UnsupervisedLearning (Clustering)

Tejasree Kilari
TEAM-BRAINIACS

ABSTRACT-

While machine learning and artificial intelligence have long been applied in networking research, the bulk of such works has focused on supervised learning. Recently, there has been a rising trend of employing unsupervised machine learning using unstructured raw network data to improve network performance and provide services, such as traffic engineering, anomaly detection, Internet traffic classification, and quality of service optimization. The growing interest in applying unsupervised learning techniques in networking stems from their great success in other fields, such as computer vision, natural language processing, speech recognition, and optimal control (e.g., for developing autonomous self-driving cars). In addition, unsupervised learning can unconstraint us from the need for labeled data and manual handcrafted feature engineering, thereby facilitating flexible, general, and automated methods of machine learning. The focus of this survey paper is to provide an overview of applications of unsupervised learning in the domain of networking. We provide a comprehensive survey highlighting recent advancements in unsupervised learning techniques, and describe their applications in various learning tasks, in the context of networking. We also provide a discussion on future directions and open research issues, while identifying potential pitfalls. While a few survey papers focusing on applications of machine learning in networking have previously been published, a survey of similar scope and breadth is missing in the literature. Through this timely review, we aim to advance the current state of knowledge, by carefully synthesizing insights from previous survey papers, while providing contemporary coverage of the recent advances and innovations.

I. INTRODUCTION

The main feature of unsupervised learning algorithms, when compared to classification and regression methods, is that input data are unlabeled (i.e. no labels or classes given) and that the algorithm learns the structure of the data without any assistance. This creates two main differences. First, it allows us to process large amounts of data because the data does not need to be manually labeled. Second, it is difficult to evaluate the quality of an unsupervised algorithm due to the absence of an explicit goodness metric as used in supervised learning. One of the most common tasks in unsupervised learning is dimensionality reduction. On one hand, dimensionality reduction may help with data visualization (e.g. t-SNA method) while, on the other hand, it may help deal with the multicollinearity of your data and prepare the data for a supervised learning method (e.g. decision trees).

Here, are prime reasons for using Unsupervised Learning in MachineLearning: Unsupervised machine learning finds all kind of unknown patterns in data. Unsupervised methods help you to find features which can be useful for categorization. It is taken place in real time, so all the input data to be analyzed and labeled in the presence of learners. It is easier to get unlabeled data from a computer than labeled data, which needs manual intervention.

Dataset Description: The Activities of Daily Living (ADLs) Recognition dataset used in this project contains information about the ADLs performed by two users in their own homes. The dataset is composed of two instances of data, each one corresponding to a different user, and containing 21 days of fully labeled data. Each instance of the dataset is described by three text files: description, sensors' events (features), and activities of daily living (labels). The sensors' events file contains 12 sensor readings, representing the state of different devices or activities such as opening a door or turning on a light. The activities of daily living file contains labels for the ADLs performed by the users, such as cooking, cleaning, or watching TV. There are 10 different ADL labels in the dataset. The dataset can be considered a mixture of continuous and categorical data. The sensor readings are binary (0 or 1) and therefore categorical, while the frequency of the readings can be considered continuous. The ADL labels are categorical

II.PROJECT BACKGROUND

Unsupervised learning models are utilized for three main tasks—clustering, association, and dimensionality reduction. Below we'll define each learning method and highlight common algorithms and approaches to conduct them effectively.

Clustering Clustering is a data mining technique which groups unlabeled data based on their similarities or differences. Clustering algorithms are used to process raw, unclassified data objects into groups represented by structures or patterns in the information. Clustering algorithms can be categorized into a few types, specifically exclusive, overlapping, hierarchical, and probabilistic.

Exclusive and Overlapping Clustering Exclusive clustering is a form of grouping that stipulates a data point can exist only in one cluster. This can also be referred to as “hard” clustering. The Kmeans clustering algorithm is an example of exclusive clustering. K-means clustering is a common example of an exclusive clustering method where data points are assigned into K groups, where K represents the number of clusters based on the distance from each group’s centroid. The data points closest to a given centroid will be clustered under the same category. A larger K value will be indicative of smaller groupings with more granularity whereas a smaller K value will have larger groupings and less granularity. K-means clustering is commonly used in market segmentation, document clustering, image segmentation, and image compression. Overlapping clusters differs from exclusive clustering in that it allows data points to belong to multiple clusters with separate degrees of membership. “Soft” or fuzzy k-means clustering is an example of overlapping clustering. Hierarchical clustering Hierarchical clustering, also known as hierarchical cluster analysis (HCA), is an unsupervised clustering algorithm that can be categorized in two ways; they can be agglomerative or divisive. Agglomerative clustering is considered a “bottoms-up approach.” Its data points are isolated as separate groupings initially, and then they are merged together iteratively on the basis of similarity until one cluster has been achieved. Four different methods are commonly used to measure similarity:

Ward’s linkage: This method states that the distance between two clusters is defined by the increase in the sum of squared after the clusters are merged. Average linkage: This method is defined by the mean distance between two points in each cluster Complete (or maximum) linkage: This method is defined by the maximum distance between two points in each cluster Single (or minimum) linkage: This method is defined by the minimum distance between two points in each cluster Euclidean distance is the most common metric used to calculate these distances; however, other metrics, such as Manhattan distance, are also cited in clustering literature.

Divisive clustering can be defined as the opposite of agglomerative clustering; instead it takes a “top-down” approach. In this case, a single data cluster is divided based on the differences between data points. Divisive clustering is not commonly used, but it is still worth noting in the context of hierarchical clustering. These clustering processes are usually visualized using a dendrogram, a tree-like diagram that documents the merging or splitting of data points at each iteration.

Probabilistic clustering A probabilistic model is an unsupervised technique that helps us solve density estimation or “soft” clustering problems. In probabilistic clustering, data points are clustered based on the likelihood that they belong to a particular distribution. The Gaussian Mixture Model (GMM) is the one of the most commonly used probabilistic clustering methods. Gaussian Mixture Models are classified as mixture models, which means that they are made up of an unspecified number of probability distribution functions. GMMs are primarily leveraged to determine which Gaussian, or normal, probability distribution a given data point belongs to. If the mean or variance are known, then we can determine which distribution a given data point belongs to. However, in GMMs, these variables are not known, so we assume that a latent, or hidden, variable exists to cluster data points appropriately. While it is not required to use the Expectation-Maximization (EM) algorithm, it is a commonly used to estimate the assignment probabilities for a given data point to a particular data cluster.

Dimensionality reduction While more data generally yields more accurate results, it can also impact the performance of machine learning algorithms (e.g. overfitting) and it can also make it difficult to visualize datasets. Dimensionality reduction is a technique used when the number of features, or dimensions, in a

given dataset is too high. It reduces the number of data inputs to a manageable size while also preserving the integrity of the dataset as much as possible. It is commonly used in the preprocessing data stage, and there are a few different dimensionality reduction methods that can be used, such as: Principal component analysis Principal component analysis (PCA) is a type of dimensionality reduction algorithm which is used to reduce redundancies and to compress datasets through feature extraction. This method uses a linear transformation to create a new data representation, yielding a set of ”principal components.” The first principal component is the direction which maximizes the variance of the dataset. While the second principal component also finds the maximum variance in the data, it is completely uncorrelated to the first principal component, yielding a direction that is perpendicular, or orthogonal, to the first component. This process repeats based on the number of dimensions, where a next principal component is the direction orthogonal to the prior components with the most variance.

Singular value decomposition Singular value decomposition (SVD) is another dimensionality reduction approach which factorizes a matrix, A, into three, low-rank matrices. SVD is denoted by the formula, A = USVT, where U and V are orthogonal matrices. S is a diagonal matrix, and S values are considered singular values of matrix A. Similar to PCA, it is commonly used to reduce noise and compress data, such as image files.

Autoencoders Autoencoders leverage neural networks to compress data and then recreate a new representation of the original data’s input. Looking at the image below, you can see that the hidden layer specifically acts as a bottleneck to compress the input layer prior to reconstructing within the output layer. The stage from the input layer to the hidden layer is referred to as

“encoding” while the stage from the hidden layer to the output layer is known as “decoding.”

Applications of unsupervised learning Machine learning techniques have become a common method to improve a product user experience and to test systems for quality assurance. Unsupervised learning provides an exploratory path to view data, allowing businesses to identify patterns in large volumes of data more quickly when compared to manual observation. Some of the most common real-world applications of unsupervised learning are:

News Sections: Google News uses unsupervised learning to categorize articles on the same story from various online news outlets. For example, the results of a presidential election could be categorized under their label for “US” news. Computer vision: Unsupervised learning algorithms are used for visual perception tasks, such as object recognition. Medical imaging: Unsupervised machine learning provides essential features to medical imaging devices, such as image detection, classification and segmentation, used in radiology and pathology to diagnose patients quickly and accurately. Anomaly detection: Unsupervised learning models can comb through large amounts of data and discover atypical data points within a dataset. These anomalies can raise awareness around faulty equipment, human error, or breaches in security. Customer personas: Defining customer personas makes it easier to understand common traits and business clients’ purchasing habits. Unsupervised learning allows businesses to build better buyer persona profiles, enabling organizations to align their product messaging more appropriately. Recommendation Engines: Using past purchase behavior data, unsupervised learning can help to discover data trends that can be used to develop more effective cross-selling strategies. This is used to make relevant add-on recommendations to customers during the checkout process for online retailers.

III METHODOLOGY

Library Functions:

The libraries used in our project are Importing the NumPy library with the alias ”np” makes it easier to use: import numpy as np. Popular Python toolkit for numerical computing, NumPy offers robust array operations and tools for linear algebra. Importing the CSV (comma-separated values) module gives you the ability to read and write CSV files.

Importing the math module, which offers mathematical constants and functions, is known as ”import math.”

Importing the pandas library with the alias ”pd” causes it to be imported. Pandas is a well-known Python package for data analysis and manipulation that offers strong features for working with structured data (like spreadsheets and databases).

import from hmmlearn HMM: imports the hmmlearn library’s Hidden Markov Model (HMM) implementation. HMMs are statistical models that are used to represent temporal or sequential data, such market prices or speech.

imports a number of performance measures from the wellknown Python machine learning library scikit-learn using the from sklearn.metrics command. These measures are used to assess how well clustering algorithms like K-means or DBSCAN perform.

Importing the pyplot module from the Matplotlib package with the alias ”plt” causes it to be imported. Popular Python module Matplotlib is used to build graphs and visualizations.

Importing the datetime module, which offers classes for working with dates and times in Python, imports the datetime module. model selection from sklearn import train test split imports the model selection module’s train test split function from the Scikit-Learn package. For machine learning, this function divides data into training and testing sets. using sklearn.processing before import StandardScaler imports the StandardScaler class from the scikit-learn library’s preprocessing module. Data are scaled using the StandardScaler preprocessing step to have a zero mean and unit variance.

I Python.display import display: Imports the IPython.display module’s display function. In Jupyter notebooks and other IPython environments, HTML, pictures, videos, and other media kinds are shown using this function.

import from sklearn.cluster KMeans imports the KMeans class from the scikit-learn library’s cluster module. A wellliked clustering algorithm for putting related data points together is KMeans. import from sklearn.decomposition PCA: imports the PCA class from the scikit-learn library’s decomposition module. PCA (Principal Component Analysis) is a method for projecting data onto a lower-dimensional space in order to reduce the dimensionality of the data while retaining as much of the original data as possible.

One-hot coding and PCA analysis:

These lines of code are performed for data preprocessing steps using Pandas, scikit-learn, and NumPy libraries. Here’s a brief explanation of each step: columns-to-encode: a list of column names in the DataFrame to perform one-hot encoding on. for column-name in columnsto-encode:: a loop that iterates over each column in the columns-to-encode list.

sensor-dframe = pd.concat([sensor-Transpose, one-hot], axis=1): concatenates the one-hot encoded columns to the original DataFrame using the pd.concat() function in Pandas. sensor-datadf = sensor-Transpose.drop(column-name, axis=1, inplace=True): drops the original column that was one-hot encoded from the original Dataframe using the Dataframe.drop() method in Pandas. The inplace=True argument modifies the DataFrame in place rather than returning a new Dataframe. dataframe = sensordframe.dropna(): drops any rows with missing values from the DataFrame using the DataFrame.dropna() method in Pandas. scaler = StandardScaler(): initializes a StandardScaler object from the preprocessing module in scikit-learn. This object will be used to scale the data to have zero mean and unit variance. Data = scaler.fit-transform(dataframe): applies the StandardScaler object to the dataframe DataFrame and scales the data to have zero mean and unit variance using the fittransform() method. pca = PCA(n-components=3): initializes a PCA object from the decomposition module in scikit-learn. This object will be used to perform principal component analysis on the data and reduce its dimensionality to 3. data-pca = pca.fit-transform(Data): applies the PCA object to the scaled data and performs dimensionality reduction using the fit-transform() method.

PCA-df = pd.DataFrame(data = data-pca): creates a new DataFrame from the PCA output.

PCA-df.head(): displays the first few rows of the new DataFrame. Overall, these steps involve one-hot encoding categorical data, dropping missing values, scaling the data, and reducing its dimensionality using principal component analysis. These are common data preprocessing steps used to prepare data for machine learning algorithms.

K-Means:

scores = []: initializes an empty list to store the inertia scores for each value of k. for k in range:: iterates over values of k from 2 to

11. km= KMeans(n-clusters=k,random-state=123): initializes a KMeans object with the current value of k. km = km.fit(PCAdf): fits the KMeans object to the PCA-transformed data using the fit() method. scores.append(km.inertia): appends the inertia score for the current value of k to the scores list using the inertia attribute of the KMeans object. dfk = pd.DataFrame('Cluster':range(2,12), 'Score':scores): creates a new DataFrame with two columns, Cluster and Score, where Cluster contains the range of values for k and Score contains the corresponding inertia scores. plt.figure(figsize=(8,5)): initializes a new figure with a size of 8x5 using the figure() function from Matplotlib. plt.plot(dfk['Cluster'], dfk['Score'], marker='o'): creates a line plot of the inertia scores for each value of k using the plot() function from Matplotlib. The marker='o' argument adds circular markers at each data point.

plt.xlabel('Number of clusters'): sets the x-axis label to "Number of clusters" using the xlabel() function from Matplotlib.

plt.ylabel('Inertia'): sets the y-axis label to "Inertia" using the ylabel() function from Matplotlib. plt.show(): displays the plot.

The elbow method is used to determine the optimal number of clusters for the data. The plot shows the inertia scores decreasing as the number of clusters increases, with diminishing returns as the number of clusters increases beyond a certain point. The optimal number of clusters is typically the point of inflection where the curve begins to level off. for i in range(2,12):: iterates over values of k from 2 to 11. kmeans-labels=KMeans(nclusters=i,random-state=123).fit- predict(PCA-df): performs K-means clustering with the current value of k using the fit-predict() method from the KMeans object. The resulting labels are stored in the kmeans-labels variable. print("Silhouette score for clusters k-means : ".format(i,silhouette-score(Data,kmeans-labels, metric='euclidean').round(3))): computes the silhouette score for the current value of k using the silhouette-score() function from scikit-learn. The metric='euclidean' argument specifies the distance metric to use for computing pairwise distances between points. The silhouette score ranges from -1 to 1, with higher scores indicating better clustering performance.

The score is printed for each value of k.

Confusion Matrix:

pd.crosstab() is used to create a contingency table from two arrays: Updatedsensor-FinalData['Activity'] and predictedlabels. The contingency table shows how many instances fall into each combination of true activity label and predicted cluster label.

The contingency table is converted to a numpy array using .tonumpy().

sorted(Updatedsensor-FinalData['Activity'].unique()) and sorted(Updatedsensor-FinalData['Clusters'].unique()) are used to create sorted lists of unique values for the activity labels and predicted cluster labels, respectively.

A ConfusionMatrixDisplay object is created using confusion-matrix and activity-labels as input parameters. This object is used to plot the confusion matrix.

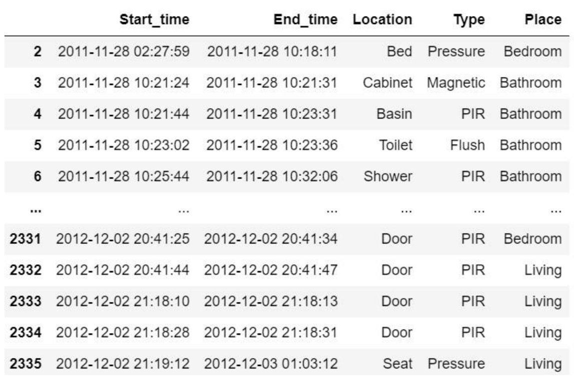
cmap='Purples' sets the color map for the plot to a purple color scheme.

xticks-rotation='vertical' rotates the x-axis tick labels verti- cally to make them easier to read. values-format='d' sets the format for the values in the cells of the confusion matrix to integers. np.arange(len(cluster-labels)) creates an array of integers from 0 to the number of predicted cluster labels, which is used as the x-axis tick positions. plt.xticks() sets the x-axis tick labels to the sorted predicted cluster labels. plt.title() sets the title of the plot to "Confusion

Matrix". plt.show() displays the plot. Final Accuracy: confusion-matrix.max(axis=0) calculates the maximum value in each column of the confusion matrix. This gives an array of the number of instances for each predicted cluster label that were correctly assigned to their true activity label. .sum() adds up all the values in the array from step 1. This gives the total number of instances that were correctly assigned to their true activity label.

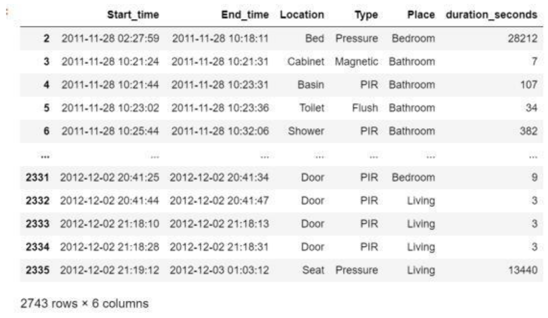
/len(Updatedsensor-FinalData) divides the total number of correctly assigned instances by the total number of instances in the dataset. This gives the final accuracy of the clustering model. final-accuracy = accuracy assigns the final accuracy to a variable called final-accuracy.

IV RESULTS



	Start_time	End_time	Location	Type	Place
2	2011-11-28 02:27:59	2011-11-28 10:18:11	Bed	Pressure	Bedroom
3	2011-11-28 10:21:24	2011-11-28 10:21:31	Cabinet	Magnetic	Bathroom
4	2011-11-28 10:21:44	2011-11-28 10:23:31	Basin	PIR	Bathroom
5	2011-11-28 10:23:02	2011-11-28 10:23:36	Toilet	Flush	Bathroom
6	2011-11-28 10:25:44	2011-11-28 10:32:06	Shower	PIR	Bathroom
...	...	...	...	...	...
2331	2012-12-02 20:41:25	2012-12-02 20:41:34	Door	PIR	Bedroom
2332	2012-12-02 20:41:44	2012-12-02 20:41:47	Door	PIR	Living
2333	2012-12-02 21:18:10	2012-12-02 21:18:13	Door	PIR	Living
2334	2012-12-02 21:18:28	2012-12-02 21:18:31	Door	PIR	Living
2335	2012-12-02 21:19:12	2012-12-03 01:03:12	Seat	Pressure	Living

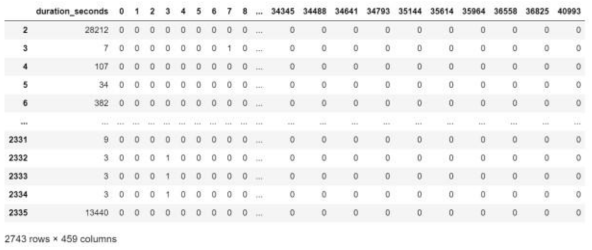
Sensors Data of User A and User B are combined.



	Start_time	End_time	Location	Type	Place	duration_seconds
2	2011-11-28 02:27:59	2011-11-28 10:18:11	Bed	Pressure	Bedroom	28212
3	2011-11-28 10:21:24	2011-11-28 10:21:31	Cabinet	Magnetic	Bathroom	7
4	2011-11-28 10:21:44	2011-11-28 10:23:31	Basin	PIR	Bathroom	107
5	2011-11-28 10:23:02	2011-11-28 10:23:36	Toilet	Flush	Bathroom	34
6	2011-11-28 10:25:44	2011-11-28 10:32:06	Shower	PIR	Bathroom	382
...	...	...	...	...	...	...
2331	2012-12-02 20:41:25	2012-12-02 20:41:34	Door	PIR	Bedroom	9
2332	2012-12-02 20:41:44	2012-12-02 20:41:47	Door	PIR	Living	3
2333	2012-12-02 21:18:10	2012-12-02 21:18:13	Door	PIR	Living	3
2334	2012-12-02 21:18:28	2012-12-02 21:18:31	Door	PIR	Living	3
2335	2012-12-02 21:19:12	2012-12-03 01:03:12	Seat	Pressure	Living	13440

2743 rows × 6 columns

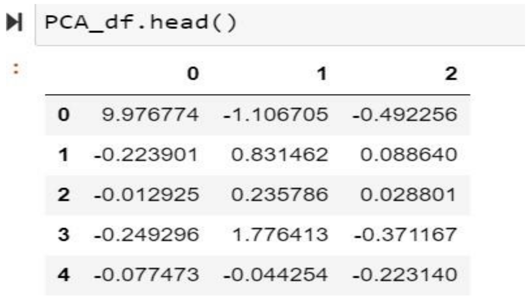
Sensors Data with Duration



	duration_seconds	0	1	2	3	4	5	6	7	8	...	34345	34488	34641	34793	35144	35614	35964	36558	36825	40993
2	28212	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	7	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
4	107	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5	34	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
6	382	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2331	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2332	3	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2333	3	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2334	3	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2335	13440	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

2743 rows × 459 columns

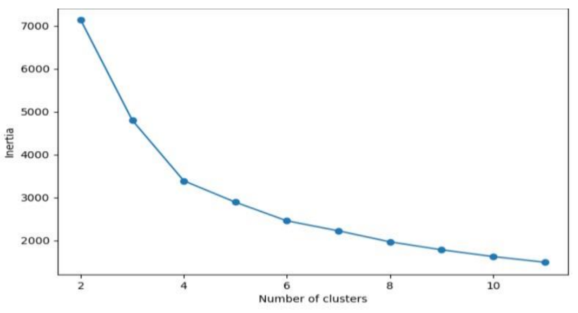
data



```
PCA_df.head()
```

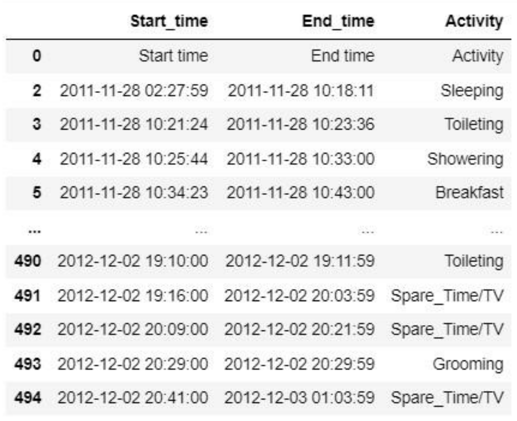
	0	1	2
0	9.976774	-1.106705	-0.492256
1	-0.223901	0.831462	0.088640
2	-0.012925	0.235786	0.028801
3	-0.249296	1.776413	-0.371167
4	-0.077473	-0.044254	-0.223140

One-Hot coding



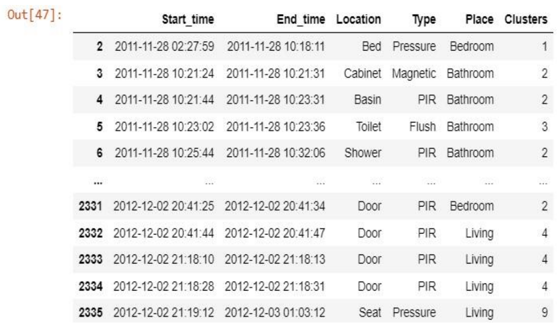
```
Silhouette score for 2 clusters k-means : 0.651
Silhouette score for 3 clusters k-means : 0.071
Silhouette score for 4 clusters k-means : 0.051
Silhouette score for 5 clusters k-means : 0.06
Silhouette score for 6 clusters k-means : 0.068
Silhouette score for 7 clusters k-means : 0.092
Silhouette score for 8 clusters k-means : 0.101
Silhouette score for 9 clusters k-means : 0.13
Silhouette score for 10 clusters k-means : 0.123
Silhouette score for 11 clusters k-means : 0.13
```

K-Means



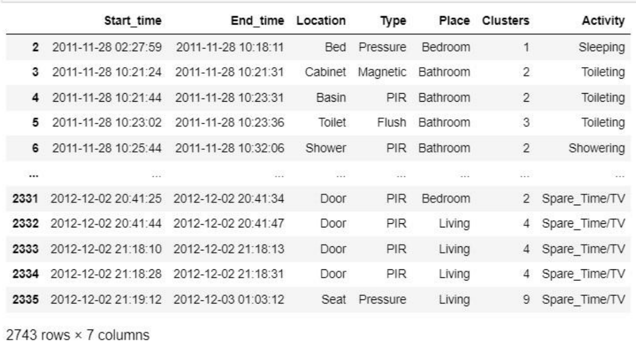
	Start_time	End_time	Activity
0	Start time	End time	Activity
2	2011-11-28 02:27:59	2011-11-28 10:18:11	Sleeping
3	2011-11-28 10:21:24	2011-11-28 10:23:36	Toileting
4	2011-11-28 10:25:44	2011-11-28 10:33:00	Showering
5	2011-11-28 10:34:23	2011-11-28 10:43:00	Breakfast
...	...	...	...
490	2012-12-02 19:10:00	2012-12-02 19:11:59	Toileting
491	2012-12-02 19:16:00	2012-12-02 20:03:59	Spare_Time/TV
492	2012-12-02 20:09:00	2012-12-02 20:21:59	Spare_Time/TV
493	2012-12-02 20:29:00	2012-12-02 20:29:59	Grooming
494	2012-12-02 20:41:00	2012-12-03 01:03:59	Spare_Time/TV

ADL-Label



Out[47]:

	Start_time	End_time	Location	Type	Place	Clusters
2	2011-11-28 02:27:59	2011-11-28 10:18:11	Bed	Pressure	Bedroom	1
3	2011-11-28 10:21:24	2011-11-28 10:21:31	Cabinet	Magnetic	Bathroom	2
4	2011-11-28 10:21:44	2011-11-28 10:23:31	Basin	PIR	Bathroom	2
5	2011-11-28 10:23:02	2011-11-28 10:23:36	Toilet	Flush	Bathroom	3
6	2011-11-28 10:25:44	2011-11-28 10:32:06	Shower	PIR	Bathroom	2
...	...	...	...	...	...	...
2331	2012-12-02 20:41:25	2012-12-02 20:41:34	Door	PIR	Bedroom	2
2332	2012-12-02 20:41:44	2012-12-02 20:41:47	Door	PIR	Living	4
2333	2012-12-02 21:18:10	2012-12-02 21:18:13	Door	PIR	Living	4
2334	2012-12-02 21:18:28	2012-12-02 21:18:31	Door	PIR	Living	4
2335	2012-12-02 21:19:12	2012-12-03 01:03:12	Seat	Pressure	Living	9



	Start_time	End_time	Location	Type	Place	Clusters	Activity
2	2011-11-28 02:27:59	2011-11-28 10:18:11	Bed	Pressure	Bedroom	1	Sleeping
3	2011-11-28 10:21:24	2011-11-28 10:21:31	Cabinet	Magnetic	Bathroom	2	Toileting
4	2011-11-28 10:21:44	2011-11-28 10:23:31	Basin	PIR	Bathroom	2	Toileting
5	2011-11-28 10:23:02	2011-11-28 10:23:36	Toilet	Flush	Bathroom	3	Toileting
6	2011-11-28 10:25:44	2011-11-28 10:32:06	Shower	PIR	Bathroom	2	Showering
...	...	...	...	...	...	...	...
2331	2012-12-02 20:41:25	2012-12-02 20:41:34	Door	PIR	Bedroom	2	Spare_Time/TV
2332	2012-12-02 20:41:44	2012-12-02 20:41:47	Door	PIR	Living	4	Spare_Time/TV
2333	2012-12-02 21:18:10	2012-12-02 21:18:13	Door	PIR	Living	4	Spare_Time/TV
2334	2012-12-02 21:18:28	2012-12-02 21:18:31	Door	PIR	Living	4	Spare_Time/TV
2335	2012-12-02 21:19:12	2012-12-03 01:03:12	Seat	Pressure	Living	9	Spare_Time/TV

2743 rows × 7 columns

Without Labels

PCA

With labels

```
Predicted   0   1    2   3    4   5   6   7   8   9
True
0           4   0   77   4   44  49   0   7   5   0
1           0   0   21   0   23  17   0   0   1   0
2           8   0   84  12   52  46   0  10  10   0
3           0   0   86   0   29  18   0   0   0   0
4           1   0   71   0   61  39   0   6   4   0
5           1   0    4   5    2   6   0   5   1   0
6           2  15   22   1   19  10  16   1   2   2
7           0   0   85   1   41  46   0   1   1   0
8          22   1  144  35  184  89   0  19  18  31
9           1   0   68   4   31  39   0   4   3   0
```

```python
print("Activity Labels:")
for i, activity in enumerate(le.classes_):
    print(f"{activity} -> {i}")
```

```
Activity Labels:
Breakfast -> 0
Dinner -> 1
Grooming -> 2
Leaving -> 3
Lunch -> 4
Showering -> 5
Sleeping -> 6
Snack -> 7
Spare_Time/TV -> 8
Toileting -> 9
```

encoding ,PCA analysis ,Without labels and confusion matrix plot.For document referred different documents and worked on overleaf.

Vinusha Garlapati:

Worked on data preprocessing for labels along with that worked on K-Means ,With labels and confusion matrix and overall accuracy.For document referred different IEEE papers and Kaggle.

Confusion Matrix

REFERENCES

[1] 1. "Activities of Daily Living (ADLs) Recognition Using Binary Sensors Dataset", UCI Machine Learning Repository, https://archive.ics.uci.edu/ml/datasets/Activities+of+Daily+Living+(ADLs)+Recognition

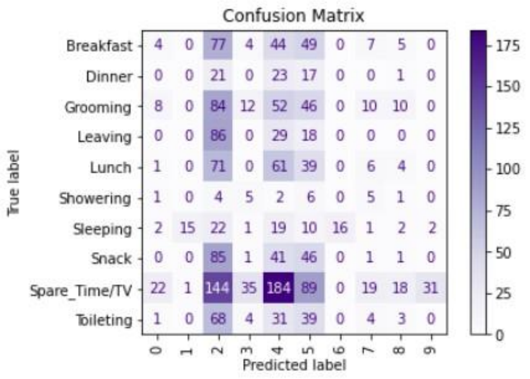
[2] IEEE Standard for Standard Data Mining Exchange (IEEE Std 1484.12009): This standard provides a framework for exchanging data mining models and results, including those generated through unsupervised learning techniques like clustering.

[3] IEEE Standard for Machine Learning Metadata (IEEE P1872): This standard provides a common metadata format for machine learning models and related artifacts, including unsupervised learning models like clustering.

[4] IEEE Transactions on Pattern Analysis and Machine Intelligence (IEEE TPAMI): This is a leading journal in the

Activity Labels

field of pattern analysis and machine intelligence, which includes research on

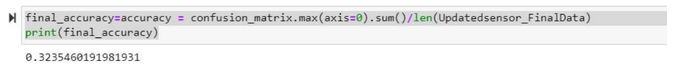


unsupervised learning techniques such as clustering.

[5] Bishop, Christopher M. Pattern recognition and machine learning. springer, 2006.

[6] Hastie, Trevor, Robert Tibshirani, and Martin Wainwright. Statistical learning with sparsity: the lasso and generalizations. CRC press, 2015

Plot for Confusion Matrix



```
final_accuracy=accuracy = confusion_matrix.max(axis=0).sum()/len(Updatedsensor_FinalData)
print(final_accuracy)

0.3235460191981931
```

Accuracy

ROLES AND RESPONSIBILITIES

Tejasree Kilari:

Worked on data preprocessing for sensors along with that worked on Feature extraction ,one hot