Indian Institute Of Technlogy, Delhi

Project Report on:

"Impact of Feature Engineering on the Performance of Machine Learning Model"

Submitted To:
Prof. Brejesh Lall,
Department of Electrical Engineering,
Indian Institute Of Technlogy, Delhi

Submitted By:
Kumar Utsav,
Roll No.: 21117056,
Semester: VI,
Department of Electrical Engineering,
National Institute of Technology, Raipur

CERTIFICATE

This is to certify that the minor project report entitled **"Impact of Feature Engineering on the performance of Machine Learning Model"**, submitted by **Kumar Utsav** is the bonafied work completed under my supervision and guidance during his research internship at Indian Institute of Technology, Delhi.

....
Prof. Brejesh Lall,
Department of Electrical Engineering,
Indian Institute of Technology, Delhi

ACKNOWLEDGEMENTS

The portion of success is brewed by the efforts put in by many individuals. It is constant support provided by people who give you the initiative, who inspire you at each step of your endeavor that eventually helps you in your goal.
I would like to express my sincere gratitude to **Prof. Brejesh Lall** for providing me with the opportunity to undertake this internship project.
I would also like to extend my heartfelt thanks to **Vinti Nayar Ma'am** for her continuous guidance and mentorship throughout the internship. Her knowledge, expertise, and patience have been instrumental in shaping my research and enhancing my understanding of the subject matter. I am grateful for her constant support, insightful discussions, and valuable suggestions that have significantly enriched my learning experience.

Kumar Utsav

ROLL NO: 21117056
6th Sem
B.Tech[Electrical Engineering]
National Institute of Technology Raipur

ABSTRACT

This internship report focuses on the analysis and preprocessing of datasets, along with the study of machine learning algorithms and neural networks for pattern recognition. The report begins with an exploration of the importance and limitations of datasets, followed by an examination of different types of data and techniques for converting categorical and continuous data. Additionally, correlation, covariance, and outlier detection methods are discussed, along with strategies for treating outliers. Feature scaling and the application of Principal Component Analysis (PCA) are also explored.

Chapter 3 introduces various machine learning algorithms, including regression and classification models such as linear regression, support vector regression, logistic regression, and support vector classification. The report further presents an internship project that analyzes the impact of feature engineering on the performance of different machine learning models, with a focus on logistic regression, support vector classification.The project involves analysis and preprocessing of the dataset, followed by prediction using the mentioned supervised machine learning algorithms.

Chapter 5 provides an introduction to neural networks, highlighting their working and their role in pattern recognition. The report concludes with a summary of the key findings and contributions of each chapter.

Overall, this internship report offers valuable insights into the pre-processing of datasets, the application of machine learning algorithms, and the fundamentals of neural networks. The practical project demonstrates the impact of feature engineering on model performance, while the exploration of neural networks expands the understanding of pattern recognition. The findings from this report contribute to the broader field of data analysis and machine learning, providing a foundation for further research and application.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Introduction

In the current age of fourth industrial revolution, the world has digitized rapidly. It has led to generation of enormous amount of data, such as Internet of Things (IoT) data, cybersecurity data, mobile data, business data, social media data, health data, etc. To have a accurate analysis of data, and developing automated solutions for it. Traditional programming algorithms are not sufficient to provide accurate automated solutions, hence machine learning algorithm have been created to intelligently give efficient analysis of data.

Rapid digitization has also led to considerable increase in the users using digital services. Ensuring a smooth quality of experience[QoE] has become an important field of study. It has also resulted in the huge amount of data relating to the user experience, and various other fields like marketing trends. To draw conclusions from this huge chunks of raw data, data analysis is required.

The field of data analysis heavily relies on preprocessing techniques to ensure the quality and reliability of data. Data preprocessing involves a series of steps such as cleaning, normalization, feature scaling, handling missing values, and outlier detection and treatment. These techniques are essential to enhance the accuracy and performance of subsequent analytical processes. During this internship, a comprehensive study of these preprocessing methods was conducted, exploring their theoretical foundations and practical implementations.

Additionally, the report presents an introductory research on neural networks, with a specific focus on pattern recognition algorithms. Neural networks have gained prominence in various fields for their ability to learn patterns and make accurate predictions. Understanding the principles behind neural networks and their application in pattern recognition is crucial for effective data analysis.

The outcomes of this internship report contribute to a deeper understanding of the importance of preprocessing techniques in data analysis and the potential of neural networks in pattern recognition tasks. The practical demonstration of these concepts through the project serves as a valuable learning experience, highlighting the effectiveness of preprocessing methods and the application of neural networks in real-world scenarios.

1.2 Need of project

The field of data analysis heavily relies on preprocessing techniques to ensure the quality and reliability of data. Data preprocessing involves a series of steps such as cleaning, normalization, feature scaling, handling missing values, and outlier detection and treatment. These techniques are essential to enhance the accuracy and performance of subsequent analytical processes. During this internship, a comprehensive study of these preprocessing methods was conducted, exploring their theoretical foundations and practical implementations.

Additionally, the report presents an introductory research on neural networks, with a specific focus on pattern recognition algorithms. Neural networks have gained prominence in various fields for their ability to learn patterns and make accurate predictions. Understanding the principles behind neural networks and their application in pattern recognition is crucial for effective data analysis.

1.3 Objective of Project

To showcase the practical implementation of the studied concepts, a project was undertaken. The project involved utilizing the patternet function in MATLAB, which facilitated the building and training of neural network models for pattern recognition tasks. By leveraging this function, the project demonstrated the network's ability to learn and recognize patterns within the provided data.

To highlight the objectives more precisely:

- To demonstrate the application of various steps involved in pre-processing of data.
- A close observation about how the computation time and accuracy changes with changes in dataset.
- To showcase why ITU-T reccomends use of discretized sacle for recording user opinion.
- Finally, an introductory study about the pattern recognition algorithm of neural network.

CHAPTER 2

Dataset Analysis

Before starting to create a machine learning model for existing dataset. It is important to analyze the dataset, to find the optimal model for automated solution. Dataset is the collection of data in various format stored in digital format. It is used for the training of the model

2.1 Importance of Dataset

[1] Machine Learning models are data-hungry and require a lot of data to create the best model or a system with high accuracy. In my study, I found out that both quality and quantity of data plays an important role.

From my understandings from various research papers on machine learning model creation, it is clear that dataset analysis and understanding plays an important role. As it helps in finding the optimal way of preprocessing it, to enhance it's quality. At the same time, choice of machine learning model for training and prediction becomes easier.

2.2 Limitation of Dataset

A quality dataset is a fundamental requirement for a machine learning model. But real world datasets are not very accurate and are messier. Therefore, during analysis, it is important to look for such limitations and handle it.

Following are the limitations that are found in real world dataset:

- Insufficient data: amount of data available is not enough to train the model.
- Bias and Human error: Tools used for data recording often leads to either human error or bias.
- Quality: Real world datasets are complex therefore is of low quality.
- Privacy and Compliance: Most sources do not share their data due to some privacy and compliance regulations.
- Data Annotations Process: Human intervention in labeling the data to improve quality often produces error.

2.3 Types of Data in a Dataset

Type of data in data analysis refers to the level of detail and precision with which the data is recorded.[2] There are two categories of data which is further of two types:

- Categorical Data:
  This type of data represents qualities and characteristics of a person or item. It consist of qualitative variables and are non numerical in nature. Hence, they are required to be taken care of before using it to train the model. As ML models generally take numerical input.

  It is further divided into two types:
  - Nominal Data: It is the simplest type of data. It classifies the data purely by labelling or naming values. There is no hierarchy associated with the labels.
    Examples: Marital Status, Gender, Eye Color.
  - Ordinal Data: It is similar to Nominal data but classifies data by introducing an order.There is hierarchy order or ranking associated with data representing it's relative importance.
    Examples: User feedback on a defined scale of 1 - 10, Letter grades, Ranking in a competition.

- Numerical Data: It is the quantitative data expressed in numerical values. It is the type of data that answers "how much," "how many," and "how often." types of questions about a dataset. It can be used for statistical manipulation and can be plotted on various type of graphs.

  It is further divided into two types:

- Discrete data: It is the numerical data which contain the values that fall under integers or whole numbers. These data can't be broken into decimal or fraction values.
  Examples: Number of students, Age, Class.
- Continuous Data: It is the type of numerical data which contain decimal values. The data in this variable type can take any real value within a range. It is complex type of data and contains meaningful insights.
  Examples: Height, Speed, Frequncy, Time taken to complete a task.

To have a deeper understanding on how to classify data. I used a dataset which is collected during the PoQeMoN project[3] where a crowd measurement platform is implemented to assess YouTube end user's QoE in different mobile environments (UMTS, HSPA, LTE, etc.).

Table 2.1: Feature Name with it's Data Type

Sr No	Feature Name	Data Type
01	id	discrete data
02	user_id	discrete data
03	QoA_VLCresolution	discrete data
04	QoA_VLCbitrate	continuous data
05	QoA_VLCframerate	continuous data
06	QoA_VLCdropped	discrete data
07	QoA_VLCaudiorate	continuous data
08	QoA_VLCaudioloss	discrete data
09	QoA_BUFFERINGcount	discrete data
10	QoA_BUFFERINGtime	discrete data
11	QoS_type	nominal data
12	QoS_operator	nominal data
13	QoD_model	nominal data
14	QoD_os_version	nominal data
15	QoD_api_level	discrete data
16	QoU_sex	nominal data
17	QoU_age	discrete data
18	QoU_Ustedy	ordinal data
19	QoF_begin	ordinal data
20	QoF_shift	ordinal data
21	QoF_audio	ordinal data
22	QoF_video	ordinal data

I classified feature columns in the above table using the understanding that I made researching about the four types of data in a dataset.
I classified data as:

- dicrete data: if data is numerical integer values and has no decimal values.
- continuous data: if data is numerical decimal values within a specific range.

- nominal data: if data show some association with type of label and has no hierarchy associated with it.
- ordinal data: if data contains values that represent hierarchy or ranking associated with data.

Distinguishing the data is an important step of data analysis as it helps to determine the method of statistical technique to be used to analyze it. Before training machine learning model with a dataset, it is very important to make sure that the range and type of data is processable by the model.

2.4 Converting Categorical Data to Numeric Data

Categorical data cannot be directly used to train a ML model, as they are non numeric. Hence, they are given a numerical form using[4]:

2.4.1 One Hot Encoding

It is the most common method to convert categorical data to a type of data that can be fed to machine learning model. It creates separate columns for each unique occurrence of a value in original feature. Then, marking the data values as either O, if that value is absent in the data or 1 if it is present in data. The resulting numeric data has no hierarchy associated with it.

The most important disadvantage of this method is, it increases the dimensionality of dataset. This may cause increased computaion time for training or overfitting issues in the model. Hence, this method needs to be used cautiously so that, model works with greater accuracy.

```python
#importing library
import pandas as pd

#loading dataset in pandas dataframe
data_raw = pd.read_excel('dataset.xlsx')

#Performing One Hot Encoding
one_hot_encoded_data = pd.get_dummies(data_raw, columns = columnlist)
one_hot_encoded_data.head()
```

2.4.2 Ordinal Encoding

In this method of categorical data handling, each unique data value in the feature column is associated to a unique integer. The order of integer represents the hierarchy or rank associated with the data. Machine learning algorithms understands and this relationship during training.

```python
# impporting required packages
from numpy import asarray
from sklearn.preprocessing import OrdinalEncoder

# define data
data = asarray([['red'], ['green'], ['blue']])

# define ordinal encoding
encoder = OrdinalEncoder()

# transform data to encded data
result = encoder.fit_transform(data)
result.head()
```

2.4.3 Dummy Variable Encoding

One hot encoding though is the simplest method but it increases the dimension of dataset. Also, it makes the input matrix of the dataset to become singular which means it cannot be inverted. Hence, linear regression coefficients cannot be calculated by linear algebra.
In dummy variable encoding above issues are taken care of, It is similar to one hot encoding but the number of variables created = n-1, if n is the number of unique values in the festure column.

```python
#importing library
import pandas as pd

#loading dataset in pandas dataframe
data_raw = pd.read_excel('dataset.xlsx')

#Performing One Hot Encoding
dummy_df = pd.get_dummies(data_raw, columns = columnlist, drop_first=
    True)
dummy_df.head()
```

2.5 Converting Continuous Data to Discrete Data

[5] Continuous data, though carries a lot of information but interpretation and visualization can be very difficult. Hence, converting it to discrete data simplifies the complexity.It helps in better analysis by categorizing data in several intervals.

The other reason for such conversion is certain analysis technique may require discrete data as input.For example, decision tree requires input data to be discrete. Discretizing continuous data often reduces the computational time. It is also called as binning the data.

The methods that I studied for binning the data:

2.5.1 Uniform Discretization:

This method preserves the probability distribution of each input variable and makes it discrete with the specified number of ordinal groups or label.

We can implement this method using sklearn library:

```
# performing a uniform discretization transform of the dataset
trans = KBinsDiscretizer(n_bins=10, encode='ordinal',
        strategy='uniform')
data = trans.fit_transform(data)
```

2.5.2 K-means Discretization:

This method first fits the K clusters for each of the input variable and then assign each observation to a cluster.

We can implement this method using sklearn library:

```
# perform a k-means discretization transform of the dataset
trans = KBinsDiscretizer(n_bins=3, encode='ordinal',
        strategy='kmeans')
data = trans.fit_transform(data)
```

2.5.3 Quantile Discretization:

This method splits the observations of each input variable into n groups such that each group have approximately equal number of observations for each input variable.

We can implement this method using sklearn library:

```
# perform a k-means discretization transform of the dataset
trans = KBinsDiscretizer(n_bins=10, encode='ordinal',
        strategy='quantile')
data = trans.fit_transform(data)
```

2.6 Correlation and Covariance in Dataset

[6] To analyse the relationship that feature columns have among themselves and

the relationship that exist between feature columns and target, we use correlation and covariance. This helps in removing the features that are not much related to the target variable.

2.6.1 Correlation:

Correlation is the measure that determines the degree to which two or more random variables move in sequence. It determines the above degree by comparing the movement of one variable to the movement of the other variable in a particular direction.
Mathematical Equation:

$$Corr(x,y) = \frac{Cov(x,y)}{\sqrt{std(x)} * \sqrt{std(y)}}$$

Corr = correlation
Cov = covariance
std = standard deviation

Corr(x,y) = 1 shows maximum linear correlation exist between x and y whereas a value = -1 show inverse relationship while a value = 0 shows no relation exist between x and y.

2.6.2 Covariance:

Covarience is the statistical measure that determines the relationship between two random variables such that a change in other variable reflects the change in one variable. The greater the number, the more reliant is the relationship.
Mathematical Equation:

$$Cov(x,y) = \frac{1}{N}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

Cov = covariance
$\overline{x}$ = mean of x
$\overline{y}$ = mean of y
N = number of data values

Positive covariance shows that both the variables are heading in the same direction. Negative covariance shows the shift in the values of variables is in the opposite direction.

2.7 Detecting the Outliers in the Dataset

[7] During my study about the preprocessing of data, I came to know about the term outliers. Detection and treatment of outliers plays a very important role in the preprocessing of data.

An outlier can be defined as the data value that is abnormally larger or smaller than the rest of the data values in the dataset. It can also be understood as the datavalues that do not follow the nature or the constraints of the dataset.

An outlier can happen for a various reasons such as variability in the dataset or due to experimental error/ human error.

Removal of outliers is very important as presence of outliers negatively affects the statistical analysis and the training process of a machine learning algorithm, thereby reducing its accuracy.

If the dataset is small, it is easier to treat outliers just by looking at the dataset. But, real datasets are not always small, we often have to work with larger datasets in which detection and removal of outliers requires robust method. I have discussed the three most commonly used method:

2.7.1 Using Graphical Analysis

In graphical analysis, we plot the data values of each of the feature column. If there are unusual large spikes or depressions in the plot, it shows the presence of outliers.

```python
# Outlier Detection using Visual Analysis:
import matplotlib.pyplot as plt

# Plotting the feature column
pre_data.plot(subplots=True, layout=(5,3))
plt.show()
```

2.7.2 Using Z-score Analysis

[8] The Z-score method works on the principle of three sigma rule or empirical rule of statistic. According to this rule, every observed data of the normal distribution should fall within three standard deviations of the mean. Thus, using three as the threshold value, we can quantify the number of outliers present in the dataset. It is only applicable to the dataset which is normally distributed.

```python
# Outlier Detection using  Z-score Analysis:
threshold = 3
```

```

# caluclating mean and standard deviation
mean = dataset[column_name].mean()
std  = dataset[column_name].std()

# Calculate the z-score for each data point in the feature column
z_scores = (pre_data[column] - mean) / std

#printing number of outliers
num_outliers = sum(abs(z_scores) > threshold)
print(num_outliers)
```

We can adjust the value of threshold as per the nature of data.

2.7.3 Using Inter-Quartile Range Analysis

[9] Unlike Z-score method, Inter-Quartile Range method can be used to detect outliers in a skewed dataset. IQR method uses the differece between the data at 25th percentile and the data at 75th percentile to determine the upper and lower limits for the dataset. Hence, finding the number of outliers present.

After finding the inter quartile range, we determine the lower limit by subtracting the product of IQR value and multiplier (generally, multiplier = 1.5) from data at at 25th percentile. While, upper limit is determined by adding the above pproduct to the data at 75th percentile.

```
# Outlier Detection using  IQR method:
multiplier = 1.5

# first quartile (Q1) and third quartile (Q3) of feature column
Q1 = pre_data[column].quantile(0.25)
Q3 = pre_data[column].quantile(0.75)

# Calculate the interquartile range (IQR)
IQR = Q3 - Q1

# Define the lower and upper bounds to identify outliers
lower_bound = Q1 - multiplier * IQR
upper_bound = Q3 + multiplier * IQR

# Count the number of outliers based on the lower and upper bounds
num_outliers = sum((pre_data[column] < lower_bound) | (pre_data[column]
    > upper_bound))
print(num_outliers)
```

The value of multiplier can be adjusted as per the need, to get the desired results.

2.8 Treating the Outliers

I came across two of the most commonly used ways to treat the outliers in the dataset. These methods are:

2.8.1 Trimming/Removing the outliers

In this method we remove the outliers from the dataset. This method is to be performed cautiously as the removal of data means a loss of information. Therefore, it is very necessary to check the percentage of data that we are removing. A loss of 1-2% data is permissible. Below I have provided the code to implement the same:

1. Using Inter-Quartile Range as the basis for defining boundaries of data: Value of multiplier is to be such that there is desirable deletion of data.

```
#Removing outliers based on IQR:
multiplier = 1.5

Q1 = dataset[column_name].quantile(0.25)
Q3 = dataset[column_name].quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - multiplier* IQR
upper = Q3 + multiplier* IQR

# Create arrays of Boolean values indicating the outlier rows
upper_array = np.where(data_processed_iqr[column_name]>=upper)[0]
lower_array = np.where(data_processed_iqr[column_name]<=lower)[0]

# Removing the outliers
data_without_outlier = dataset.drop(index=upper_array, inplace=True)
data_without_outlier = dataset.drop(index=lower_array, inplace=True)
data_without_outlier = data_without_outlier.reset_index(drop=True)
```

2. Using Z-score as the basis for defining boundaries of data: Value of threshold is to be determined such that the data loss is not more than 2%.

```
#Removing outliers based on z-Score:
threshold = 3

# caluclating mean and standard deviation
mean = dataset[column_name].mean()
std  = dataset[column_name].std()

# Calculate the z-score for each data point in the feature column
z_scores = (dataset[column] - mean) / std

```

```python
# Create arrays of Boolean values indicating the outlier rows
upper_array = np.where(z_scores > 3)[0]

# Removing the outliers
data_without_outlier = dataset.drop(index=upper_array, inplace=True)
data_without_outlier = dataset.reset_index(drop=True)
```

2.8.2 Mean/Median Imputation

Presence of outliers greatly infuences the value of mean in a dataset. In this method outliers are replaced by the mean or median of that column. This results is lesser loss of information. But, might decrease the accuracy of the model, by introducing bias.

```python
#Replacing outliers with mean:
mean = np.mean(data)
median = np.median(data)

# Calculate the absolute deviation from the median
abs_deviation = np.abs(data - median)
mad = np.median(abs_deviation)

# Calculate the modified Z-score
modified_z_scores = 0.6745 * abs_deviation / mad

outliers = data[modified_z_scores > threshold]
data_without_outliers = np.where(modified_z_scores > threshold, mean,
    data)
```

2.9 Feature Scaling or Transformation of Dataset

[10] The real world dataset are huge with data varying in large limits. Also, Dataset contains different feature column and each feature column will have it's own range of data values. This huge variation in the ranges of feature column not only incresases the computation time but also negatively affects the accuracy of the model.

To resolve this issue, we transform the datavalues of the dataset, also known as the feature scaling. It is the method by which the data is either standardized or normalized so that the feature column have similar scale and distribution

1. Advantages:

- It improves the quality of by removing inconsistencies from the dataset.
- It enables the integration of data from various sources, improving the accuracy and completeness of the data.

- It simplifies the process of analysis of data.
- Sensitive data can be hid from the society thus improving the security of data.
- It improves the prediction accuracy of the machine learning models.

2. Disadvantages:

- It is time consuming if the dataset is large.
- It is sometimes a bit complex to be implemented to a dataset and might require a strong understanding of data.
- It might result in the loss of data.
- It is a costlier method as it sometimes need high end hardwares and softwares.

Most commonly used feature scaling method that I come across while doing the research are as follows:

2.9.1 Min-Max Normalization

It is the simplest method of feature scaling.It applies linear transformation to the data resulting in scaled data varying between 0 and 1. It preserves the relative relationship between the data points and is suitable for the data of known and relative ranges. This method is not used generally in case of dataset with outliers.

Mathematical Formula used for transforming data values:

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Python Code:

```python
# Applying normalization
from sklearn.preprocessing import MinMaxScaler

# fit scaler on training data
norm = MinMaxScaler().fit_transform(dataset)
normalized_data = pd.DataFrame(norm, columns=dataset.columns)
normalized_data
```

2.9.2 Standardization/ Z-Score Normalization

It is the method of feature scaling that transforms data in a manner such that the mean of the data values in the feature columns of the dataset becomes 0 while the standard

deviation becomes 1. It helps in preserving the shape of the data, and is appropriate only if the dataset has no gaussian distribution.
Mathematical Formula used for transforming data values:

$$X_{new} = \frac{X - X_{mean}}{X_{std}}$$

Python Code:

```
# Applying standardization
from sklearn.preprocessing import StandardScaler

nume= StandardScaler().fit_transform(dataset)
standardized_data = pd.DataFrame(nume,columns=dataset.columns)
standardized_data
```

2.10 Principal Component Analysis(PCA)

[11] Dimensionality of dataset refers to the number of feature columns or to be very precise, the number of dimension required to visualize the dataset. A highly dimensional dataset must contain a large number of datapoints for ML model to perform accurately. As the number of features required increases, the number of datapoints required increases exponentially. It also makes visualization and analysis of data very difficult.

To overcome this issue of dimensionality, dimensions of the dataset is needed to be reduced with the least loss of information. Therefore, Principal component analysis, a dimensionality reduction method is used. Reducing the dimension though negatively affects the accuracy of model, still makes the visualization of the dataset easy. It drastically reduces the computation time.

2.10.1 Working of PCA

The number of principal components is equal to the number of feature column. Therefore, they are constructed in such a manner that the first principal component contains the maximum possible variance. Similarly, the second principal component is created but it should be perpendicular to first principal component, capturing the highest possible variance. While for the third principal component, it should be perpendicular to both previous component and capturing maximum possible variance. This is how all the principal components are created.

It will depend on the usecases that how many principal components are required for the analysis.

2.10.2 Steps for PCA Algorithm

1. Standardize the data: PCA requires the standardized data, therefore the first step is to standardize the data to ensure the mean = 0 and standard devaition = 1.

2. Calculate the covariance matrix: The next step is to calculate the covariance matrix of the standardized data.

3. Calculate eigen vectors and eigen values: The eigen vectors and eigen value of the covariance matrix are calculated. The eigenvectors represent the directions in which the data varies the most, while the eigenvalues represent the amount of variation along each eigenvector.

4.Choose the principal components: The principal components are the eigenvectors with the highest eigenvalues. These components represent the directions in which the data varies the most and are used to transform the original data into a lower-dimensional space.

5. Transform the data: The final step is to transform the original data into the lower-dimensional space defined by the principal components.

2.10.3 Advantages of PCA

- Dimensionality Reduction: It reduces the dimensionality with a least loss in the information.
- Feature Extraction: It is used for creating features derived from original data which is more understandable and insightful.
- Data Visualization: We can visualize the data by projecting the data onto the first few principal components, thereby producing more interpretable visualizations.
- Noise Reduction: PCA determines the underlying signal or pattern of data. Thus, helps in reducing the noise and errors.
- Handling Correlation between features: PCA results in the creation of new features called principal components that are orthogonal to each other and thus removes the collinearity among features.

2.10.4 Disadvantages of PCA

- Interpretability: The principal components produced after PCA are not always easy to interpret and correlate with the original features.
- Information Loss: In PCA, dimensionality is reduced by choosing a subset of the features and thereby results in loss of information.
- Presence of outliers and the dataset not scaled properly results in the creation of inaccurate principal component.

2.10.5 Python Code

```python
# Principal Component Analysis
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(standardized_data)
principal_components = pca.transform(standardized_data)
```

2.11 Summary

The above detailed explanation about the different aspects of dataset analysis, is my understanding that I got after researching about this topics. I required this understanding for a better analysis of the dataset that I was going to work on for my project.

The first and most important step is to determine the nature of dataset and the type of data points present in the dataset. The following observation will greatly influence the steps of data preprocessing. As, the data needs to be transformed to the numerical type. Then, the data is to be checked for presence of outliers. Removal of outliers is the following step along with the scaling of data.

This steps in data preprocessing are very important as they greatly influnence the accuracy as well as computation time of machine learning model.

CHAPTER 3

Machine Learning Algorithms

In the previous section, I decribed about the steps involved in the dataset preperation and it's importance. After preprocessing, algorithms are required, which can use this data to make predicitions for unknown datapoints. The computer programs that uses such algorithms is called Machine Learning model and the process of training of model is termed as machine learning.

This section will contain the various supervised machine learning algorithm that I studied about.

3.1 Introduction

[12] Machine Learning is a subset of artificial intelligences and is mainly concerned with the development of the algorithm that helps machine to learn from historical data on their own.

It is divided into two categories:

- Supervised Machine Learning Algorithms:
  In supervised machine learning algorithm, the model is provided with the labeled data to train it and using it as the basis, it predicts the output.

The main aim of the supervised machine learning algorithm is to map the input data to the output data. This mappings then helps to predict the output in case of unknown or new inputs.

It is further divided into two categories:

- Regression:
  Regression machine learning models are used in the cases when the datasets contains continuous data as the target variable. This models finds the underlying pattern or relationship of the different features to the target variable. It generally uses mathematical equations with weights multiplied to the datapoints and the resultant is summed up to predict the target variable.

- Classification: Classification machine learning models are used in the cases when the datasets contains categorical data or discrete data as the target variable. This models finds the underlying pattern or relationship of the different features that helps in determining the class or category the given features belong to. It uses probablistic approach to create model.

• Unsupervised Machine Learning Algorithms:
In unsupervised machine learning algorithm, the model is provided with the unlabeled data to explore and analyse the data, to determine patterns and relations without any target variable.

The main aim of the unsupervised machine learning algorithm is to restructure the input data into new features or a group of objects with similar patterns, without having a predetermined result. As my internship project doesn't requires the use of unsupervised machine learning algorithm, I haven't studied much about it.

3.2 Regression Models:

This section contains the information about the regression models that I gathered for my project.

3.2.1 Linear Regression

[13] Linear Regression is a machine learning model that models a relationships between the predictors and target variable by fitting a linear equation to the observed data.

Mathematical Formula:

$$Y = w * X + b \tag{3.1}$$

where, Y = target variable, X = input variable/predictor, w = weight, b = bias

The main aim is to find the values of w and b that produces the minimum deviation of the predicted target values and actual target values.

Cost function is used as the parameter to determine the degree of difference between the actual and predicted values. It is determined by the following formula:

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{(w,b)}(X_i) - Y_i)^2 \tag{3.2}$$

where, J = cost function, m = number of observations, $f_{(w,b)}(X_i)$ = predicted value, Y_i = actual value

To determine the values of w and b for which the cost function has minimum value, gradient descent algorithm is used. Gradien Descent algorithm has the following outline of working: 1. Assigning some values to the w and b at the starting. 2. Using the partial derivative of the cost function, values of w, b is updated in each iterations.

$$w = w - \alpha\frac{\partial J(w,b)}{\partial w} \tag{3.3}$$

$$b = b - \alpha\frac{\partial J(w,b)}{\partial b} \tag{3.4}$$

where α is the learning rate that determines the speed by which model converge to the minima of the cost function. Value of alpha should be chosen cautiously, because high learning rate may cause the model to skip the minima while converging and low learning rate may result in long computation time.

Python Code using sklearn library:

```python
from sklearn import linear_model
from sklearn.metrics import mean_squared_error,r2_score
reg = linear_model.LinearRegression()

#X_train and Y_training are the training predictor and targets
reg.fit(X_train,y_train)

# Make predictions on the test set
y_pred = reg.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

3.2.2 Support Vector Regression

[14] Support Vector Regression model works on the principle of support vector machines. Unlike SVM, SVR finds a hyperplane that can approximate a relationship which best fits the data in a continuous space.

It has the ability to handle non linear relationship by using the kernel function to map data to higher dimensional space.

Mathematical Formula:

$$Y = w * X + b \tag{3.5}$$

where, Y = target variable, X = input feature vector, w = weight vector, b = bias

The main idea of SVR is to find a hyperplane that lies within a certain margin around the actual target values. SVR aims to minimize the error while allowing a tolerance or epsilon around the true values.

Tolerance can be defined as the permissible deviation allowed between the actual target variables and predicted target variable.When training the model, data points that fall within the margin or have deviations smaller than the tolerance are considered correctly predicted.But, data points that fall outside the margin or have deviations larger than the tolerance contribute to the error and are penalized during the model optimization process.

To account for the tolerance, SVR introduces two additional parameters, epsilon (ε) and slack variables (ξ_i, ${\xi^*}_i$), which control the margin and the error tolerance:

- ξ_i represents the deviation of the predicted value from the true value for training examples above the hyperplane.
- ${\xi^*}_i$ represents the deviation of the predicted value from the true value for training examples below the hyperplane.

Optimization formula can be defined as:
Minimize:

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\left(\sum \xi_i + \sum \xi_i^*\right) \tag{3.6}$$

subject to:

$$y - \mathbf{w}^T\mathbf{x} - b \leq \varepsilon + \xi_i \tag{3.7}$$

$$\mathbf{w}^T\mathbf{x} + b - y \leq \varepsilon + \xi_i^* \tag{3.8}$$

C is a regularization parameter that determines the importance given to minimizing errors versus maximizing the margin.
With the help of iteration, equation 3.6 is repeated and (w,b) is updated, so that thee tolerance is minimum.

Python Code using sklearn library:

```python
from sklearn import svm
from sklearn.metrics import mean_squared_error,r2_score
svr = svm.SVR(kernel = 'linear')

#X_train and Y_training are the training predictor and targets
svr.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svr.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

3.3 Classification Models:

This section contains the information about the classification models that I gathered for my project.

3.3.1 Logistic Regression

[15] Logistic Regression is the statistical model also called as Logit model, is used for the classification task in machine learning.
It uses the pricnciple of probability as a measure to determine the class to which a particular set of features belong.

Logistic regression is divided into categories based on the number of classes it can deal with:

1. Binary Logistic Regression

Binary Logistic regression is used when we have to predict out of the two classes. Logistic regression determines the probability of a particular class based on the input features. To be precise, the model determines the probability that an input X belongs to the particular class.

The approach used is similar to that of linear regression, but the prediction made by the model instead of a continuous variable varying between large ranges, is actually transformed to a value between 0 and 1 using logistic function.

Mathematical Formula:

$$P = \frac{1}{1 + e^{-(wX+b)}} \tag{3.9}$$

where, P = target variable, X = input feature vector, w = weight vector, b = bias
The above formula is the sigmoid function

In case of linear regression, MSE is used as the cost function but, in case of logistic regression the mathematical formula is non linear. This results in non convex graph with many local minima. Therefore, during optimisation finding the global minima is quite difficult.

Thus, to overcome the above issue, log loss function is used.

$$J = \frac{1}{N}\sum_{i=1}^{N} -(Y_i * log(Y_i) + (1 - Y_i) * log(1 - Y_i)) \tag{3.10}$$

where, J = cost function, N = number of observations, Y_i = predicted value

To optimize the performance of model, gradient descent algorithm is used, as stated in eqn 3.3. Python Code using sklearn library:

```python
# import the necessary libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)

# Prediction
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Logistic Regression model accuracy (in %):", acc*100)
```

2. Multinomial Logistic Regression

It is that category of logistic regression that is used to solve multiclass classification problem.

Simple logistic Regression can be used for multiclass classification by using the following methods:

- One vs One Method:
  It is the least used method. It is less efficient as it uses n(n-1)/2 classifiers. For

classifications, it creates binary classification between each classes. This method has problem of ambiguous region, a condition where instances belong to different classes. Some classes are in none of the classes. It takes maximum computation time to train the classification model.
Python Code using sklearn library:

```python
    # import the necessary libraries
    from sklearn.linear_model import LogisticRegression
    from sklearn.multiclass import OneVsOneClassifier
    base_model = LogisticRegression()

    # Create an instance of the OneVsOneClassifier with the base
   model
    model = OneVsOneClassifier(base_model)

    # Fit the model to the training data
    model.fit(X_train, y_train)

    # Prediction
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print("Logistic Regression model accuracy (in %):", acc*100)

```

- One vs Rest Method:
  It is more efficient method as it uses n classifiers for multi class classification. In this method, one class is separated from the rest of the classes, and the same process repeats for the rest of the classes. This method also has the problem of ambiguous region, where a single instance is classiffied in multiple classes.
  Python Code using sklearn library:

```python
    # import the necessary libraries
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score

    # LogisticRegression using one vs rest
    clf = LogisticRegression(mult_class = 'ovr')

    clf.fit(X_train, y_train)

    # Prediction
    y_pred = clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print("Logistic Regression model accuracy (in %):", acc*100)

```

3. Softmax Function:

[16] The above discussed methods for multiclass classification using logistic regression is not effective. As these methods are computationally costlier. Therefore, to improve the accuracy of logistic regression for multiclass problems, softmax functions are used instead of logistic function. The softmax function is a mathematical function

commonly used in multinomial logistic regression and neural networks for converting a vector of real numbers into a probability distribution over multiple classes.

Mathematical Formula:

$$Z = wX + b$$

$$P_i = \frac{e^{Z_i}}{\sum_{j=1}^{N} e^{Z_j}} \tag{3.11}$$

where, P_i= probability of class i, X = input feature vector, w = weight vector, b = bias N= number of classses
The above formula is the softmax function.

It then uses the same cross entropy loss as the cost function for the model because the function is non linear and mean squared error cannot be used.

Use of eqn 3.10 to measure cross entropy loss and eqn 3.3 for optimization. Python Code using sklearn library:

```python
# import the necessary libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# LogisticRegression using one vs rest
clf = LogisticRegression(multi_class = 'multinomial', solver  = 'lbfgs'
    )

clf.fit(X_train, y_train)

# Prediction
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Logistic Regression model accuracy (in %):", acc*100)
```

3.3.2 Support Vector Classification

[17] Suppport vector classification is the machine algorithm that is used for both regression as well as classification. It works on the principle of support vector machine, It's implementation for regression problem is already explained in previous sections. It is supervised machine learning technique that is considered non paramateric because it relies on kernel function.

SVC works with a objective to find a hyperplane in a N-dimensional space that distinctly classifies the data points. The dimensions of hyperplane depends on the dimensions of the input feature. It chooses the best line for classification that has maximum margin. Basic terminologies related to support vector classification:

- Support Vector: The point closest to the hyperplane are called the support vectors.
- Margin: The distance between the support vector and the hyperplane.
- Hyperplane: Hyperplane is the decision boundary that is used to separate the data points of different classes in a feature space.

Types of support vector classifiers:

- Linear SVC: When the data is linearly separable, it is termed as linear SVC. It make use of linear kernel function.
- Non-Linear SVC: When the data is not lineraly separable, it is termed as non linear SVC. In such cases, polynomial or RBF kernel function is used.

SVC aims to find the hyperplane that maximises the margin while correctly classifying the training instances. The SVC finds this hyperplane by the use of mathematical formula that are as follows:

Minimize:

$$\frac{1}{2}\|\mathbf{w}\|^2 + C * \sum \xi_i \tag{3.12}$$

subject to:

$$y_i * (\mathbf{w}^T\mathbf{x_i} + b) \geq 1 - \xi_i \tag{3.13}$$

Here:

w = weight vector perpendicular to the hyperplane. b = bias term. ξ_i = slack variables that allow for misclassifications. C = regularization parameter that controls the trade-off between maximizing the margin and minimizing the training errors.

There are two application of Support Vector Classifiers:

1. Binary Class Classification

In this application, dataset with binary classes are classified using SVC. SVC consider one of the classes as postive while the other class as‘ negative. The classes are termed or labelled as positive or negative based on the relative position of the class with the hyperplane.The algorithm finds an optimal hyperplane that maximizes the margin between the classes.

To implement it using python code with sklearn library:

```python
# import the necessary libraries
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

```python

# Support Vector Classification
model = SVC(kernel='linear', C=1.0)

model.fit(X_train, y_train)

# Prediction
y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Support Vector model accuracy (in %):", acc*100)
```

2. Multiclass Classification

[18] Support Vector Classification is also used for the dataset having more than two classes. SVC utilizes two methods that is OvO (one vs one method) and OvR (one vs rest method) to extend binary classification to multiclass classification.

- One vs One Method:
  SVC constructs a binary classifier for every pair of classes, resulting in N*(N-1)/2 classifiers, where N is the number of classes. During prediction, each classifier votes for the class it predicts, and the class with the most votes is assigned to the instance.

- One vs Rest Method:
  SVC trains N binary classifiers, one for each class against the rest. Each classifier predicts the likelihood of an instance belonging to its corresponding class, and the class with the highest probability is assigned.

We can implement SVC using the sklearn library. Scikit-learn automatically selects the appropriate strategy based on the number of classes and other factors.

Python Code to implement SVC:

```python
    # import the necessary libraries
    from sklearn.svm import SVC
    from sklearn.metrics import accuracy_score

    # LogisticRegression using one vs rest
    model = SVC(kernel='linear', C=1.0)

    model.fit(X_train, y_train)

    # Prediction
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print("Support Vector model accuracy (in %):", acc*100)
```

We can change the kernel to ’poly’ or ’rbf’ subject to the performance of the model.

3.3.3 Summary

In this section, I have given the detailed description of the understanding that I had formed about the various machine learning techniques. Understanding the underlying mathematical implementation of the machine learning algorithms helps to choose the most appropriate ML model. It also helps in understanding the data in a much better way. I have discussed mostly about the supervised machine learning algorithms: Regression and Classification. As the dataset involved in my project can be trained better using these ML models.

CHAPTER 4

INTERNSHIP PROJECT: Impact of Feature Engineering on the performance of Machine Learning Model

In this section, I will briefly describe how I used my understandings made by me in the previous section to complete my internship project. In my internship project, I have used a publically available dataset to show the positive impact that feature engineering has on computation time of supervised machine learning model.

4.1 About the Dataset

In my project, I have used a realistic dataset obtained from Speed Video Global Operating Platform at Huawei[19]. SVGOP is a specific application of vMOS in mobile networks throughout the world. The dataset contains 89,266 samples with 12 features and 1 target. The features are the QoS parameters recorded during the streaming of the video, while the target is the score given by the viewers on the scale of 1 to 5.

The description and name of the varoius features along with the type of data is in Table 4.1.

Table 4.1: Feature Name with its meaning and data type

Sr. No	Feature Name	Meaning	Data type
1	average_playback_rate (kbps)	Average number of kilobits that the user's device downloads per second while playing video.	Discrete
2	total_DL_rate (kbps)	Total amount of data downloaded per second on the device.	Discrete
3	video_bitrate (kbps)	Amount of data transferred per second by the video.	Discrete
4	initial_max_DL_rate (kbps)	Maximum download speed allowed for video streaming.	Discrete
5	E2E_RTT (ms)	Time taken for a packet to travel from the source to destination and back again.	Discrete
6	initial_buffering_latency (ms)	Delay in the initial start of playback.	Discrete
7	initial_buffer_download_size (byte)	Initial downloaded data of the video.	Discrete
8	playing_time (ms)	Amount of time it takes for the video to play.	Discrete
9	playing_total_duration (ms)	Refers to the total duration of playback.	Discrete
10	stalling_times	Number of times the video stalled.	Discrete
11	stalling_duration (ms)	Duration of stalls in the video.	Discrete
12	stalling_ratio	Stalling duration divided by the playing total duration.	Continuous

From the above description of the dataset, I have the following observation:

- The dataset is large with data varying on huge ranges.
- The data values in the dataset are of the numeric type mostly belonging to the discrete type.
- The data values does not required to be converted to numeric type as they are already numeric.

Description about the target variable:

Table 4.2: Taget Name with its meaning and data type

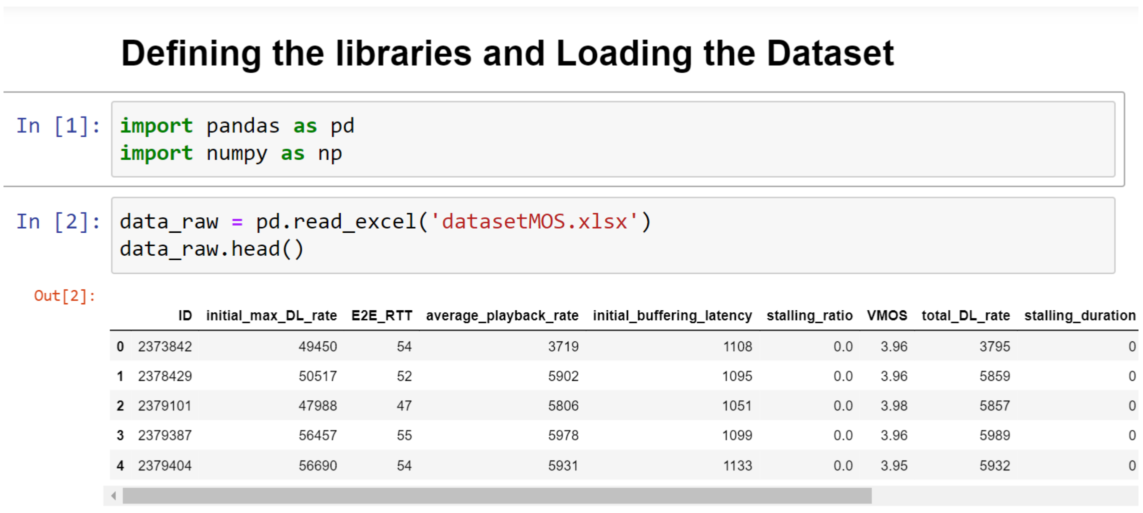
Sr. No	Target Name	Meaning	Data type
1	vMOS	User score based on viewing experience on a scale of 1 to 5	Continuous

4.2 Analysis and Pre-Processing of Dataset

The first step that I used in my project was to analyse the dataset properly. Then, preprocessing it to make it fit for the machine learning model. Here are the various steps with the detailed description that I followed for pre-processing. I have used jupyter notebook as my IDE, python as the programming language and open source python packages for preprocessing and training of ML model.

4.2.1 Loading the dataset

In the jupyter notebook, firstly I defined the required packages and then loaded the dataset. Here's the python code and the related output. [Fig 4.1]



Defining the libraries and Loading the Dataset

```
In [1]: import pandas as pd
        import numpy as np
```

```
In [2]: data_raw = pd.read_excel('datasetMOS.xlsx')
        data_raw.head()
```

Out[2]:

	ID	initial_max_DL_rate	E2E_RTT	average_playback_rate	initial_buffering_latency	stalling_ratio	VMOS	total_DL_rate	stalling_duration
0	2373842	49450	54	3719	1108	0.0	3.96	3795	0
1	2378429	50517	52	5902	1095	0.0	3.96	5859	0
2	2379101	47988	47	5806	1051	0.0	3.98	5857	0
3	2379387	56457	55	5978	1099	0.0	3.96	5989	0
4	2379404	56690	54	5931	1133	0.0	3.95	5932	0

Fig. 4.1: Loading Dataset

4.2.2 Understanding the Correlation of the Dataset

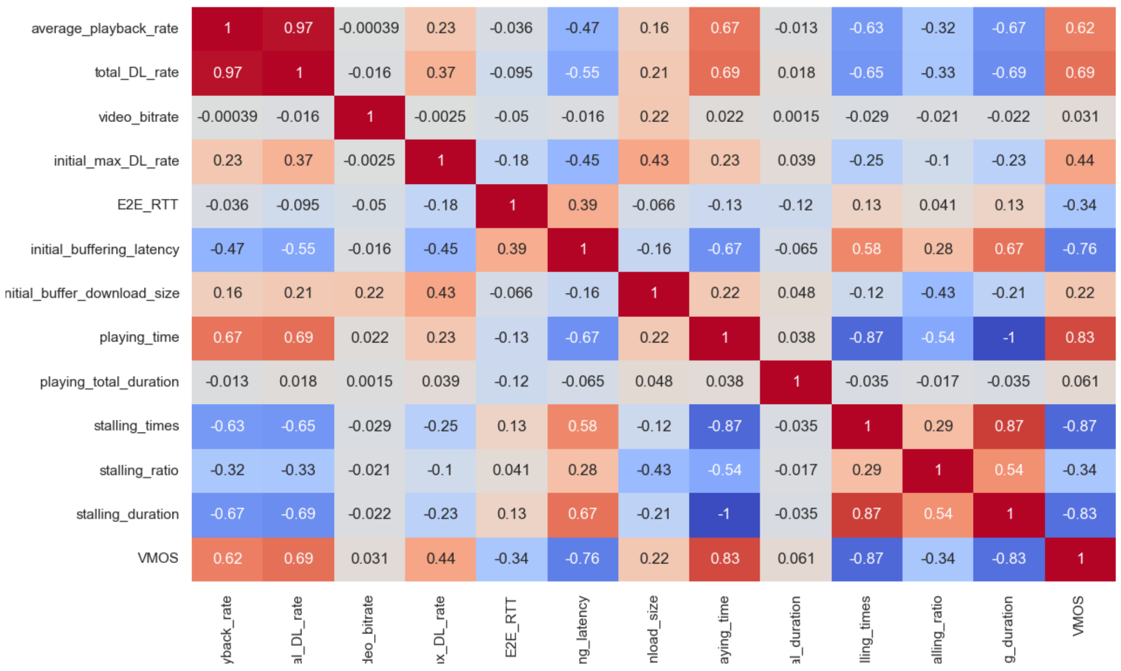
Using the pre-defined function of the pandas library and seaborn heatmap, I plotted the correlation of the feature among each other and with the target variable. Correlation will help me to determine the features that have strong correlation with the target variable and also if there are any inter-feature correlation.

In the Figure 4.2, the first part shows the python code. In the code, I have used the heatmap from the seaborn library to plot the correlations. The statement "pre_data.corr()" is used for determinig the correlation of the dataset. I have defined the underlying mathematics of the correlation in previous section: Section 2.6.

While the part following it, shows the correlation heatmap. The red shaded area with positive coefficients shows positive correlation or linear relationship existing. On the

Plotting Correlation Heatmap

```
In [9]: import seaborn as sns

        sns.set(rc = {'figure.figsize':(16,8)})
        heatmap = sns.heatmap(pre_data.corr(), annot = True, fmt='.2g',cmap= 'coolwarm')

        # Save the heatmap as an image file
        heatmap.figure.savefig('2.png')
```



	yback_rate	al_DL_rate	deo_bitrate	ix_DL_rate	E2E_RTT	ng_latency	nload_size	aying_time	al_duration	lling_times	alling_ratio	g_duration	VMOS
average_playback_rate	1	0.97	-0.00039	0.23	-0.036	-0.47	0.16	0.67	-0.013	-0.63	-0.32	-0.67	0.62
total_DL_rate	0.97	1	-0.016	0.37	-0.095	-0.55	0.21	0.69	0.018	-0.65	-0.33	-0.69	0.69
video_bitrate	-0.00039	-0.016	1	-0.0025	-0.05	-0.016	0.22	0.022	0.0015	-0.029	-0.021	-0.022	0.031
initial_max_DL_rate	0.23	0.37	-0.0025	1	-0.18	-0.45	0.43	0.23	0.039	-0.25	-0.1	-0.23	0.44
E2E_RTT	-0.036	-0.095	-0.05	-0.18	1	0.39	-0.066	-0.13	-0.12	0.13	0.041	0.13	-0.34
initial_buffering_latency	-0.47	-0.55	-0.016	-0.45	0.39	1	-0.16	-0.67	-0.065	0.58	0.28	0.67	-0.76
nitial_buffer_download_size	0.16	0.21	0.22	0.43	-0.066	-0.16	1	0.22	0.048	-0.12	-0.43	-0.21	0.22
playing_time	0.67	0.69	0.022	0.23	-0.13	-0.67	0.22	1	0.038	-0.87	-0.54	-1	0.83
playing_total_duration	-0.013	0.018	0.0015	0.039	-0.12	-0.065	0.048	0.038	1	-0.035	-0.017	-0.035	0.061
stalling_times	-0.63	-0.65	-0.029	-0.25	0.13	0.58	-0.12	-0.87	-0.035	1	0.29	0.87	-0.87
stalling_ratio	-0.32	-0.33	-0.021	-0.1	0.041	0.28	-0.43	-0.54	-0.017	0.29	1	0.54	-0.34
stalling_duration	-0.67	-0.69	-0.022	-0.23	0.13	0.67	-0.21	-1	-0.035	0.87	0.54	1	-0.83
VMOS	0.62	0.69	0.031	0.44	-0.34	-0.76	0.22	0.83	0.061	-0.87	-0.34	-0.83	1

Fig. 4.2: Correlation Heatmap

contrary, the blue shaded areas shows the negative correlation or the inverse relationship.

Observations made from the heatmap plot:

- The first important observation was few of the feature show strong correlation among itself. This will negatively impact the performance of our ML model. For Example: average_playback_rate and total_DL_rate.
- When features are highly correlated, it becomes challenging for the model to differentiate the individual effects of each feature on the target variable. This can lead to unstable coefficient estimates and decreased predictive accuracy.
- Therefore, the inter-feature correlation needs to be removed.
- Second important observation, is the presence of few feature which have very poor correlation with the target. For Example: video_bitrate.

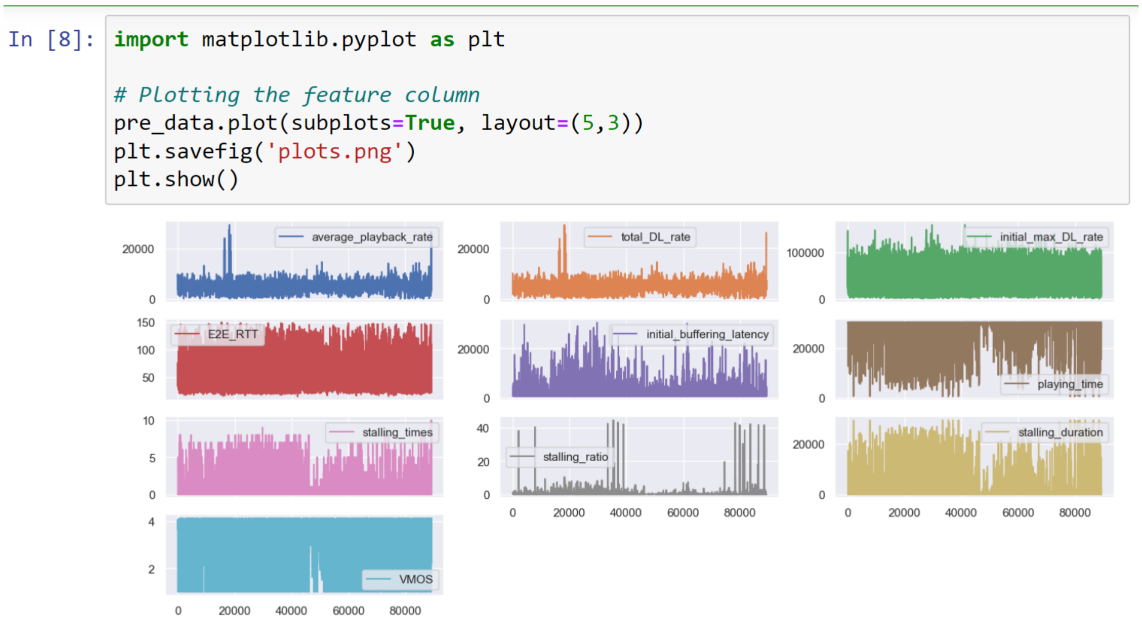
Removing the feature from the dataset that showed poor correlations with the target using the following code:

```python
# selecting Columns that showed significant correlation with VMOS
pre_data = pre_data.drop(['video_bitrate','initial_buffer_download_size
    ','playing_total_duration'], axis = 1)
```

Removal of poor correlated feature columns will result in reduced computational complexity. I will also decrease the computation time and will also prevent the problems like overfitting.

4.2.3 Detection & Removal of Outliers

The third step that I followed was to detect the presence of outliers and also to calculate the amount of data loss upon removal of outliers. There are several methods available for outlier detection, but I started with the graphical method. Fig 4.3 shows the



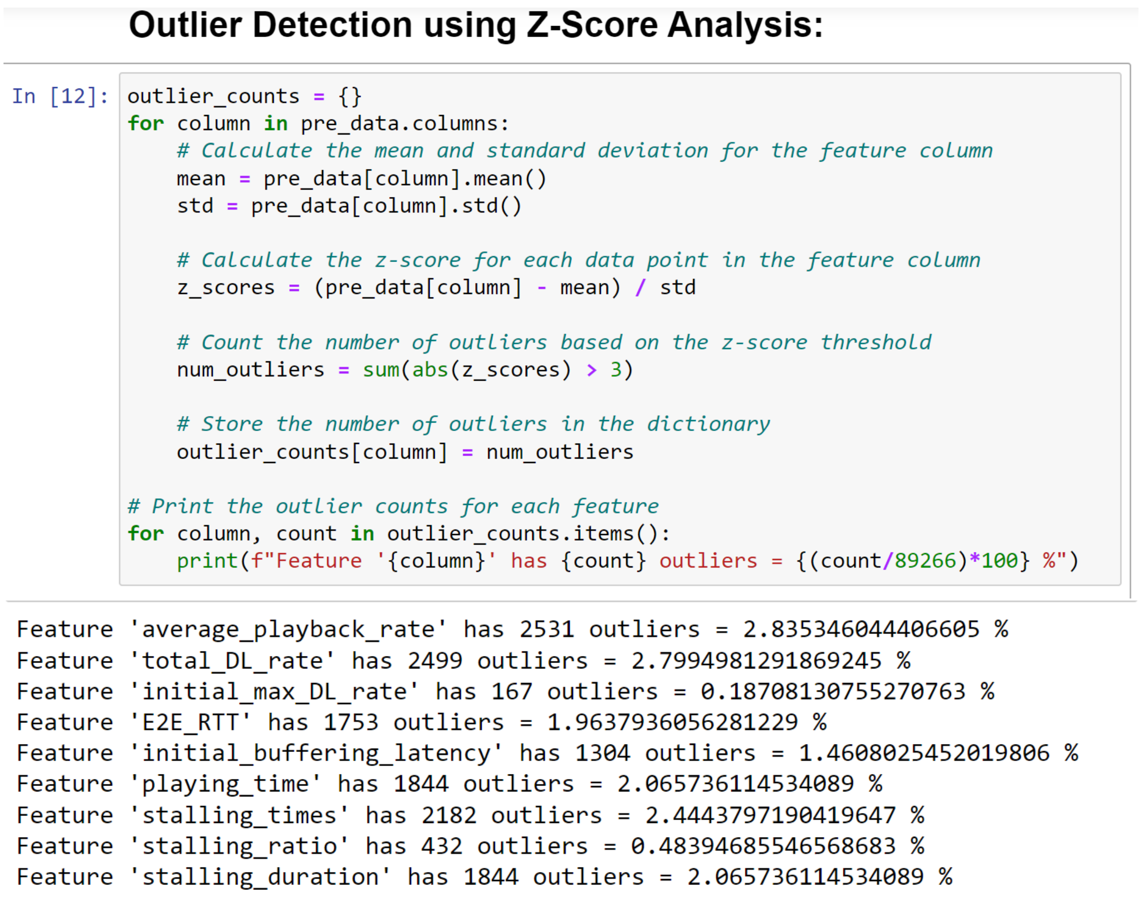
```
In [8]: import matplotlib.pyplot as plt

        # Plotting the feature column
        pre_data.plot(subplots=True, layout=(5,3))
        plt.savefig('plots.png')
        plt.show()
```

Fig. 4.3: Graphical plot of the Features

plot of all feature columns. From the figure, it is clear that the dataset contains outliers as there are unusual spikes in the distribution of the plot. For plotting, I used the matplotlib library. Refer to Section 2.7 for the detailed description.

Then, I calculates the amount of outliers present, firstly by using z-score analysis. While using the z-score analysis, I have used ±3 standard deviation as the boundary to determine the outliers. The use of threshold value of ±3 is because the normal distribution has 99% of the data within the 3 standard deviation. Fig 4.4 shows the amount of outliers present in each feature columns that is the number of data values are lying outside the range.

Secondly, I used Inter-Quartile Range method, to again determine the amount of



Outlier Detection using Z-Score Analysis:

```
In [12]: outlier_counts = {}
         for column in pre_data.columns:
             # Calculate the mean and standard deviation for the feature column
             mean = pre_data[column].mean()
             std = pre_data[column].std()

             # Calculate the z-score for each data point in the feature column
             z_scores = (pre_data[column] - mean) / std

             # Count the number of outliers based on the z-score threshold
             num_outliers = sum(abs(z_scores) > 3)

             # Store the number of outliers in the dictionary
             outlier_counts[column] = num_outliers

         # Print the outlier counts for each feature
         for column, count in outlier_counts.items():
             print(f"Feature '{column}' has {count} outliers = {(count/89266)*100} %")
```

```
Feature 'average_playback_rate' has 2531 outliers = 2.835346044406605 %
Feature 'total_DL_rate' has 2499 outliers = 2.7994981291869245 %
Feature 'initial_max_DL_rate' has 167 outliers = 0.18708130755270763 %
Feature 'E2E_RTT' has 1753 outliers = 1.9637936056281229 %
Feature 'initial_buffering_latency' has 1304 outliers = 1.4608025452019806 %
Feature 'playing_time' has 1844 outliers = 2.065736114534089 %
Feature 'stalling_times' has 2182 outliers = 2.4443797190419647 %
Feature 'stalling_ratio' has 432 outliers = 0.48394685546568683 %
Feature 'stalling_duration' has 1844 outliers = 2.065736114534089 %
```

Fig. 4.4: Outlier Detection using Z-score Analysis

outliers obtained in this method. I kept the value of multiplier as 1.5. The Fig 4.5 shows the python code used and the resulting calculated number of outliers.

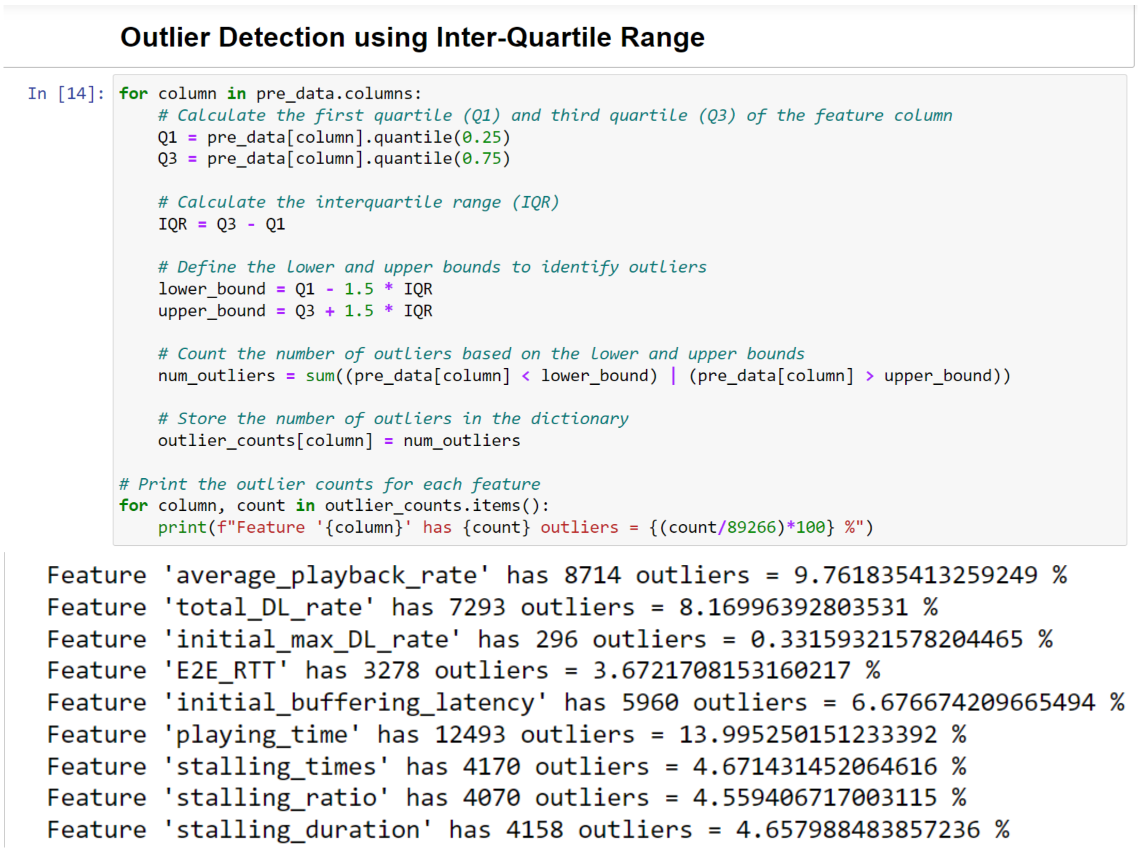
Observations:

- Using Z-score as a basis for detecting amount of outliers resulted in less number of outliers, while using IQR with multiplier = 1.5, the number of outliers detected is large.
- Therefore, using IQR for outliers removal will result in greater data loss. Hence, I will be using z-score method for outlier removal.

For the removal of outliers and to obtain the most ideal results, I experimented with both the methods, using different feature column as the basis in each case. After removal, I then plotted the correlation heatmap to see if the correlations have improved.

The python code that I used to create two dataframes one using the IQR method while the other using Z-score method.

```
#Using the IQR method
data_processed_iqr = pre_data.copy()
#Selecting feature column
```



Outlier Detection using Inter-Quartile Range

```
In [14]: for column in pre_data.columns:
             # Calculate the first quartile (Q1) and third quartile (Q3) of the feature column
             Q1 = pre_data[column].quantile(0.25)
             Q3 = pre_data[column].quantile(0.75)

             # Calculate the interquartile range (IQR)
             IQR = Q3 - Q1

             # Define the lower and upper bounds to identify outliers
             lower_bound = Q1 - 1.5 * IQR
             upper_bound = Q3 + 1.5 * IQR

             # Count the number of outliers based on the lower and upper bounds
             num_outliers = sum((pre_data[column] < lower_bound) | (pre_data[column] > upper_bound))

             # Store the number of outliers in the dictionary
             outlier_counts[column] = num_outliers

         # Print the outlier counts for each feature
         for column, count in outlier_counts.items():
             print(f"Feature '{column}' has {count} outliers = {(count/89266)*100} %")
```

```
Feature 'average_playback_rate' has 8714 outliers = 9.761835413259249 %
Feature 'total_DL_rate' has 7293 outliers = 8.16996392803531 %
Feature 'initial_max_DL_rate' has 296 outliers = 0.33159321578204465 %
Feature 'E2E_RTT' has 3278 outliers = 3.6721708153160217 %
Feature 'initial_buffering_latency' has 5960 outliers = 6.676674209665494 %
Feature 'playing_time' has 12493 outliers = 13.995250151233392 %
Feature 'stalling_times' has 4170 outliers = 4.671431452064616 %
Feature 'stalling_ratio' has 4070 outliers = 4.559406717003115 %
Feature 'stalling_duration' has 4158 outliers = 4.657988483857236 %
```

Fig. 4.5: Outlier Detection using IQR Analysis

```
column_name = 'initial_max_DL_rate'

Q1 = data_processed_iqr[column_name].quantile(0.25)
Q3 = data_processed_iqr[column_name].quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - 1.5* IQR
upper = Q3 + 1.5* IQR

# Create arrays of Boolean values indicating the outlier rows
upper_array = np.where(data_processed_iqr[column_name]>=upper)[0]
lower_array = np.where(data_processed_iqr[column_name]<=lower)[0]

# Removing the outliers
data_processed_iqr.drop(index=upper_array, inplace=True)
data_processed_iqr.drop(index=lower_array, inplace=True)
data_processed_iqr = data_processed_iqr.reset_index(drop=True)
data_processed_iqr.head()

#Removing Outliers based on Z-score method
data_processed_zsc = pre_data.copy()

#Selecting feature column
column = 'initial_max_DL_rate'
```

```python
mean = data_processed_zsc[column].mean()
std = data_processed_zsc[column].std()
z_scores = (data_processed_zsc[column] - mean) / std

# Create arrays of Boolean values indicating the outlier rows
upper_array = np.where(z_scores > 3)[0]


# Removing the outliers
data_processed_zsc.drop(index=upper_array, inplace=True)
data_processed_zsc = data_processed_zsc.reset_index(drop=True)
data_processed_zsc.head()
```

In the obseravtion below, for the IQR method I have taken multiplier as 7 and average playback rate as the basis for feature. It resulted in the removal of about 2.23% of data elements. While for the Z-score method, I have taken 3 as max limit for Z-score and average_playback_rate feature as basis which removed around 0.99% of data from dataset.

Table 4.3: Changes in correlation [with target] coefficients in two different methods

Sr. No	**Feature Name**	**Before Outlier Removal**	**Applying IQR method**	**Applying Z-score method**
1	average_playback_rate	0.62	0.33	0.67
2	total_DL_rate	0.69	0.45	0.74
3	initial_max_DL_rate	0.44	0.45	0.44
4	E2E_RTT	-0.34	-0.39	-0.34
5	initial_buffering_latency	-0.76	-0.8	-0.76
6	playing_time	0.83	0.77	0.83
7	stalling_times	-0.87	-0.78	-0.87
8	stalling_duration	-0.34	-0.47	-0.34
9	stalling_ratio	-0.83	-0.77	-0.83

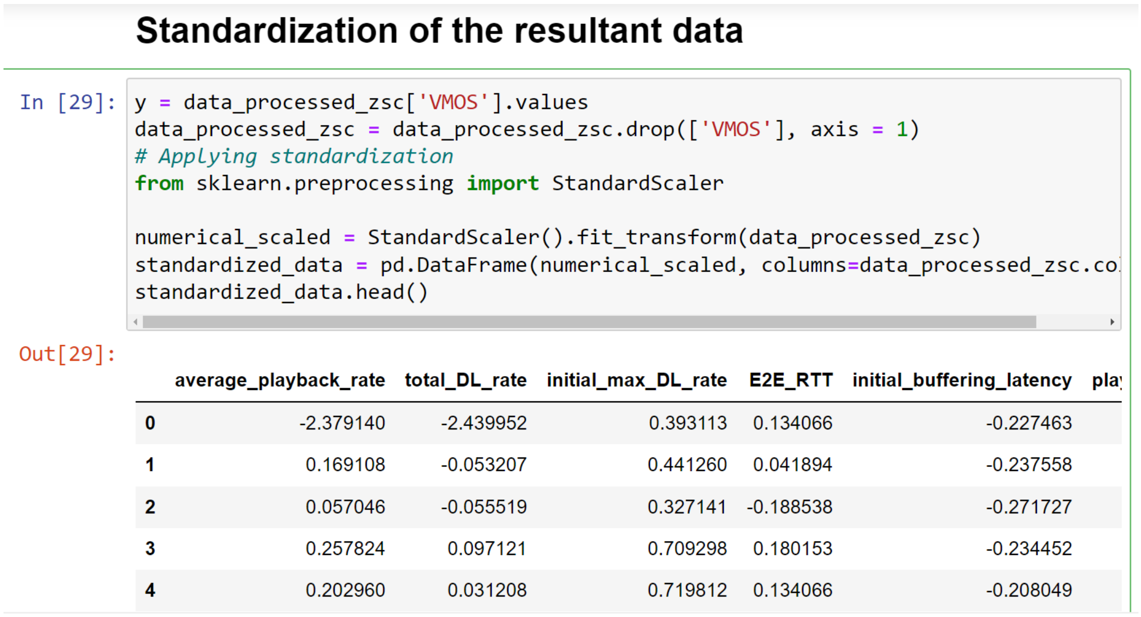
Hence, I made the following inferences on the basis of the above observation:

- Firstly, removal of outlier using IQR method resulted in high data loss, which also negatively affected the correlation with the target variable.
- Using Z-score method not only improved the correlations but also resulted in very less loss of data.
- Among all the feature columns, average_playback_rate showed the most promising results among all the features.
- Hence, I will be using z-score method and average_playback_rate as the feature basis for removal of outlier.

4.2.4 Standardization of Data

From the Fig 4.3, it is very evident that the data values are very heterogenic in nature. The data values in the dataset vary over large ranges which will decrease the accuracy of ML model. Also, standardization rescales the variables to a common scale, typically with a mean of zero and a standard deviation of one. This process eliminates the inherent scale differences, allowing for fair and meaningful comparisons between variables. This process ensures that each variable contributes equally to the learning process and avoids biased results.Refer to Section 2.9, for the detailed description about it.

Also, In the following steps, I will be using Principal Component Analysis and it requires the dataset to be standardized.



Standardization of the resultant data

In [29]:
```
y = data_processed_zsc['VMOS'].values
data_processed_zsc = data_processed_zsc.drop(['VMOS'], axis = 1)
# Applying standardization
from sklearn.preprocessing import StandardScaler

numerical_scaled = StandardScaler().fit_transform(data_processed_zsc)
standardized_data = pd.DataFrame(numerical_scaled, columns=data_processed_zsc.co
standardized_data.head()
```

Out[29]:

	average_playback_rate	total_DL_rate	initial_max_DL_rate	E2E_RTT	initial_buffering_latency	pla
0	-2.379140	-2.439952	0.393113	0.134066	-0.227463	
1	0.169108	-0.053207	0.441260	0.041894	-0.237558	
2	0.057046	-0.055519	0.327141	-0.188538	-0.271727	
3	0.257824	0.097121	0.709298	0.180153	-0.234452	
4	0.202960	0.031208	0.719812	0.134066	-0.208049	

Fig. 4.6: Python Code for Standardization

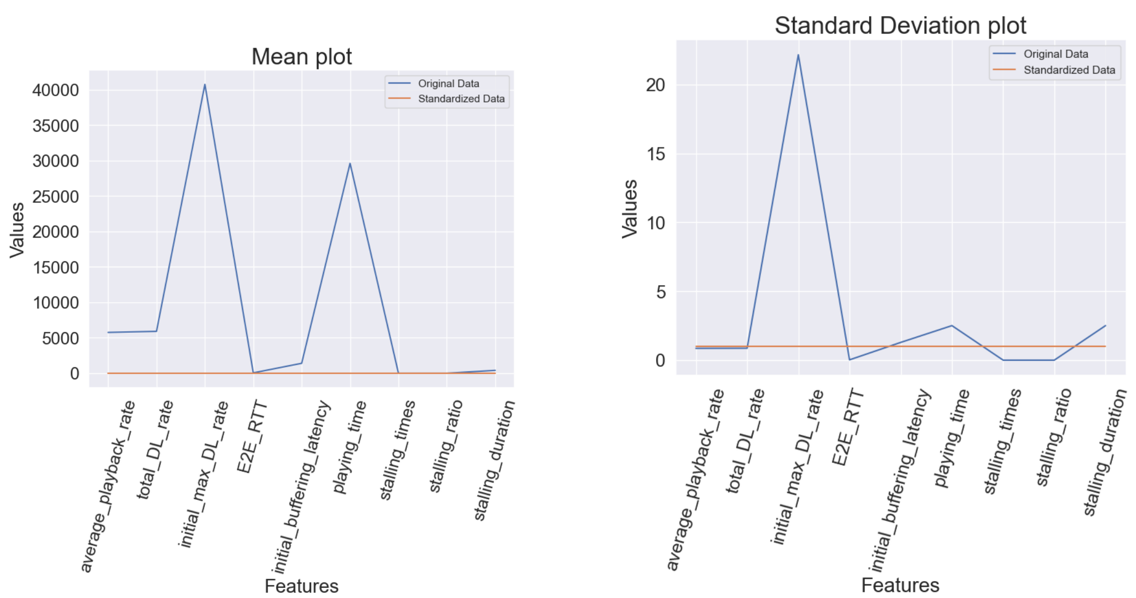
Now, to demonstrate the changes that happened in the dataset due to standardization, I have used the following python code:

```
# To calculate the mean and the standard deviation of each feature
    before and after the standardization
mean_arr = []
std_arr = []
org_mean_arr = []
org_std_arr = []
for column in standardized_data.columns:
    #Standardized Data mean and standard devaition
    mean = standardized_data[column].mean()
    std = standardized_data[column].std()
    mean_arr.append(mean)
    std_arr.append(std)
```

```python

    #Original Data mean and standard devaition
    mean = data_processed_zsc[column].mean()
    std = data_processed_zsc[column].std()
    org_mean_arr.append(mean)
    org_std_arr.append(std)
```

Here’s the python code that I used for plotting the data:

```python
# Plot of Mean
plotdata = pd.DataFrame({
    "Original Data":org_mean_arr,
    "Standardized Data":mean_arr},
    index=x)

plotdata.plot(figsize=(8, 6))
plt.title("Mean plot", fontsize=24)
plt.xlabel("Features", fontsize=20)
plt.ylabel("Values", fontsize=20)

plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.xticks(rotation = 75)
plt.savefig("5.1.png", bbox_inches='tight')
plt.show()
```



(a) Mean Plot (b) Standard Deviation Plot

Fig. 4.7: Changes in the Dataset after standardization

```python
# Plot of Standard Deviation
org_std_arr = [x / 1000 for x in org_std_arr]
# divided the values of original standard deviation by 1000 to
    accomodate the plot and highlight the difference
```

```python

plotdata = pd.DataFrame({
    "Original Data":org_std_arr,
    "Standardized Data":std_arr},
    index=x)

plotdata.plot(figsize=(8, 6))
plt.title("Standard Deviation plot", fontsize=24)
plt.xlabel("Features", fontsize=20)
plt.ylabel("Values", fontsize=20)

plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.xticks(rotation = 75)
plt.savefig("5.2.png", bbox_inches='tight')
plt.show()
```

From the Fig 4.7, it is quite clear that standardization helps in making the data uniform. The resultant dataset has mean = 0, while standard deviation = 1.

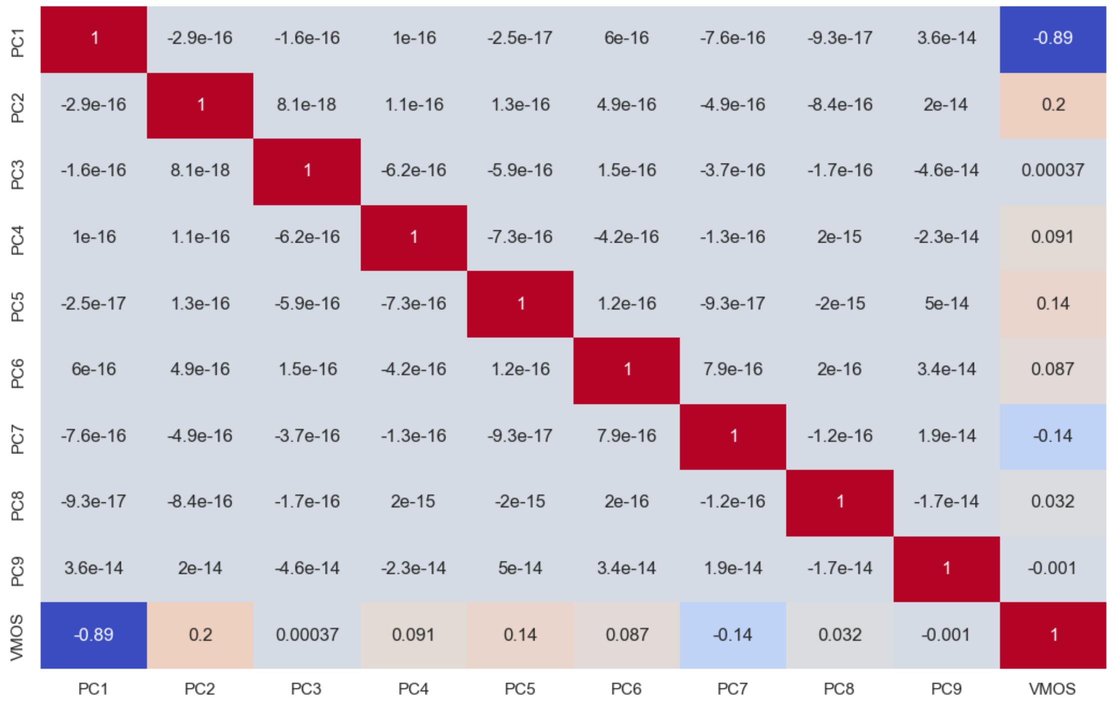
4.2.5 Applying PCA on the Dataset

From the observations made from correlation heatmap in Fig 4.2, the dataset show high correlated feature. Presence of correlated features increases the computation time and reduces the accuracy of prediction model. As it creates confusion in the process of assigning weights during training.

In order to treat the high correlations, I applied principal component analysis as it will result in the principal components which are the new features and are having zero inter-correlation. Refer to Section 2.10, for the detailed underlying math and the various steps involved in the principal component analysis. For my project, I have used the PCA from sklearn library. Here's the python code and the resulting correlation heatmap.

```python
pc_names = ['PC'+str(i) for i in range(1,10)]

# Principal Component Analysis
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(standardized_data)
principal_components = pca.transform(standardized_data)
explained_variance_ratio = pca.explained_variance_ratio_
result_df = pd.DataFrame(principal_components, columns=pc_names)

target = pd.Series(y, name='VMOS')
result_df = pd.concat([result_df, target], axis=1)
```

```
result_df
```



	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	VMOS
PC1	1	-2.9e-16	-1.6e-16	1e-16	-2.5e-17	6e-16	-7.6e-16	-9.3e-17	3.6e-14	-0.89
PC2	-2.9e-16	1	8.1e-18	1.1e-16	1.3e-16	4.9e-16	-4.9e-16	-8.4e-16	2e-14	0.2
PC3	-1.6e-16	8.1e-18	1	-6.2e-16	-5.9e-16	1.5e-16	-3.7e-16	-1.7e-16	-4.6e-14	0.00037
PC4	1e-16	1.1e-16	-6.2e-16	1	-7.3e-16	-4.2e-16	-1.3e-16	2e-15	-2.3e-14	0.091
PC5	-2.5e-17	1.3e-16	-5.9e-16	-7.3e-16	1	1.2e-16	-9.3e-17	-2e-15	5e-14	0.14
PC6	6e-16	4.9e-16	1.5e-16	-4.2e-16	1.2e-16	1	7.9e-16	2e-16	3.4e-14	0.087
PC7	-7.6e-16	-4.9e-16	-3.7e-16	-1.3e-16	-9.3e-17	7.9e-16	1	-1.2e-16	1.9e-14	-0.14
PC8	-9.3e-17	-8.4e-16	-1.7e-16	2e-15	-2e-15	2e-16	-1.2e-16	1	-1.7e-14	0.032
PC9	3.6e-14	2e-14	-4.6e-14	-2.3e-14	5e-14	3.4e-14	1.9e-14	-1.7e-14	1	-0.001
VMOS	-0.89	0.2	0.00037	0.091	0.14	0.087	-0.14	0.032	-0.001	1

Fig. 4.8: Correlation Heatmap of Principal Component

From the Fig 4.8, I have successfully got rid of the interfeature correlation and also the first 2 principal components shows the best correlation with the target. Thus, I need not to use all the principal components, instead use first 4 or 5 principal components for predictions. Hence, it also helped in reducing the dimension of the data set.

To understand the principal components more, I looked into variance captured by each principal component. I used:

```
explained_variance_ratio
```

command to get the variance ratio. I got the following array as output:

array([5.68620171e-01, 1.39228849e-01, 1.00272381e-01, 8.40406572e-02, 6.05502152e-02, 3.48431215e-02, 1.03952419e-02, 2.04872539e-03, 6.37194513e-07])

Thus, the first principal component captures most of the variance and the variance ration keeps on decreasing.

4.2.6 Discretization of Target Variable

The target variable in the dataset is continuous and therefore I have to discretize it. This will help me to use the dataset for training of both regression and classification model. For regression, I will use the continuous target variable while for the classification model, I will usee the discretized target variable.

This is the final step in the process of the preparation of dataset. have already discussed about the discretiazation methods and their codes. Hence, I used three types of discretizer to create three types of dicretized target variable, so that I can find out the discretizing method that performs the best in prediction.

Here's the python code:

```python
from sklearn.preprocessing import KBinsDiscretizer

# perform a k-means discretization transform of the dataset
trans = KBinsDiscretizer(n_bins=5, encode='ordinal', strategy='uniform'
    )
data = trans.fit_transform(y_2d)

# convert the array back to a dataframe
dataset = pd.DataFrame(data)
target_unf=dataset[0]
target_unf
```

Note: for other two discretized target variable I used the same python code and replaced the value of strategy with "kmeans" and then with "quantile".

By the end of this steps, I now have 4 sets of processed datasets for the training of machine learning model. I will be using this dataset and also the unprocessed dataset for training to highlight the objective of my project.

4.3 Prediction using Supervised Machine Learning Algorithm

In the last section of my project, I prepared various datasets which are at different stages of pre-processing to experiment with the performance of the model in different so that, the most optimal way of pre-processing and prediction. To explain in detail, I initially used two classification models:

- Logistic Regression
- Support Vector Classification

In each case, I used different parameters to define the model and used different number of principal components to find the most optimum parameters and the number of principal components.

Then, I experimented with the previously created datasets that is:

- Standardized Data
- Data without Outliers
- Raw Data

I did the classification in each case using the two models to highlight how pre-processing affects the performance in each case.

I have divided this section in three subcategories, first will contain all the python code I used, second subcategory will have the observations I made, third subcategory will have the conclusions that I drew after this experiment.

Here's the first subcategory containing the python code that I used:

4.3.1 Prediction Model: Logistic Regression

```
# Defining the Logistic Regression Model.
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter=8000,solver='liblinear',
    multi_class = 'ovr')
```

Note: It is the code I used for first case, in the following cases, I replaced the value of solver with 'sag' and 'saga' while the value of multiclass replaced with 'multinomial'. As the various parameters are associated with different optimization algorithms.

a. Using Final Processed Data

```
# Defining the predictors
predictors = result_df[['PC1','PC2','PC3', 'PC4', 'PC5', 'PC6']]
predictors.head()

#Splitting the dataset into training and testing
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(predictors,
    target_unf, test_size = 0.2)

#Model training
%%time
model.fit(X_train, y_train)

#Determining the accuracy of Model
model.score(X_test, y_test)
```

Note: For my experimentation, I ineterfered with the number of principal components that was being used for training.

b. Using Standardized Data

```python
# Defining the predictors
predictors = standardized_data
predictors.head()

#Splitting the dataset into training and testing
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(predictors,
    target_unf, test_size = 0.2)

#Model training
%%time
model.fit(X_train, y_train)

#Determining the accuracy of Model
model.score(X_test, y_test)
```

c. Using Data Without Outliers

```python
# Defining the predictors
predictors = data_processed_zsc
predictors.head()

#Splitting the dataset into training and testing
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(predictors,
    target_unf, test_size = 0.2)

#Model training
%%time
model.fit(X_train, y_train)

#Determining the accuracy of Model
model.score(X_test, y_test)
```

d. Using Raw Data

```
# Defining the predictors
predictors = data_raw.iloc[:,:-1]
predictors.head()

#Splitting the dataset into training and testing
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(predictors,
    target_unf, test_size = 0.2)

#Model training
%%time
model.fit(X_train, y_train)

#Determining the accuracy of Model
model.score(X_test, y_test)
```

This is the bunch of code I used for my experimentation with logistic regression model.

4.3.2 Prediction Model: Support Vector Classification

```
from sklearn import svm
classifier = svm.SVC(kernel = 'rbf')
```

Note: It is the code I used in first case, for the second case I replaced the value of kernel with 'poly'.

a. Using Final Processed Data

Refer to subsection (a) of previous section, as I used the same code just replaced model with classifier.

b. Using Standardized Data

Refer to subsection (b) of previous section, as I used the same code just replaced model with classifier.

c. Using Data without Outliers

Refer to subsection (c) of previous section, as I used the same code just replaced model with classifier.

d. Using Raw Data

Refer to subsection (d) of previous section, as I used the same code just replaced model with classifier.

I used regression models only on the processed data with continuous target variable:

4.3.3 Prediction Model: Linear Regression

```python
#Defining the linear Regression model
from sklearn import linear_model
from sklearn.metrics import mean_squared_error,r2_score
reg = linear_model.LinearRegression()

#Defining the predictors
predictors = result_df[['PC1','PC2','PC3', 'PC4', 'PC5', 'PC6']]
predictors.head()

#Splitting the dataset into training and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(predictors,y,
    test_size = 0.2)

#Model Training
reg.fit(X_train,y_train)

# Make predictions on the test set
y_pred = reg.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

4.3.4 Prediction Model: Support Vector Regression

```python
#Defining the support regression model
from sklearn import svm
from sklearn.metrics import mean_squared_error,r2_score
svr = svm.SVR(kernel = 'linear')

#Defining the predictors
predictors = result_df[['PC1','PC2','PC3', 'PC4', 'PC5', 'PC6']]
```

```python
predictors.head()

#Splitting the dataset into training and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(predictors,y,
    test_size = 0.2)

#Model Training
svr.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svr.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

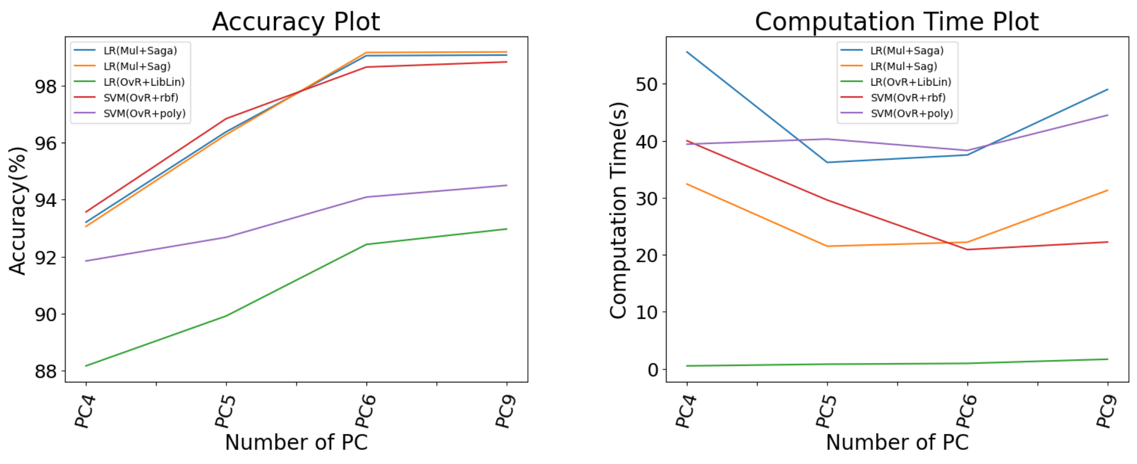
4.3.5 Observation Tables

This section contains the table that I created using the results I obtained by running the python code. The table depicts how the perofrmance abd computation time is affected with the change in dataset.

Table 4.4: Computation Time & Accuracy for different number of Principal Component

Number of PC	**Machine Learning Model**				
	Logistic Regression (Multinomial + Saga)	**Logistic Regression (Multinomial + Sag)**	**Logistic Regression (OvR + liblinear)**	**SVM (OvR + rbf)**	**SVM (OvR + poly)**
PC4	55.58 s	32.4 s	0.5 s	40 s	39.4 s
	93.21%	93.06%	88.17%	93.57%	91.85%
PC5	36.2 s	21.5 s	0.80 s	29.6 s	40.3 s
	96.38%	96.28%	89.92%	96.84%	92.68%
PC6	37.5 s	22.2 s	0.94 s	20.9 s	38.3 s
	99.05%	99.16%	92.43%	98.65%	94.09%
PC9	49 s	31.3 s	1.67 s	22.23 s	44.8 s
	99.07%	99.18%	92.97%	98.83%	94.50%

From the observation table, it can be clearly seen that number of components greatly affects the performance of machine learning model. The choice of parameters also play a very important role in defining the machine learning model. As, in the observation table, the parameters not only affects the computation time as well as the accuracy. Using the observation table, I plotted the data obtained the following graph:



(a) Accuracy Plot (b) Computation Time Plot

Fig. 4.9: Observation Plot for different principal component

Upon seeing the above figure, it is quite evident that the accuracy increases for a while for increasing number of principal components and then saturates to a particular value. This means increases to a spicific value and does not increase more for increasing number of principal components.

While in the case of computation time, the time first decreases for increasing number of principal component achieves a minima then start to increase. According to me, the optimum number of principal components is 6 and the best machine learning model is Logistic Regression with parameter multinomial and solver saga.

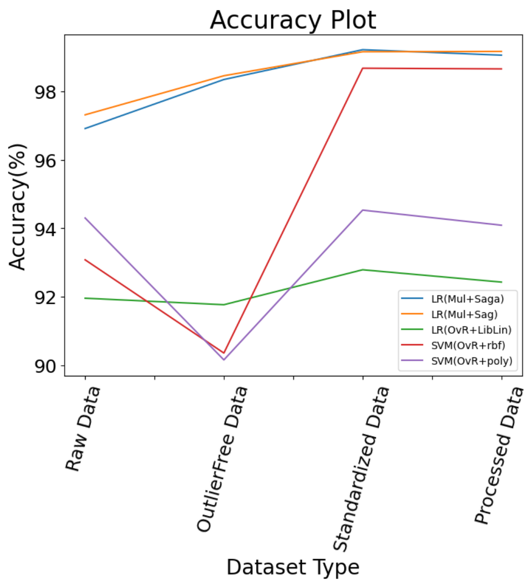
I experimented with unprocessed dataset and here are my observation for the unprocessed dataset.

Table 4.5: Computation Time & Accuracy for unprocessed Dataset

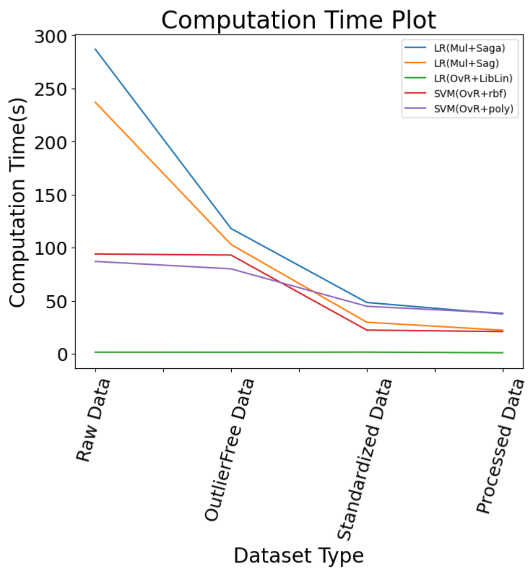
Dataset Type	**Machine Learning Model**				
	Logistic Regression (Multinomial + Saga)	**Logistic Regression (Multinomial + Sag)**	**Logistic Regression (OvR + liblinear)**	**SVM (OvR + rbf)**	**SVM (OvR + poly)**
Processed Dataset	37.5 s	22.2 s	0.94 s	20.9 s	38.3 s
	99.05%	99.16%	92.43%	98.65%	94.09%
Standardized Dataset	48.3 s	29.7 s	1.52 s	22.3 s	44.7 s
	99.21%	99.15%	92.79%	98.67%	94.53%
OutlierFree Dataset	118 s	103 s	1.42 s	93 s	80 s
	99.34%	99.45%	92.77%	90.36%	90.16%
Raw Dataset	287 s	237 s	1.48 s	91 s	37 s
	96.91%	97.31%	97.96%	93.08%	94.30%

From the datapoints of the observation, there is no much difference created by preprocessing in the accuracy of logistic regression. For the support vector machine, the pre-processing has greatly improved the performance of model.

The significance of pre-processing is more properly depicted by the computation time, the computation time has improved at each step of pre-processing.This more clearly shown on the plot below:



(a) Accuracy Plot



(b) Computation Time Plot

Fig. 4.10: Observation Plot for different principal component

The computation time gets reduced significantly by pre-processing. Theres also slight improvement in the accuracy.

For discretizing the target variable, I had three methods of discretizer available. In order to use the most optimal method, I experimented with the three discretizer obtained the following table.

Table 4.6: Accuracy for different methods of Discretizer

Discretizer Method	**Machine Learning Model**				
	Logistic Regression (Multinomial + Saga)	**Logistic Regression (Multinomial + Sag)**	**Logistic Regression (OvR + liblinear)**	**SVM (OvR + rbf)**	**SVM (OvR + poly)**
Kmeans	99.05%	99.16%	92.43%	98.65%	94.09%
Uniform	99.46%	99.59%	98.29%	99.46%	99.56%
Quantile	94.97%	95.18%	77.05%	95.41%	79.93%

From the table, I cam conclude that uniform method of discretization produced the most favourable results.

4.4 Need for Discretizing the vMOS

MOS or Mean Opinion Score is the most commonly and globally accepted measure used in the domain of QoE(Quality of Experience) and telecommunication engineering, representing overall quality of system. It also enables us to understand the important factors determining the QoE of the service.

The abbreviation MOS is defined in ITU-T Rec.P.10(International Telecommunication Union - Telecommunication) in following way:

The mean of opinion scores, i.e. of the values on a predefined scale that subject assign to their opinion of the performance of the telephone transmission system used either for conversation or for listening to spoken material.

ITU-T has recommended standardized quality rating scales. One of the most commonly used rating scales is ACR(Absolute Category Ranking). It maps rating between bad and excellent to numbers between 1 and 5. Thus, it provides a standardized and quantifiable way to measure subjective responses. It helps researchers to gather numerical data that can be analysed statistically to analyse pattern. Now, the table below shows the performance of different prediction model when vMOS is undiscretized and discretized:

Table 4.7: Accuracy for Continuous & Discrete Target Variable

Discrete Target Variable		**Continuous Target Variable**	
Logistic Regression	**Support Vector Classification**	**Linear Regression**	**Support Vector Regression**
99.16%	99.45%	87.02%	77.77%

Thus, from the above observations also, it proves that the process of pattern recognition to predict data correctly is more accurate in case of discretized vMOS. Hence, this proves the need to discretize the target data.

4.5 Summary

Github Link To my Project:"https://shorturl.at/crx34"
To summarise, in this section of my report I explained in detail the various steps I took for the processing of data. The numerous experiments I did with various models of prediction, to find the most optimum one. I also highlighted how pre-processing creates a difference in the coputation process of the model. At the end, the following important takeaways are to be considerd, firstly preprocessing of dataset and its analysis is a very important step. Secondly, target variables like vMOS needs to be discretized as per the recommendations

of ITU-T for obtaining the best results.

CHAPTER 5

Neural Network

After completion of my internship project, I also explored the field of neural network. I familiarised myself with the introductory pattern recognition algorithm of the neural networks to perform the classification that I did using machine learning. Therefore, this section will contain a very brief introduction to the neural networks and how I used the predefined functions of the MATLAB to perform the classification on the previous dataset using pattern recognition.

5.1 Introduction

[20] A neural network is a computational model inspired by the structure and functioning of biological neural networks, such as the human brain. It is a powerful machine learning algorithm that has found extensive applications in various domains, including image recognition, natural language processing, and reinforcement learning.

It consists of the node layers replicating the network of neurons in our brain, containing 1 input layer, 1 or more hidden layers and an output layer.

Each node/neuron connects to another and has an associated weight and threshold value. If the output of any individual node is above threshold, that node is activated and sending data to next layer else no data is send. Just like how responses flows through the neurons of our brain.

Neural networks are highly versatile and can be adapted to solve a wide range of problems. They excel at tasks involving pattern recognition, classification, regression, and sequence learning.

5.2 Working of Neural Network

In a neural network, each individual node is considered to have it's own linear regression model composed of input features multiplied with weights and added to bias.Mathematical formula is similar to the formula of linear regression model. The difference is the output of this formula is 1 if it is greater than the threshold value else output is 0 if it is less than the threshold value.

Input layer is determined and weights are assigned, multiplied to the features added to the bias. The resultant is treated as an activation formula. In most use cases, supervised labelled dataset are used to train the algorithm. To enhance the acccuracy using a cost function or loss function such as mean squared error is used. The goal is to minimize cost function. Therefore, model adjust it's weight and bias, uses cost function and reinforcement learning, to reach point of convergence.

Most Neural Network are feed forward, meaning they flow in one direction only from input to output.

There is also another way called back propagation, where training of model happens in backward direction, that is using outputs to determine the weights and biases.

5.3 Pattern Recognition

[21] Pattern recognition is a fundamental capability of neural networks. Neural networks excel at learning and recognizing patterns in data, allowing them to classify and identify complex patterns in various domains.

Neural networks are known for their ability to learn complex patterns and generalize well to new data. Through the training process, they extract relevant features and capture intricate patterns in the input data, making them effective tools for various pattern recognition tasks.

I used the patternet function of the MATLAB to perform the classification on the dataset that I used in the previous section. I have used the pre-processed dataset that I created in previous sections. The dataset contains the first 6 principal components and the

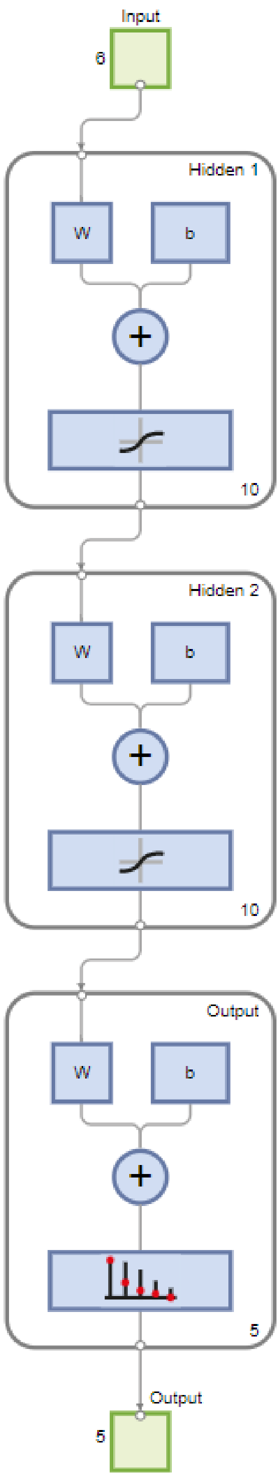


Fig. 5.1: Pattern Recognition Neural Network

uniformly discretized target variable.

The patternet function of the MATLAB uses feed forward neural network for pattern recognition tasks. The underlying process involved in this function can be understood in following subcategories:

- 1. Network Architecture
  Patternet function of MATLAB allows us to define the architecture of the neural network. It allows us to specify the number of hidden layers and the number of neurons in each layer.
  For Example:

  ```
      hiddenLayerSize = [10 10];

  ```

  It creates 2 hidden layers with 10 neurons in each layer.

- 2. Training Algorithm
  It uses resilient backpropagation algorithm as default training algorithm. It uses gradient based optimization that adjust the weights and biases of the neural network during training to minimize the error between predicted output and actual target.

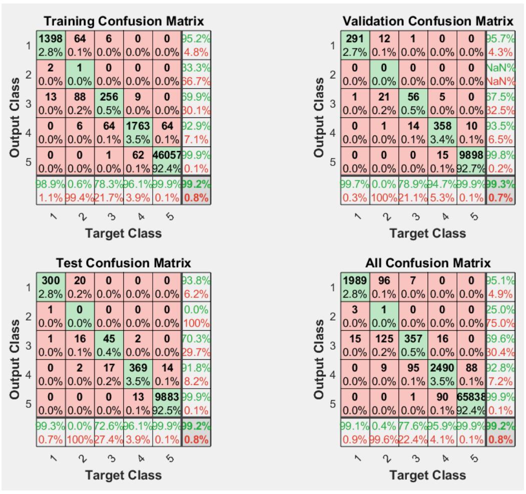
- 3. Parameters
  The patternet function provides various parameters to configure and to customize the training process.

Here's the MATLAB code I used:

```
% Loading dataset:
data = xlsread('FinalDataset_unf.xlsx');
features = data(:, 1:6);
target = data(:, 7)+1;

%%
% Preparing Training and Test dataset
cv = cvpartition(target, 'HoldOut', 0.2);  % 80% for training, 20% for
    testing
trainFeatures = features(training(cv), :)';
trainTarget = target(training(cv))';
testFeatures = features(test(cv), :)';
testTarget = target(test(cv))';

%%
% Creating and Training the neural n/w
hiddenLayerSize = [10 10];  % Adjust the number of hidden layers and
    neurons as needed
net = patternnet(hiddenLayerSize);
net = train(net, trainFeatures, ind2vec(trainTarget));

%%
```

```
predictions = vec2ind(net(testFeatures));

%%
accuracy = sum(predictions == testTarget) / numel(testTarget);
% Calculate accuracy
fprintf('Accuracy: %.2f%%\n', accuracy * 100);
```

Here's the output I got:
The pattern recognition neural network got an accurcy of 99.20



Training Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	
1	1398 2.8%	64 0.1%	6 0.0%	0 0.0%	0 0.0%	95.2% 4.8%
2	2 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	33.3% 66.7%
3	13 0.0%	88 0.2%	256 0.5%	9 0.0%	0 0.0%	69.9% 30.1%
4	0 0.0%	6 0.0%	64 0.1%	1763 3.5%	64 0.1%	92.9% 7.1%
5	0 0.0%	0 0.0%	1 0.0%	62 0.1%	46057 92.4%	99.9% 0.1%
	98.9% 1.1%	0.6% 99.4%	78.3% 21.7%	96.1% 3.9%	99.9% 0.1%	99.2% 0.8%

Validation Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	
1	291 2.7%	12 0.1%	1 0.0%	0 0.0%	0 0.0%	95.7% 4.3%
2	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN% NaN%
3	1 0.0%	21 0.2%	56 0.5%	5 0.0%	0 0.0%	67.5% 32.5%
4	0 0.0%	1 0.0%	14 0.1%	358 3.4%	10 0.1%	93.5% 6.5%
5	0 0.0%	0 0.0%	0 0.0%	15 0.1%	9898 92.7%	99.8% 0.2%
	99.7% 0.3%	0.0% 100%	78.9% 21.1%	94.7% 5.3%	99.9% 0.1%	99.3% 0.7%

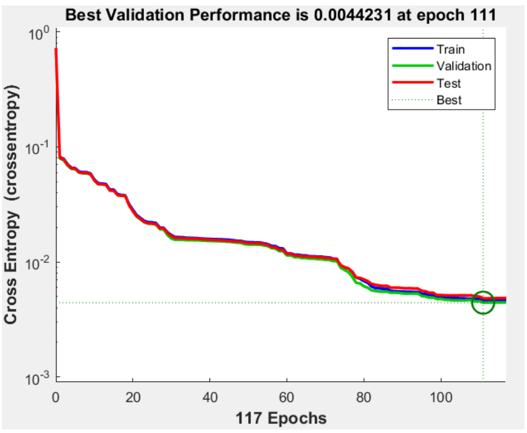
Test Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	
1	300 2.8%	20 0.2%	0 0.0%	0 0.0%	0 0.0%	93.8% 6.2%
2	1 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0.0% 100%
3	1 0.0%	16 0.1%	45 0.4%	2 0.0%	0 0.0%	70.3% 29.7%
4	0 0.0%	2 0.0%	17 0.2%	369 3.5%	14 0.1%	91.8% 8.2%
5	0 0.0%	0 0.0%	0 0.0%	13 0.1%	9883 92.5%	99.9% 0.1%
	99.3% 0.7%	0.0% 100%	72.6% 27.4%	96.1% 3.9%	99.9% 0.1%	99.2% 0.8%

All Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	
1	1989 2.8%	96 0.1%	7 0.0%	0 0.0%	0 0.0%	95.1% 4.9%
2	3 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	25.0% 75.0%
3	15 0.0%	125 0.2%	357 0.5%	16 0.0%	0 0.0%	69.6% 30.4%
4	0 0.0%	9 0.0%	95 0.1%	2490 3.5%	88 0.1%	92.8% 7.2%
5	0 0.0%	0 0.0%	1 0.0%	90 0.1%	65838 92.4%	99.9% 0.1%
	99.1% 0.9%	0.4% 99.6%	77.6% 22.4%	95.9% 4.1%	99.9% 0.1%	99.2% 0.8%

(a) Confusion Matrix



(b) Model Performance

Fig. 5.2: MATLAB Outputs after training of Neural Network

The confusion matrix in the above figure highlights the deviation that the predicted output of the neural network has from actual target. While the model performance figure shows the point which the performance is the best and the cross entropy loss at this point.

5.4 Summary

In this section, I have presented my understandings about the neural networks in brief. I used the pre-processed dataset of the previous section to highlight, how the predefined patternet function is coded to train and predict. The various parameters that we can change to implement such complex networks easily.

CHAPTER 6

Conclusion

6.1 Conclusion

In conclusion, during the course of this internship, I extensively studied about the different aspects of data analysis. I came to know about the various types of data that a dataset may contain. I understood that ML models cannot work with all types of data. Hence the dataset need to be understood and analyzed for any abnormalitites.

I analyzed various methods of preprocessing in order to enhance the quality and reliability of data for subsequent analysis. These pre-processing techniques included data cleaning, normalization, feature scaling, handling missing values, and outlier detection and treatment. I also explored the conversion of non numeric data to numeric data to make the dataset fit for training. I also studied about the supervised machine learning model in depth as I used it for prediction.

Through the practical implementation of these methods in a project, I successfully demonstrated their effectiveness in improving the accuracy and performance of the predictive models. The project served as a valuable opportunity to apply the theoretical knowledge gained during the internship and provided insights into the importance of proper data preprocessing for obtaining reliable and meaningful results. I can conclude the following results:

- A deep analysis nad understanding of dataset is very important.
- Treating the non numeric data, outliers and scaling the dataset is another important step.
- Discretizing method is also another factor in preprocessing.
- Choosing the appropriate ML model with appropriate parameters is another important step.

I have also highlighted the importance of discretizing the MOS as per the recommended ITU-T standards for MOS, and how it improves accuracy of prediction.
In addition to above, I also conducted an introductory research on neural networks, with a specific focus on pattern recognition algorithms. I gained insights into the fundamental principles of neural networks and their application in pattern recognition tasks. To showcase the practical implementation of these concepts, I utilized the patternet function in MATLAB, which allowed me to build and train neural network models for pattern recognition. Overall, this internship experience has deepened my understanding of pre-processing techniques and their significance in the field of data analysis and machine learning.

REFERENCES

[1] R. Khan, “Importance of datasets in machine learning and ai research.” Available at: `https://www.datatobiz.com/blog/datasets-in-machine-learning/`(2022/13/05).

[2] R. Sharma, “4 types of data: Nominal, ordinal, discrete, continuous.” Available at: `https://www.upgrad.com/blog/types-of-data/`(2022/04/10).

[3] A. M. Lamine Amour, Sami Souihi, “Poqemon-qoe-dataset.” Available at: `https://github.com/Lamyne/Poqemon-QoE-Dataset/tree/master`(2018/16/01).

[4] S. Ray, “Simple methods to deal with categorical variables in predictive modeling.” Available at: `https://www.analyticsvidhya.com/blog/2015/11/easy-methods-deal-categorical-variables-predictive-modeling/`(2022/16/06).

[5] J. Brownlee, “How to use discretization transforms for machine learning.” Available at: `https://machinelearningmastery.com/discretization-transforms-for-machine-learning/`(2022/20/05).

[6] N. Duggal, “Difference between covariance and correlation: A definitive guide.” Available at: `https://www.simplilearn.com/covariance-vs-correlation`(2023/31/05).

[7] J. Brownlee, “How to remove outliers for machine learning.” Available at: `https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/`(2018/25/04).

[8] A. P. Gulati, “Dealing with outliers using the z-score method.” Available at: `https://www.analyticsvidhya.com/blog/2022/08/dealing-with-outliers-using-the-z-score-method/`(2022/13/08).

[9] D. Rao, “Removing outliers by iqr method.” Available at: `https://www.kaggle.com/code/durgeshrao9993/removing-outliers-by-iqr-method`(2021/01/06).

[10] A. Bhandari, “Feature engineering: Scaling, normalization, and standardization.” Available at: `https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/`(2020/03/04).

[11] A. Biswal, “Principal component analysis in machine learning: Complete guide.” Available at: `https://www.simplilearn.com/tutorials/machine-learning-tutorial/principal-component-analysis`(2023/24/03).

[12] javaTpoint, “Machine learning tutorial.” Available at: `https://www.javatpoint.com/machine-learning`(NA).

[13] K. Mali, “Everything you need to know about linear regression!.” Available at: https://www.analyticsvidhya.com/blog/2021/10/everything-you-need-to-know-about-linear-regression/(2021/04/10).

[14] T. Sharp, “An introduction to support vector regression (svr).” Available at: https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2(2023/24/03).

[15] A. Saini, “Conceptual understanding of logistic regression for data science beginners.” Available at: https://www.analyticsvidhya.com/blog/2021/08/conceptual-understanding-of-logistic-regression-for-data-science-beginners/(2021/03/08).

[16] S. Raschka, “What is softmax regression and how is it related to logistic regression?.” Available at: https://www.kdnuggets.com/2016/07/softmax-regression-related-logistic-regression.html(2016/07/01).

[17] A. Saini, “Support vector.” Available at: https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/(2021/12/10).

[18] baeldung, “Multiclass classification using support vector machines.” Available at: https://www.baeldung.com/cs/svm-multiclass-classification(2022/11/11).

[19] D. W. a. H. X. Qingyong Wang, Hong-Ning Dai, “Data analysis on video streaming qoe over mobile networks,” *EURASIP Journal onWireless Communications and Networking*, 2018.

[20] IBM, “What are neural networks?.” Available at: https://www.ibm.com/topics/neural-networks(NA).

[21] S. Ansari, “Pattern recognition | introduction.” Available at: https://www.geeksforgeeks.org/pattern-recognition-introduction/(NA).