, 10(3):035001, aug 2016.

[15] Jorge M. Santos and Mark Embrechts. On the use of the adjusted rand index as a metric for evaluating supervised classification. In Cesare Alippi, Marios Polycarpou, Christos Panayiotou, and Georgios Ellinas, editors, *Artificial Neural Networks – ICANN 2009*, pages 175–184, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[16] Jorge M Santos and Mark Embrechts. On the use of the adjusted rand index as a metric for evaluating supervised classification. In *International conference on artificial neural networks*, pages 175–184. Springer, 2009.

[17] Russell J Steele and Adrian E Raftery. Performance of bayesian model selection criteria for gaussian mixture models. *Frontiers of statistical decision making and bayesian analysis*, 2:113–130, 2010.

[18] Dehua Wang, Yang Zhang, and Yi Zhao. Lightgbm: an effective mirna classification method in breast cancer patients. In *Proceedings of the 2017 international conference on computational biology and bioinformatics*, pages 7–11, 2017.

[19] Guorong Xuan, Wei Zhang, and Peiqi Chai. Em algorithms of gaussian mixture model and hidden markov model. In *Proceedings 2001 international conference on image processing (Cat. No. 01CH37205)*, volume 1, pages 145–148. IEEE, 2001.