ADVANCED ACADEMIC CENTER

A CENTER FOR INTER-DISCIPLINARY RESEARCH
2020-21

TITLE

“DROUGHT PREDICTION”

SUPERVISED BY

MEDISHETTY SUMANA

GOKARAJU RANGARAJU
INSTITUTE OF ENGINEERING AND TECHNOLOGY
AUTONOMOUS

Advanced Academic Center

(A Center For Inter-Disciplinary Research)

This is to certify that the project titled

“DROUGHT PREDICTION”

is a bonafide work carried out by the following students in partial fulfilment of the requirements for Advanced Academic Center intern, submitted to the chair, AAC during the academic year 2020-21.

NAME	ROLL NO.	BRANCH
HARSHITHA CHILUPURI	21241A6619	CSM
MAHITHA MUDIREDDY	21241A0437	ECE
PRATIK NADIAPALLY	21241A66H3	CSM

NAME	ROLL N0	BRANCH
SAI MEGHNADH GANTA	21241A05M5	CSE

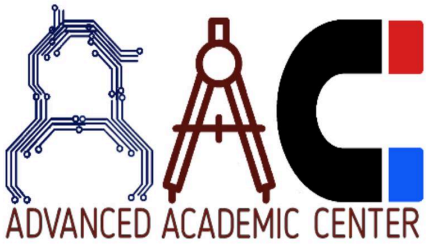
This work was not submitted or published earlier for any study

Dr/Ms./Mr.

Project Supervisor

Dr.B.R.K.Reddy
Program Coordinator

Dr.Ramamurthy Suri
Associate Dean,AAC



ACKNOWLEDGEMENTS

We express our deep sense of gratitude to our respected Director, Gokaraju Rangaraju Institute of Engineering and Technology, for the valuable guidance and for permitting us to carry out this project.

With immense pleasure, we extend our appreciation to our respected Principal, for permitting us to carry out this project.

We are thankful to the Associate Dean, Advanced Academic Centre, for providing us an appropriate environment required for the project completion.

We are grateful to our project supervisor who spared valuable time to influence us with their novel insights.

We are indebted to all the above mentioned people without whom we would not have concluded the project.

CONTENTS

1.ABSTRACT

Climate change has increased frequency, severity and areal extent of droughts across the world in the last few decades magnifying their adverse impacts. Prediction of droughts is immensely helpful in early warning and preparing the most vulnerable communities to their adverse impacts. For the first time, this study investigated the potential of developing drought prediction models over a region using three state-of-the-art Machine Learning (ML) techniques; Support Vector Machine (SVM), Artificial Neural Network (ANN) and k-Nearest Neighbour (KNN).

Three categories of droughts; moderate, severe, and extreme considering two major cropping seasons called Rabi and Kharif were estimated using Standardized Precipitation Evaporation Index (SPEI) and then predicted using the predictor data obtained from the National Centres for Environmental Prediction/National Centre for Atmospheric Research (NCEP/NCAR) reanalysis database. Also, for the first time in drought modelling, a novel feature selection approach called Recursive Feature Elimination (RFE) was used for identifying optimum sets of predictors. In validation, SVM-based models were able to better capture the temporal and spatial characteristics of droughts over a region compared to those by ANN and KNN-based models. KNN which was used in developing drought models for the first time displayed limited performance in comparison to that by SVM and ANN-based drought models, in validation. It was found that in the Rabi season SPEI is positively correlated with relative humidity over the Mediterranean Sea and the region north of the Caspian Sea. In the Kharif season, SPEI is positively correlated with the humid region over the south-eastern part of the Bay of Bengal and the regions north of the Mediterranean and Caspian Seas. In developing a drought prediction model for a given region, relative humidity, temperature and wind speed should be considered with a domain which encompasses the Mediterranean Sea, the region north of the Caspian Sea, the Indian Ocean and the Arabian Sea.

Drought prediction is of critical importance to early warning for drought managements. This work provides a synthesis of drought prediction based on statistical, dynamical, and hybrid methods. Statistical drought prediction is achieved by modeling the relationship between drought indices of interest and a suite of potential predictors, including large-scale climate indices, local climate variables, and land initial conditions. Dynamical meteorological drought prediction relies on seasonal climate forecast from general circulation models (GCMs), which can be employed to drive hydrological models for agricultural and hydrological drought prediction with the predictability determined by both climates forcing and initial conditions.

SVM model has been applied for classification of real time data obtained from Meteorological department. Here we considered parameters like maximum rainfall, minimum rainfall and precipitation. The prediction is base on the dataset collected for a period of ten years over a region.

2.INTRODUCTION

2.1 What is Drought?

A drought is defined as drier than normal conditions. This means that a drought is "a moisture deficit relative to the average water availability at a given location and season". A drought can last for days, months or years. Drought can have a serious impact on health, agriculture, economies, energy and the environment.

2.2 What causes a Drought?

When rainfall is less than normal for a period of weeks to years, stream flows decline, water levels in lakes and reservoirs fall, and the depth to water in wells increases. If dry weather persists and water-supply problems develop, the dry period can become a drought. by not receiving rain or snow over a period of time. We learned in the discussions about the water cycle and weather that changes in the wind patterns that move clouds and moisture through the atmosphere can cause a place to not receive its normal amount of rain or snow over a long period of time.

Drought can also cause long-term public health problems, including: Shortages of drinking water and poor quality drinking water. Impacts on air quality, sanitation and hygiene, and food and nutrition. More disease, such as West Nile Virus carried by mosquitoes breeding in stagnant water. Atmospheric conditions such as climate change, ocean temperatures, changes in the jet stream, and changes in the local landscape are all factors that contribute to drought.

2.3 Machine Learning and Drought Prediction

Machine learning is a dynamic field with wide-ranging applications, including drought modeling and forecasting. Drought is a complex, devastating natural disaster for which it is challenging to develop effective prediction models. Machine learning (ML) have achieved significant breakthroughs for drought modelling in recent years. Therefore, our review focuses on basic information about machine learning methods (MLMs) and their potential applications in developing efficient and effective drought forecasting models.

3.MACHINE LEARNING

3.1 Introduction to Machine Learning

Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and by the use of data. It is seen as apart of artificial intelligence. Machine learning algorithms build a model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

Machine learning involves computers discovering how they can perform tasks without being explicitly programmed to do so. It involves computers learning from data provided so that they carry out certain tasks. For simple tasks assigned to computers, it is possible to program algorithms telling the machine how to execute all steps required to solve the problem at hand; on the computer's part, no learning is needed. For more advanced tasks, it can be challenging for a human to manually create the needed algorithms. In practice, it can turn out to be more effective to help the machine develop its own algorithm, rather than having human programmers specify every needed step.

A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers, but not all machine learning is statistical learning.

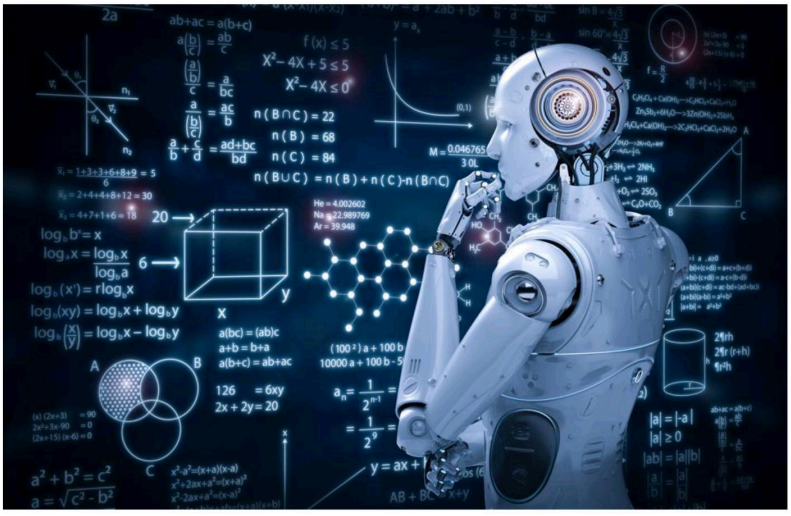


Fig: *Machine Learning*

3.2 Data in ML

One of the toughest issues to resolve in machine learning has nothing to do with neural networks; it's the matter of obtaining the correct information within the right format. If you are doing not have the correct information, then your efforts to create AI resolution which should come back to the information assortment stage. . Data can be any unprocessed fact, value, text, sound or picture that is not be interpreted and analyzed. Data is the most important part of all Data Analytic, Machine Learning, Artificial Intelligence. Without data, we can't train any model and all modern research and automation will go vain.

There are three main types of Data used in ML. They are:

- Training Data
- Validation Data
- Testing Data

Training Data: The part of data we use to train our model. This is the data which your model actually sees(both input and output) and learn from.

Validation Data: The part of data which is used to do a frequent evaluation of model, fit on training dataset along with improving involved hyperparameters (initially set parameters before the model begins learning). This data plays it's part when the model is actually training.

Testing Data: Once our model is completely trained, testing data provides the unbiased evaluation. When we feed in the inputs of Testing data, our model will predict some values(without seeing actual output). After prediction, we evaluate our model by comparing it with actual output present in the testing data. This is how we evaluate and see how much our model has learned from the experiences feed in retreving data, set at the time of training.

If accurate predictions are not obtained, then the data used must be retrained. Here we investigate the hyper parameters accustomed to tune the network. This process enhances the quality of knowledge the pre-processing technique , which eventually increases the accuracy of the project.

3.3 ML Algorithms

An “algorithm” in machine learning is a procedure that is run on data to create a machine learning “model.” Machine learning algorithms perform“ pattern recognition.” Algorithms “learn” from data, or are “fit” on a dataset. There are many machine learning algorithms.

Types of ML Algorithms(Classifiers):

There are various types of classifiers. Some of them are:

- ❑ Linear Classifiers : logistic Regression
- ❑ Tree Based Classifiers : Decision Tree Classifier
- ❑ Support Vector Machine(SVM)
- ❑ Artificial Neural Networks
- ❑ Bayesian Regression
- ❑ Gaussian Naïve Bayes Regression
- ❑ Stochastic Gradient Descent(SGD) Classifier
- ❑ Classifier, ExtraTrees Classifiers

3.4 More about Random Forest:

Every decision tree has high variance, but when we combine all of them together inparallel then the resultant variance is low as each decision tree gets perfectly trained on that particular sample data and hence the output doesn’t depend on one decision tree but multiple decision trees. In the case of a classification problem, the final output is taken by using the majority voting classifier. In the case of are regression problem, the final output is the mean of all the outputs. This part is Aggregation.

Step 1 : Import the required libraries.

Step 2 : Import and print the dataset

Step 3 : Select all rows and column 1 from dataset to x and all rows and column2as y

Step 4 : Fit Random forest regressor to the dataset

Step 5 : Predicting a new result

Step 6 : Visualising the result

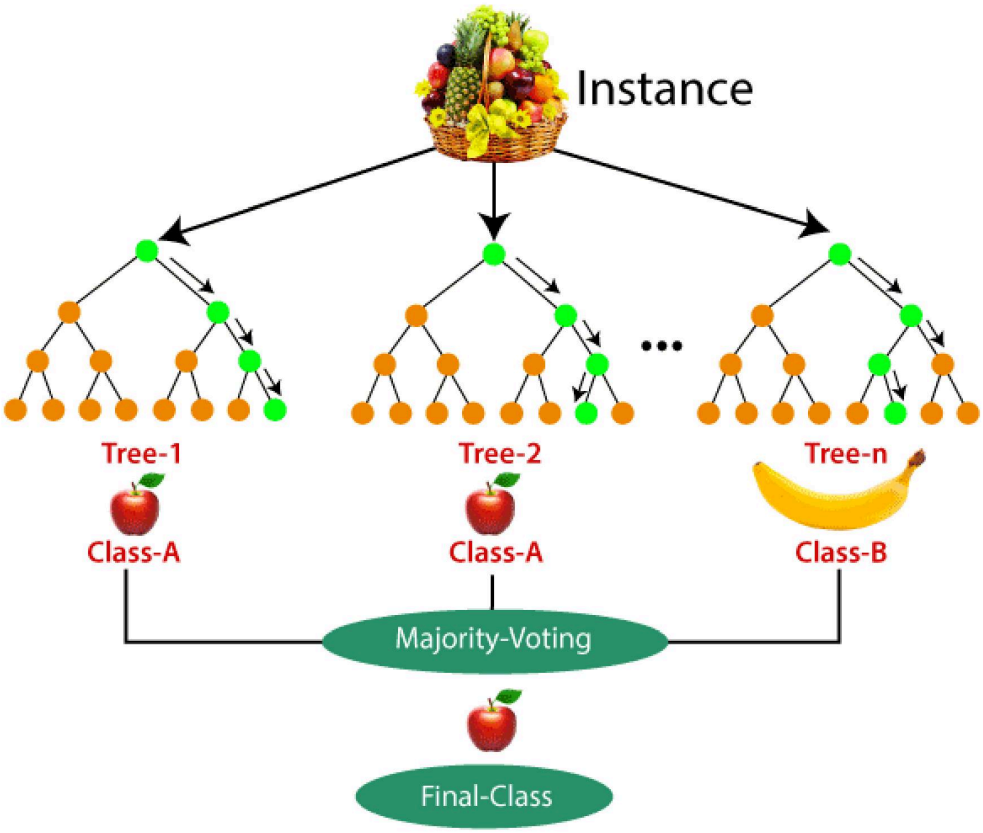


Fig: *Random Forest Algorithm*

3.5 History of ML

Tom M. Mitchell provided a widely quoted, more formal definition of the algorithms studied in the machine learning field: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." This definition of the tasks in which machine learning is concerned offers a fundamentally operational definition rather than defining the field incognitive terms.

Modern day machine learning has two objectives, one is to classify data based on models which have been developed, the other purpose is to make predictions for future outcomes based on these models. A hypothetical algorithm specific to classifying data may use computer vision of moles coupled with supervised learning in order to train it to classify the cancerous moles. Where as, a machine learning algorithm for stock trading may inform the trader of future potential predictions.

3.6 Future of ML

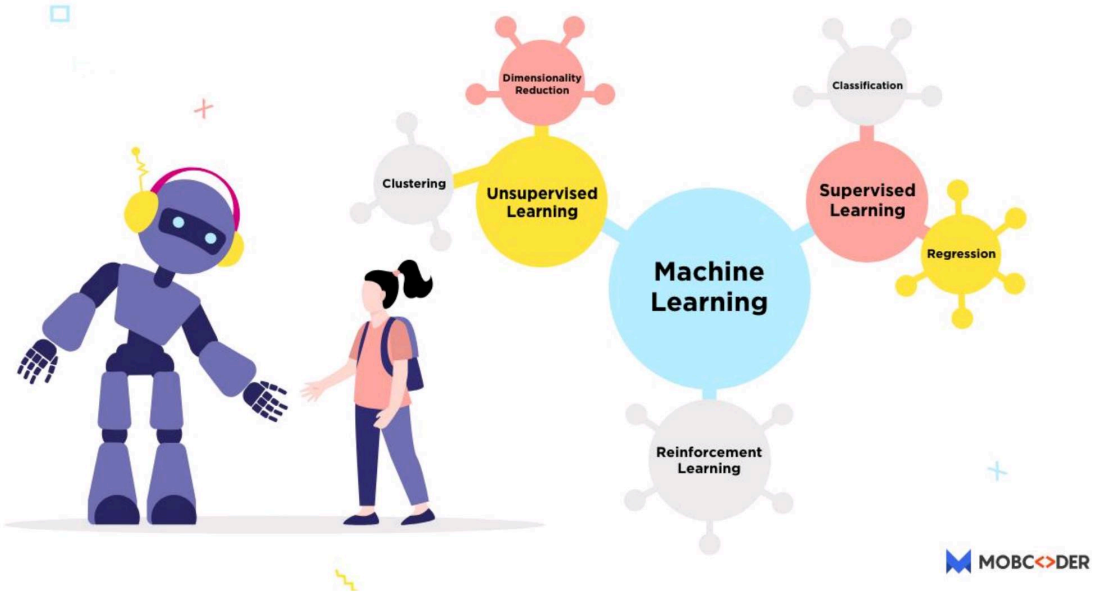
Machine Learning has been one of the hottest topics of discussion among the C-suite. With its incredible potential to compute and analyze huge amounts of data, advancedML techniques are being used in businesses to perform complex tasks quicker and more efficiently. The machine learning market is expected to grow from USD 1.03 Billion in2016 to USD 8.81 Billion by 2022, at a Compound Annual Growth Rate (CAGR) of 44.1% during the forecast period.

Machine learning-driven solutions are being leveraged by organizations to improve customer experience, ROI, and to gain a competitive edge in business. Big players in the field like Google, IBM, Microsoft, Apple, and Salesforce are already leveraging ML benefits using Machine Learning.

3.7 Applications of ML

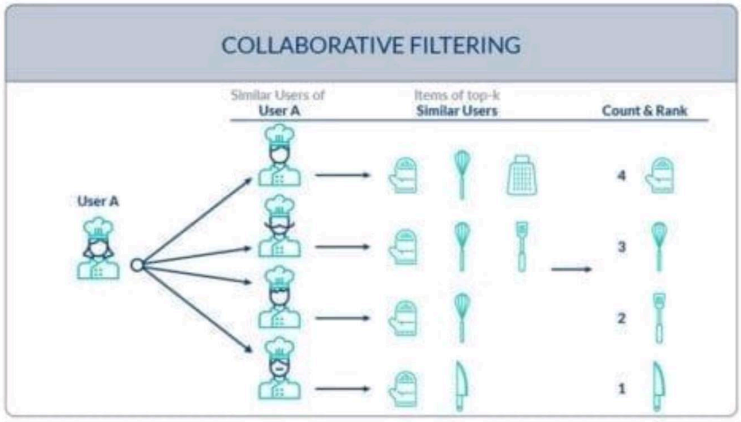
a. Machine Learning in Education

Advances in AI are enabling teachers to realize a far better understanding of how their students are progressing with learning. AI will make big and positive changes in education helping students to enjoy the training process and have a far better understanding with their teachers. Students will not feel apprehensive towards their teachers and be frightened of being judged.



b. Machine Learning in Search Engine

Search engines rely on machine learning to improve their services is no secret today. Implementing these Google has introduced some amazing services. Such as voice recognition, image search and many more. Google services like its image search and translation tools use sophisticated machine learning which permit computers to ascertain , listen and speak in much an equivalent way as human do.



d. Machine Learning in Health Care

Machine learning, simply put, may be a sort of AI when computers are programmedto find out information without human intervention. The foremost common healthcare use cases for machine learning are automating medical billing, clinical decision support and therefore the development of clinical care guidelines. More importantly, scientists and researchers are using machine learning (ML) to churn out variety of smart solutions which will ultimately help in diagnosing and treating an illness.

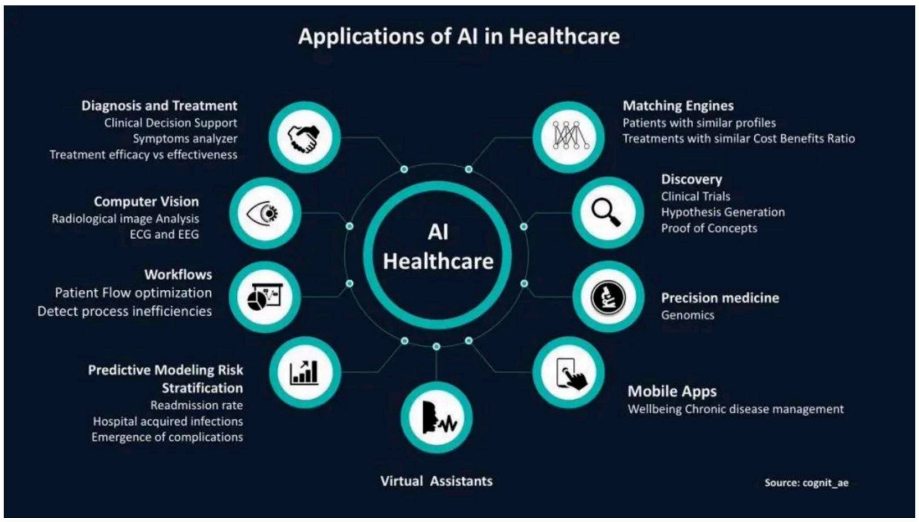
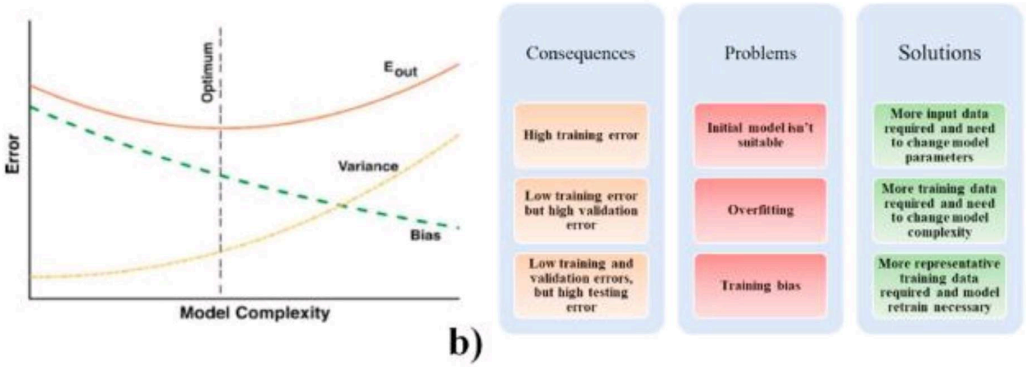


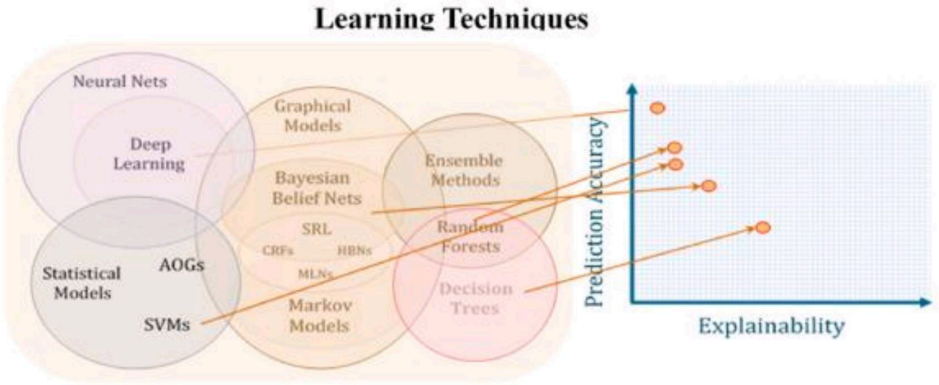
Fig:*Health care in ML*

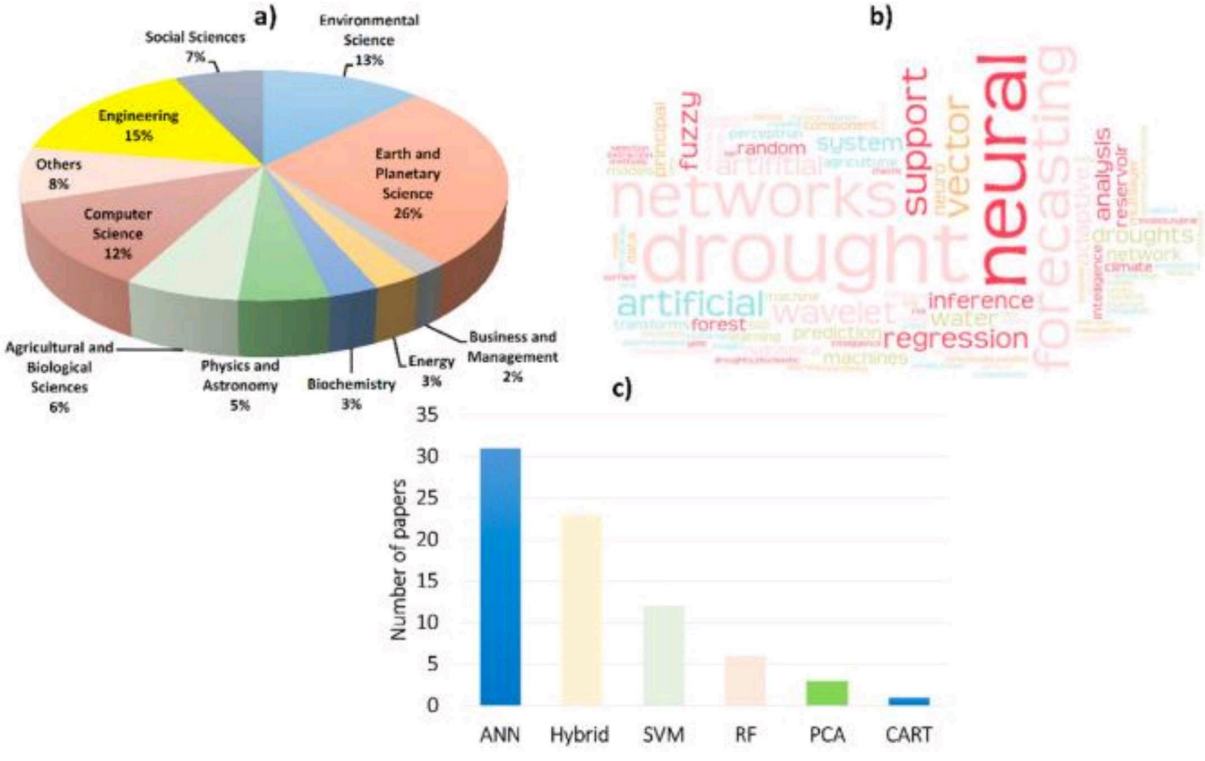
Graphical Analysis

a)



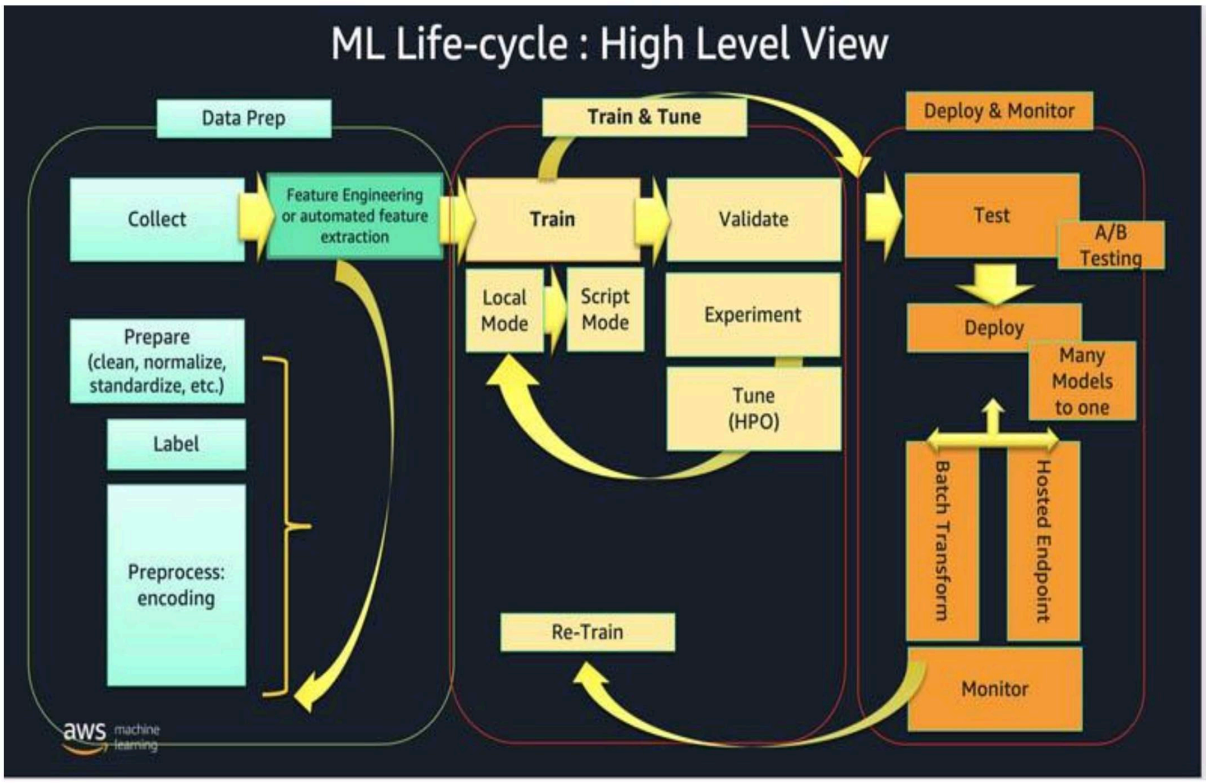
b)





PROJECT WORKFLOW

Machine learning workflows define the steps initiated during a particular machine learning implementation. Machine learning workflows vary by project, but four fundamental phases are typically included.



4.1 DATA GATHERING

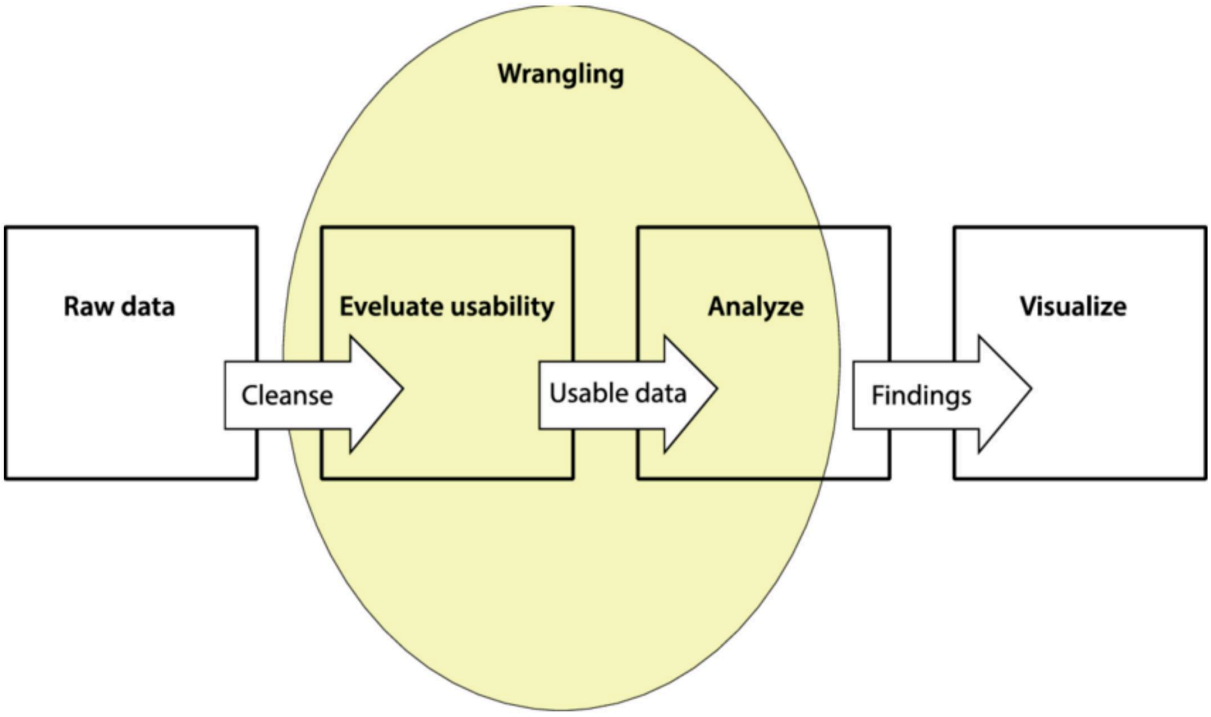
- Gathering data is one of the most important stages of machine learning workflows. During data collection, you are defining the potential usefulness and accuracy of your project with the quality of the data you collect.
- To collect data, you need to identify your sources and aggregate data from those sources into a single dataset. This could mean streaming data from Internet of Things sensors, downloading open source data sets, or constructing a data lake from assorted files, logs, or media.
- Collecting data for training the ML model is the basic step in the machine learning pipeline. The predictions made by ML systems can only be as good as the data on which they have been trained. Following are some of the problems that can arise in data collection:
- **Inaccurate data:** The collected data could be unrelated to the problem statement.
- **Missing data:** Sub-data could be missing. That could take the form of empty values in columns or missing images for some class of prediction.
- **Data imbalance:** Some classes or categories in the data may have a disproportionately high or low number of corresponding samples. As a result, they risk being under-represented in the model.

- **Data bias:** Depending on how the data, subjects and labels themselves are chosen, the model could propagate inherent biases on gender, age or region, for example. Data bias is difficult to detect and remove.
- Several techniques can be applied to address those problems:
- Pre-cleaned, freely available datasets. If the problem statement (for example, image classification, object recognition) aligns with a clean, pre-existing, properly formulated dataset, then take advantage of existing, open-source expertise.i.e:Kaggle datasets,aws open source etc.

4.2 DATA WRANGLING

Data Wrangling is the process of gathering, collecting, and transforming Raw data into another format for better understanding, decision-making, accessing, and analysis in less time. Data Wrangling is also known as Data Munging.

1. Data wrangling deals with the below functionalities:
2. **Data exploration:** In this process, the data is studied, analyzed and understood by visualizing representations of data.
3. **Dealing with missing values:** Most of the datasets having a vast amount of data contain missing values of *NaN, they are needed to be taken* care of by replacing them with mean, mode, the most frequent value of the column or simply by dropping the row having a *NaN* value.
4. **Reshaping data:** In this process, data is manipulated according to the requirements, where new data can be added or pre-existing data can be modified.
5. **Filtering data:** Some times datasets are comprised of unwanted rows or columns which are required to be removed or filtered
6. **Other:** After dealing with the raw dataset with the above functionalities we get an efficient dataset as per our requirements and then it can be used for a required purpose like data analyzing, machine learning, data visualization, model training etc.



4.3 DATA PREPARATION

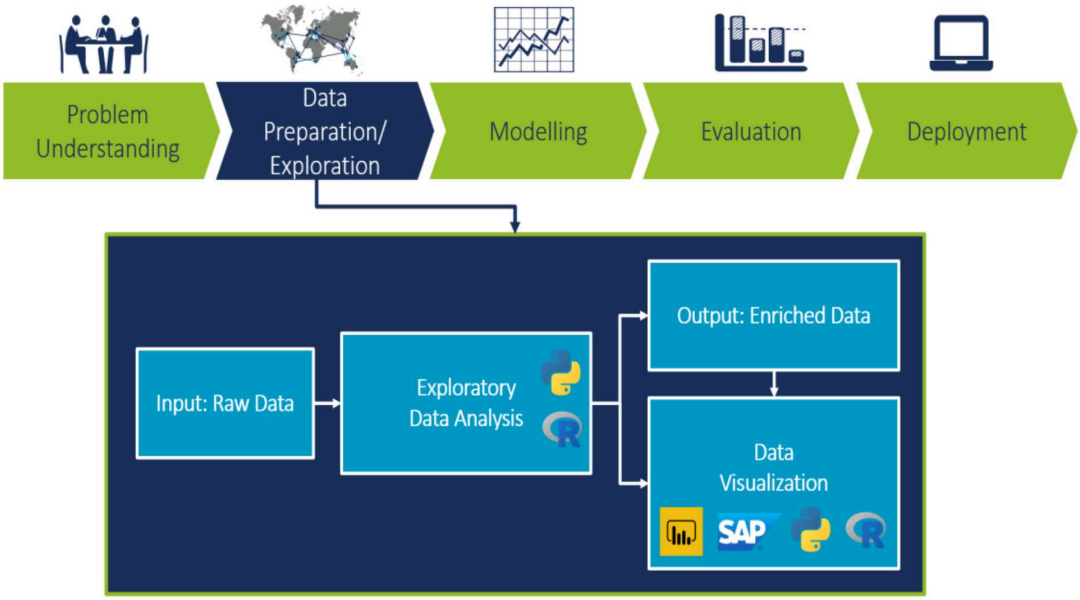
- Data preparation is one of the critical steps in the machine learning project building process, and it must be done in particular series of steps which includes different tasks. There are some essential steps of the data preparation process in machine learning are:
- **Profiling and Data Exploration:** After analyzing and collecting data from various data sources, it's time to explore data such as trends, outliers, exceptions, incorrect, inconsistent, missing, or skewed information, etc. Although source data will provide all model findings, it does not contain unseen biases. Data exploration helps to determine problems such as co-linearity, which means a situation when the Standardization of data sets and other data transformations are necessary.
- **Data Cleaning and Validation:** Data cleaning and validation techniques help determine and solve inconsistencies, outliers, anomalies, incomplete data, etc. Clean data helps to find valuable patterns and information in data and ignores irrelevant data in the datasets. It is very much essential to build high-quality models, and missing or incomplete data is one of the best examples of poor data. Since missing data always reduces prediction accuracy and performance of the model, data must be cleaned and validated through various imputation tools to fill incomplete fields with statistically relevant substitutes.
- **Data Formatting:** After cleaning and validating data, the following approach is to ensure that the data is correctly formatted or not. If data is formatted incorrectly, it will help build a high-quality model.
  Since data comes from various sources or is sometimes updated manually, there are high chances of discrepancies in the data format Similarly, there may be anomalies in their spelling, abbreviation, etc. This type of data formation leads to incorrect predictions. To reduce these errors, formatting of data in a consistent manner by using some input formatting protocols is done.
- **Improve data quality:** Quality is one of the essential parameters in building high-quality models. Quality data helps to reduce errors, missing data, extreme values, and outliers in the datasets., Intelligent ML algorithms must have the ability to match these columns and join the dataset for a singular view of the customer.
- **Split Data**: After feature engineering and selection, the last step is to split your data into two different sets (training and evaluation sets). Further, always select non-overlapping subsets of your data for the training and evaluation sets to ensure proper testing.

4.4 EXPLORATORY DATA ANALYSIS

- Exploratory data analysis is the process of analyzing and interpreting datasets while summarizing their particular characteristics with the help of data visualization methods.
- EDA assist in determining the best possible ways to manipulate data resources to obtain required interferences, making data easier to study and discover hidden trends, test a hypothesis and check assumptions. Moreover, the method scrutinizes data in order to deliver:

1. Optimal interpretation into a dataset,

2. Unearth promising structures,
3. Identify outliers and anomalies,
4. Determine optimal factor settings,
5. Detect significant data variables



4.5 Training of model

- The process of training an ML model involves providing an ML algorithm (that is, the *learning algorithm*) with training data to learn from. The term *ML model* refers to the model artefact that is created by the training process.

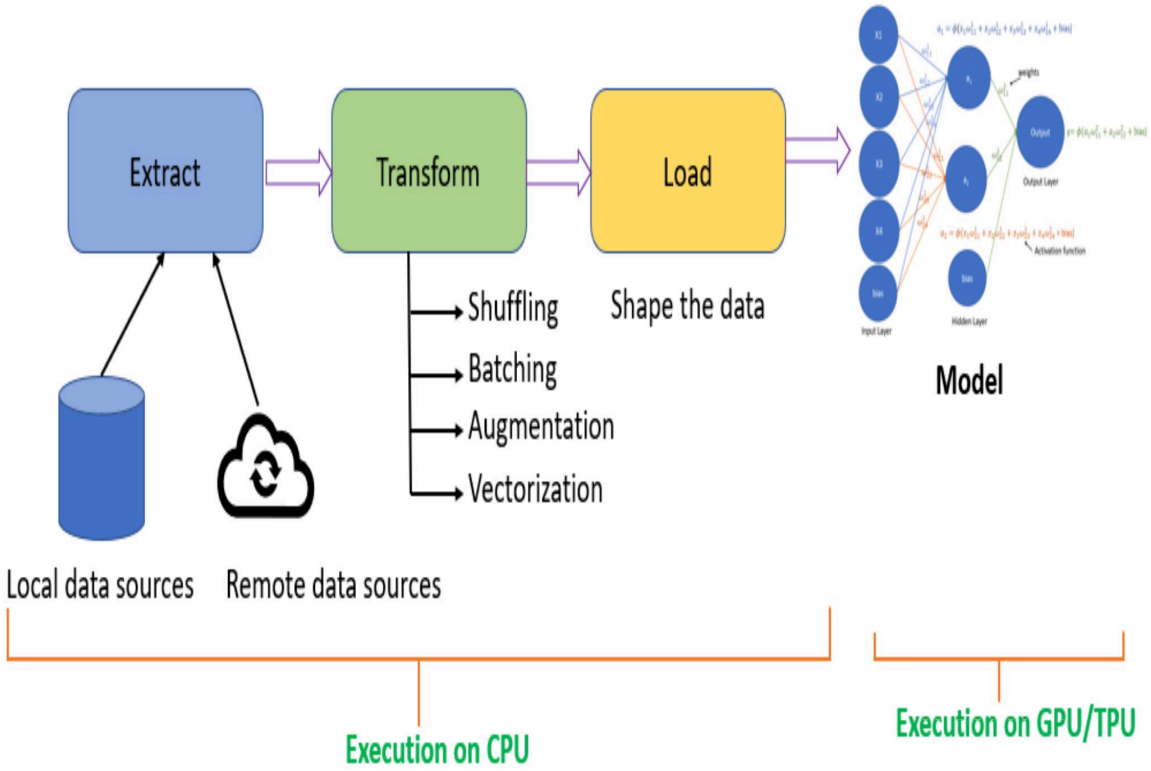
- The training data must contain the correct answer, which is known as a *target* or *target attribute*. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict), and it outputs an ML model that captures these patterns.

- Typically, machine learning algorithms accept parameters that can be used to control certain properties of the training process and of the resulting ML model. These are called *training parameters*.

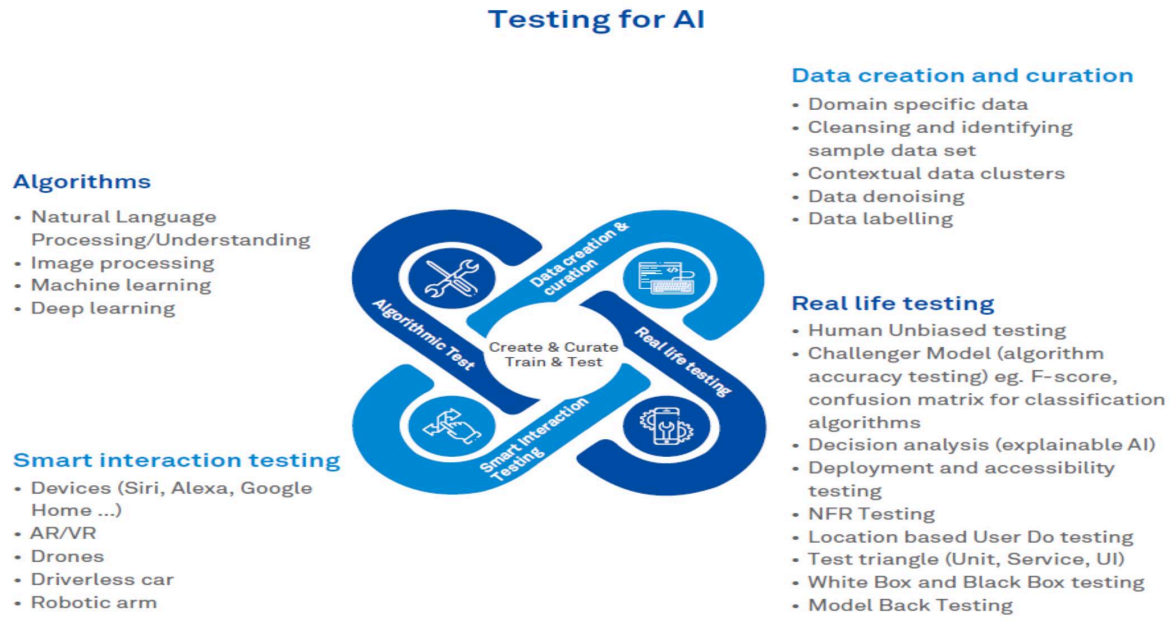
  Usually specified training parameters:

1. Maximum model size
2. Maximum number of passes over training data

3. Shuffle type
4. Regularization type
5. Regularization amount



4.6 Testing of model



The top four elements that we have considered for testing of ML systems and applications are illustrated in Figure 2.

- Once our machine learning model has been trained on a given dataset, then we test the model. In this step, we check for the accuracy of our model by providing a test dataset to it. Testing the model determines the percentage accuracy of the model as per the requirement of project or problem.Testing and analysis provide a reinforcing loop to stabilize the training of the model.

4.8 Deployment

- Deployment of an ML-model simply means the integration of the model into an existing production environment which can take in an input and return an output that can be used in making practical business and research decisions and inferences.
- When building an ML-model for deployment purposes, we must always have your end-users in mind.

import os

```
import subprocess
import pandas as pd
import numpy as
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NeighbourhoodCleaningRule
from imblearn.under_sampling import NearMiss
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as
 LDA
from sklearn.decomposition import PCA, KernelPCA
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, c
lassification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import cohen_kappa_score
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
import pickle
from google.colab import drive
drive.mount('/content/gdrive')
drive.mount("/content/gdrive", force_remount=True)
!unzip gdrive/My\Drive/train
df=pd.read_csv('train_timeseries.csv')
df.head()
df.shape
df.info()
df.isnull().sum()
```

```
df=df.dropna()
df.isnull().sum()
df.dtypes
df['year']=pd.DatetimeIndex(df['date']).year
df['month']=pd.DatetimeIndex(df['date']).month
df['day']=pd.DatetimeIndex(df['date']).day
df['score']=df['score'].round().astype(int)
df.dtypes
df['fips'].nunique()
df['score'].round().value_counts()
df.describe()
column_list=list(df.columns)
plt.figure(figsize=(10,40))
for x in range(1,19):
    plt.subplot(19,1,x)
    sns.boxplot(x =df.columns[x-1], data=df)
    x_name = df.columns[x-1]
    plt.title(f'Distribution of {x_name}')
plt.tight_layout()
print('Total rows = ',len(df.index))
for i in df.select_dtypes(exclude = ['object']).columns:
    print ('Number of values beyong standard outlier limit in ', i)
    print(len(df[(df[i] >df[i].mean() + 3*df[i].std()) | (df[i] < df[i]
.mean() - 3*df[i].std())]))

df = [(df['PRECTOT'] >=df['PRECTOT'].mean() + 3*df['PRECTOT'].std()) &
        (df['PRECTOT'] >= df['PRECTOT'].mean() - 3*df['PRECTOT'].std())
]

df =[(df['PS'] <= df['PS'].mean() + 3*df['PS'].std()) &
        (df['PS'] >= df['PS'].mean() - 3*df['PS'].std())]

df = df[(df['QV2M'] <= df['QV2M'].mean() + 3*df['QV2M'].std()) &
        (df['QV2M'] >= df['QV2M'].mean() - 3*df['QV2M'].std())]

df = df[(df['T2M'] <= df['T2M'].mean() + 3*df['T2M'].std()) &
        (df['T2M'] >= df['T2M'].mean() - 3*df['T2M'].std())]

df = df[(df['T2MDEW'] <= df['T2MDEW'].mean() + 3*df['T2MDEW'].std()) &
        (df['T2MDEW'] >= df['T2MDEW'].mean() - 3*df['T2MDEW'].std())]
df = df[(df['T2MWET'] <= df['T2MWET'].mean() + 3*df['T2MWET'].std()) &
        (df['T2MWET'] >= df['T2MWET'].mean() - 3*df['T2MWET'].std())]

df = df[(df['T2M_MAX'] <= df['T2M_MAX'].mean() + 3*df['T2M_MAX'].std())
 &
        (df['T2M_MAX'] >= df['T2M_MAX'].mean() - 3*df['T2M_MAX'].std())
]
```

```
df = df[(df['T2M_MIN'] <= df['T2M_MIN'].mean() + 3*df['T2M_MIN'].std())
 &
        ( df['T2M_MIN'] >= df['T2M_MIN'].mean() - 3*df['T2M_MIN'].std()
)]

df = df[(['T2M_RANGE'] <=df['T2M_RANGE'].mean() + 3*df['T2M_RANGE'].std
()) &
        (df['T2M_RANGE'] >= df['T2M_RANGE'].mean() - 3*df['T2M_RANGE'].
std())]

df = df[(df['TS'] <= df['TS'].mean() + 3*df['TS'].std()) &
        (df['TS'] >= df['TS'].mean() - 3* df['TS'].std())
df = df[(df['WS10M'] <= df['WS10M'].mean() + 3*df['WS10M'].std())] &
        (df['WS10M'] >= df['WS10M'].mean() - 3*df['WS10M'].std())

df = df[(df['WS10M_MAX'] <= df['WS10M_MAX'].mean() + 3*df['WS10M_MAX'].
std()) &
        (df['WS10M_MAX'] >= df['WS10M_MAX'].mean() -
3*df['WS10M_MAX'].std())]

df = df[(df['WS10M_MIN'] <= df['WS10M_MIN'].mean() + 3*df['WS10M_MIN'].
std()) &
        (df['WS10M_MIN'] >= df['WS10M_MIN'].mean() - 3*df['WS10M_MIN'].
std())]

df = df[(df['WS10M_RANGE'] <= df['WS10M_RANGE'].mean() + 3*df['WS10M_RA
NGE'].std()) &
        (df['WS10M_RANGE'] >= df['WS10M_RANGE'].mean() - 3*df['WS10M_RA
NGE'].std())]
df = df[(df['WS50M'] <= df['WS50M'].mean() + 3*df['WS50M'].std()) &
        (df['WS50M'] >= df['WS50M'].mean() - 3*df['WS50M'].std())]

df = df[(df['WS50M_MAX'] <= df['WS50M_MAX'].mean() + 3*df['WS50M_MAX'].
std()) &
        (df['WS50M_MAX'] >= df['WS50M_MAX'].mean() - 3*df['WS50M_MAX'].
std())]

df = df[(df['WS50M_MIN'] <= df['WS50M_MIN'].mean() + 3*df['WS50M_MIN'].
std()) &
        (df['WS50M_MIN'] >= df['WS50M_MIN'].mean() - 3*df['WS50M_MIN'].
std())]

df = df[(df['WS50M_RANGE'] <= df['WS50M_RANGE'].mean() + 3*df['WS50M_RA
NGE'].std()) &
        (df['WS50M_RANGE'] >= df['WS50M_RANGE'].mean() - 3*df['WS50M_RA
NGE'].std())]

print('Total rows = ',len(df.index))
```

```
df = df[(df['PRECTOT'] <= df['PRECTOT'].mean() + 3*df['PRECTOT'].std())
 &
        (df['PRECTOT'] >= df['PRECTOT'].mean() - 3*df['PRECTOT'].std())
]

df = df[(df['PS'] <= df['PS'].mean() + 3*df['PS'].std()) &
        (df['PS'] >= df['PS'].mean() - 3*df['PS'].std())]

df = df[(df['QV2M'] <= df['QV2M'].mean() + 3*df['QV2M'].std()) &
        (df['QV2M'] >= df['QV2M'].mean() - 3*df['QV2M'].std())]

df = df[(df['T2M'] <= df['T2M'].mean() + 3*df['T2M'].std()) &
        (df['T2M'] >= df['T2M'].mean() - 3*df['T2M'].std())]

df = df[(df['T2MDEW'] <= df['T2MDEW'].mean() + 3*df['T2MDEW'].std()) &
        (df['T2MDEW'] >= df['T2MDEW'].mean() - 3*df['T2MDEW'].std())]

df = df[(df['T2MWET'] <= df['T2MWET'].mean() + 3*df['T2MWET'].std()) &
        (df['T2MWET'] >= df['T2MWET'].mean() - 3*df['T2MWET'].std())]

df = df[(df['T2M_MAX'] <= df['T2M_MAX'].mean() + 3*df['T2M_MAX'].std())
 &
        (df['T2M_MAX'] >= df['T2M_MAX'].mean() - 3*df['T2M_MAX'].std())
]

df = df[(df['T2M_MIN'] <= df['T2M_MIN'].mean() + 3*df['T2M_MIN'].std())
 &
        (df['T2M_MIN'] >= df['T2M_MIN'].mean() - 3*df['T2M_MIN'].std())
]

df = df[(df['T2M_RANGE'] <= df['T2M_RANGE'].mean() + 3*df['T2M_RANGE'].
std()) &
        (df['T2M_RANGE'] >= df['T2M_RANGE'].mean() - 3*df['T2M_RANGE'].
std())]

df = df[(df['TS'] <= df['TS'].mean() + 3*df['TS'].std()) &
        (df['TS'] >= df['TS'].mean() - 3*df['TS'].std())]

df = df[(df['WS10M'] <= df['WS10M'].mean() + 3*df['WS10M'].std()) &
        (df['WS10M'] >= df['WS10M'].mean() - 3*df['WS10M'].std())]

df = df[(df['WS10M_MAX'] <= df['WS10M_MAX'].mean() + 3*df['WS10M_MAX'].
std()) &
        (df['WS10M_MAX'] >= df['WS10M_MAX'].mean() - 3*df['WS10M_MAX'].
std())]

df = df[(df['WS10M_MIN'] <= df['WS10M_MIN'].mean() + 3*df['WS10M_MIN'].
std()) &
```

```
        (df['WS10M_MIN'] >= df['WS10M_MIN'].mean() - 3*df['WS10M_MIN'].
std())]

df = df[(df['WS10M_RANGE'] <= df['WS10M_RANGE'].mean() + 3*df['WS10M_RA
NGE'].std()) &
        (df['WS10M_RANGE'] >= df['WS10M_RANGE'].mean() - 3*df['WS10M_RA
NGE'].std())]

df = df[(df['WS50M'] <= df['WS50M'].mean() + 3*df['WS50M'].std()) &
        (df['WS50M'] >= df['WS50M'].mean() - 3*df['WS50M'].std())]

df = df[(df['WS50M_MAX'] <= df['WS50M_MAX'].mean() + 3*df['WS50M_MAX'].
std()) &
        (df['WS50M_MAX'] >= df['WS50M_MAX'].mean() - 3*df['WS50M_MAX'].
std())]

df = df[(df['WS50M_MIN'] <= df['WS50M_MIN'].mean() + 3*df['WS50M_MIN'].
std()) &
        (df['WS50M_MIN'] >= df['WS50M_MIN'].mean() - 3*df['WS50M_MIN'].
std())]

df = df[(df['WS50M_RANGE'] <= df['WS50M_RANGE'].mean() + 3*df['WS50M_RA
NGE'].std()) &
        (df['WS50M_RANGE'] >= df['WS50M_RANGE'].mean() - 3*df['WS50M_RA
NGE'].std())]

print('Total rows = ',len(df.index))
categorical_column_list = ['score','year','month','day']
df_categorical = df[['score','year','month','day']]

plt.figure(figsize=(10,40))
for col_name in categorical_column_list:
    plt.figure()
    df_categorical[col_name].value_counts().plot(kind = 'bar')
    x_name = col_name
    y_name = 'Density'
    plt.xlabel(x_name)
    plt.ylabel(y_name)
    plt.title('Distribution of {x_name}'.format(x_name=x_name))
    plt.tight_layout()
plt.scatter(df['year'], df['score'], c ="blue")
plt.show()

plt.scatter(df['QV2M'], df['T2M'], c =df['score'])
plt.xlabel('QV2M')
plt.ylabel('T2M')
plt.title('Variation of T2M vs QV2M')
plt.show()
```

```
plt.scatter(df['T2M'], df['T2MDEW'], c =df['score'])
plt.xlabel('T2M')
plt.ylabel('T2MDEW')
plt.title('Variation of T2MDEW vs T2M')
plt.show()
temp_df = df[df['score']==5]
plt.scatter(df['WS10M'], df['WS50M'], c= df['score'])
plt.xlabel('WS10M')
plt.ylabel('WS50M')
plt.title('Variation of WS50M vs WS10M')
plt.show()
independent_variables = df.drop('score', 1)
independent_variables = independent_variables.drop('fips', 1)
independent_variables = independent_variables.drop('date', 1)
independent_variables.head()

target = df['score']
target.head()

target = df['score']
target.head()

print("Train features shape", X_train.shape)
print("Train target shape", y_train.shape)
print("Test features shape", X_test.shape)
print("Test target shape", y_test.shape)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_train

model = RandomForestClassifier(n_estimators=10) # n_estimators is the h
yperparameter

rfe = RFE(model, n_features_to_select=15) # n_features_to_select is cho
sen on a trial and error basis
fit = rfe.fit(X_train, y_train)
print("Num Features: %s" % (fit.n_features_))
print("Selected Features: %s" % (fit.support_))
print("Feature Ranking: %s" % (fit.ranking_))
selected_features = independent_variables.columns[(fit.get_support())]
print(selected_features)
independent_variables = independent_variables.drop('PRECTOT', 1)
independent_variables = independent_variables.drop('T2MWET', 1)
independent_variables = independent_variables.drop('WS10M_MAX', 1)
independent_variables = independent_variables.drop('WS10M_MIN', 1)
```

```
independent_variables = independent_variables.drop('WS50M_MIN', 1)
independent_variables = independent_variables.drop('month', 1)
independent_variables.head()
X_train, X_test, y_train, y_test = train_test_split(independent_variabl
es, target, test_size=0.2, random_state=0)

print("Train features shape", X_train.shape)
print("Train target shape", y_train.shape)
print("Test features shape", X_test.shape)
print("Test target shape", y_test.shape)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

sm = SMOTE(random_state = 5)
X_train_ures_SMOTE, y_train_ures_SMOTE = sm.fit_resample(X_train, y_tra
in.ravel())

print('Before OverSampling, the shape of train_X: {}'.format(X_train.sh
ape))
print('Before OverSampling, the shape of train_y: {} \n'.format(y_train
.shape))

print('After OverSampling, the shape of train_X: {}'.format(X_train_ure
s_SMOTE.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_
ures_SMOTE.shape))

print("Counts of label '0' - Before Oversampling:{}, After OverSampling
: {}".format(sum(y_train == 0),sum(y_train_ures_SMOTE == 0)))
print("Counts of label '1' - Before Oversampling:{}, After OverSampling
: {}".format(sum(y_train == 1),sum(y_train_ures_SMOTE == 1)))
print("Counts of label '2' - Before Oversampling:{}, After OverSampling
: {}".format(sum(y_train == 2),sum(y_train_ures_SMOTE == 2)))
print("Counts of label '3' - Before Oversampling:{}, After OverSampling
: {}".format(sum(y_train == 3),sum(y_train_ures_SMOTE == 3)))
print("Counts of label '4' - Before Oversampling:{}, After OverSampling
: {}".format(sum(y_train == 4),sum(y_train_ures_SMOTE == 4)))
print("Counts of label '5' - Before Oversampling:{}, After OverSampling
: {}".format(sum(y_train == 5),sum(y_train_ures_SMOTE == 5)))

undersample = NeighbourhoodCleaningRule(n_neighbors=3, threshold_cleani
ng=0.5)
X_train_dres, y_train_dres = undersample.fit_resample(X_train, y_train)
print('Before UnderSampling, the shape of train_X: {}'.format(X_train.s
hape))
```

```
print('Before UnderSampling, the shape of train_y: {} \n'.format(y_trai
n.shape))

print('After UnderSampling, the shape of train_X: {}'.format(X_train_dr
es.shape))
print('After UnderSampling, the shape of train_y: {} \n'.format(y_train
_dres.shape))

print("Counts of label '0' - Before UnderSampling:{}, After UnderSampli
ng: {}".format(sum(y_train == 0),sum(y_train_dres == 0)))
print("Counts of label '1' - Before UnderSampling:{}, After UnderSampli
ng: {}".format(sum(y_train == 1),sum(y_train_dres == 1)))
print("Counts of label '2' - Before UnderSampling:{}, After UnderSampli
ng: {}".format(sum(y_train == 2),sum(y_train_dres == 2)))
print("Counts of label '3' - Before UnderSampling:{}, After UnderSampli
ng: {}".format(sum(y_train == 3),sum(y_train_dres == 3)))
print("Counts of label '4' - Before UnderSampling:{}, After UnderSampli
ng: {}".format(sum(y_train == 4),sum(y_train_dres == 4)))
print("Counts of label '5' - Before UnderSampling:{}, After UnderSampli
ng: {}".format(sum(y_train == 5),sum(y_train_dres == 5)))

from imblearn.under_sampling import NearMiss
undersample = NearMiss()
X_train_dres_nm, y_train_dres_nm = undersample.fit_resample(X_train, y_
train)
print('Before UnderSampling, the shape of train_X: {}'.format(X_train.s
hape))
print('Before UnderSampling, the shape of train_y: {} \n'.format(y_trai
n.shape))

print('After UnderSampling, the shape of train_X: {}'.format(X_train_dr
es_nm.shape))
print('After UnderSampling, the shape of train_y: {} \n'.format(y_train
_dres_nm.shape))

print("Counts of label '0' - Before UnderSampling:{}, After UnderSampli
ng: {}".format(sum(y_train == 0),sum(y_train_dres_nm == 0)))
print("Counts of label '1' - Before UnderSampling:{}, After UnderSampli
ng: {}".format(sum(y_train == 1),sum(y_train_dres_nm == 1)))
print("Counts of label '2' - Before UnderSampling:{}, After UnderSampli
ng: {}".format(sum(y_train == 2),sum(y_train_dres_nm == 2)))
print("Counts of label '3' - Before UnderSampling:{}, After UnderSampli
ng: {}".format(sum(y_train == 3),sum(y_train_dres_nm == 3)))
print("Counts of label '4' - Before UnderSampling:{}, After UnderSampli
ng: {}".format(sum(y_train == 4),sum(y_train_dres_nm == 4)))
print("Counts of label '5' - Before UnderSampling:{}, After UnderSampli
ng: {}".format(sum(y_train == 5),sum(y_train_dres_nm == 5)))
pca = PCA()
```

```
X_train_dres_nm_PCAreduced = pca.fit_transform(X_train_dres_nm)
X_test_NM_PCA_transformed = pca.transform(X_test)
print(pca.explained_variance_ratio_)
pca = PCA(n_components=5)
X_train_dres_nm_PCAreduced = pca.fit_transform(X_train_dres_nm)
X_test_NM_PCA_transformed = pca.transform(X_test)

pca = PCA()
X_train_ures_SMOTE_PCAreduced = pca.fit_transform(X_train_ures_SMOTE)
X_test_SMOTE_PCA_transformed = pca.transform(X_test)
print(pca.explained_variance_ratio_)

pca = PCA(n_components=5)
X_train_ures_SMOTE_PCAreduced = pca.fit_transform(X_train_ures_SMOTE)
X_test_SMOTE_PCA_transformed = pca.transform(X_test)

print(pca.explained_variance_ratio_)
# poly_kpca = KernelPCA(kernel='poly')
# X_train_dres_nm_polykPCAreduced = poly_kpca.fit_transform(X_train_dre
s_nm)
# X_test_NM_polykPCA_transformed = poly_kpca.transform(X_test)

# print(poly_kpca.explained_variance_ratio_)

# poly_kpca = KernelPCA(kernel='poly')
# X_train_ures_SMOTE_polykPCAreduced = poly_kpca.fit_transform(X_train_
ures_SMOTE)

# X_test_SMOTE_polykPCA_transformed = poly_kpca.transform(X_test)

# print(poly_kpca.explained_variance_ratio)
# poly_kpca = KernelPCA(kernel='poly')
# X_train_polykPCAreduced = poly_kpca.fit_transform(X_train)
# X_test_polykPCA_transformed = poly_kpca.transform(X_test)
# print(poly_kpca.explained_variance_ratio_)

lda=LDA(n_components=5)
X_train_dres_nm_LDAreduced=lda.fit_transform(X_train_dres_nm,y_train_dr
es_nm)
X_test_NM_LDA_transformed=lda.transform(X_test)

print("Train features shape", X_train.shape)
print("LDA Dimensionality reduced features shape on Near Miss downsampl
ed data", X_train_dres_nm_LDAreduced.shape)
print("LDA Dimensionality reduced features shape on test data", X_test_
NM_LDA_transformed.shape)

lda=LDA(n_components=5)
```

```
X_train_ures_SMOTE_LDAreduced=lda.fit_transform(X_train_ures_SMOTE,y_tr
ain_ures_SMOTE)
X_test_SMOTE_LDA_transformed=lda.transform(X_test)
print("Train features shape", X_train.shape)
print("LDA Dimensionality reduced features shape on SMOTE Upsampled dat
a", X_train_ures_SMOTE_LDAreduced.shape)
print("LDA Dimensionality reduced features shape on test data", X_test_
NM_LDA_transformed.shape)
DT_classifier_NM = tree.DecisionTreeClassifier(criterion='gini')
DT_classifier_NM.fit(X_train_dres_nm,y_train_dres_nm)
y_pred_NM = DT_classifier_NM.predict(X_test)
pickle.dump(DT_classifier_NM, open('DT_classifier_NM.pkl', 'wb'))

print('Performance of Decision Tree Algorithm with Near Miss Downsampli
ng:\n')
print(confusion_matrix(y_test, y_pred_NM))
print(classification_report(y_test, y_pred_NM))
print('Accuracy:',accuracy_score(y_test, y_pred_NM))
print('Precision:',precision_score(y_test, y_pred_NM, average='weighted
'))
print('Recall:',recall_score(y_test, y_pred_NM, average='weighted'))
print('F1 Score:',f1_score(y_test, y_pred_NM, average='weighted'))
print('Cohen Kappa Score:',cohen_kappa_score(y_test, y_pred_NM))
fpr = dict()
tpr = dict()
thresh = dict()

for i in range(6):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, y_pred_NM, pos_label=
i)

plt.plot(fpr[0], tpr[0], linestyle='--
',color='orangered', label='Class 0 vs Rest')
plt.plot(fpr[1], tpr[1], linestyle='--
',color='green', label='Class 1 vs Rest')
plt.plot(fpr[2], tpr[2], linestyle='--
',color='blue', label='Class 2 vs Rest')
plt.plot(fpr[3], tpr[3], linestyle='--
',color='yellow', label='Class 3 vs Rest')
plt.plot(fpr[4], tpr[4], linestyle='--
',color='purple', label='Class 4 vs Rest')
plt.plot(fpr[5], tpr[5], linestyle='--
',color='magenta', label='Class 5 vs Rest')

plt.title('Multiclass ROC curve for Decision Tree with Near Miss Downsa
mpling')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
```

```
plt.legend(loc='best')
plt.savefig('Multiclass ROC curve for Decision Tree with Near Miss Down
sampling',dpi=300)
params = {
          'max_depth': [3, 5, 10, 20],
          'min_samples_leaf': [10, 20, 50, 100],
          'max_features':['log2','sqrt',None]
}
grid_search_DT_NM = GridSearchCV(estimator=DT_classifier_NM,
                           param_grid=params,
                           cv=4, n_jobs=-
1, verbose=1, scoring = "accuracy")
%%time
grid_search_DT_NM.fit(X_train_dres_nm,y_train_dres_nm)
score_df = pd.DataFrame(grid_search_DT_NM.cv_results_)
score_df.nlargest(5,"mean_test_score")
DT_classifier_SMOTE = tree.DecisionTreeClassifier(criterion='gini', max
_depth=70)
DT_classifier_SMOTE.fit(X_train_ures_SMOTE,y_train_ures_SMOTE)
y_pred_SMOTE = DT_classifier_SMOTE.predict(X_test)
pickle.dump(DT_classifier_SMOTE, open('DT_classifier_SMOTE.pkl', 'wb'))
print('Performance of Decision Tree Algorithm with SMOTE Upsampling:\n'
)
print(confusion_matrix(y_test, y_pred_SMOTE))
print(classification_report(y_test, y_pred_SMOTE))
print('Accuracy:',accuracy_score(y_test, y_pred_SMOTE))
print('Precision:',precision_score(y_test, y_pred_SMOTE, average='weigh
ted'))
print('Recall:',recall_score(y_test, y_pred_SMOTE, average='weighted'))
print('F1 Score:',f1_score(y_test, y_pred_SMOTE, average='weighted'))
print('Cohen Kappa Score:',cohen_kappa_score(y_test, y_pred_SMOTE))

fpr = dict()
tpr = dict()
thresh = dict()

for i in range(6):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, y_pred_SMOTE, pos_lab
el=i)

plt.plot(fpr[0], tpr[0], linestyle='--
',color='orangered', label='Class 0 vs Rest')
plt.plot(fpr[1], tpr[1], linestyle='--
',color='green', label='Class 1 vs Rest')
plt.plot(fpr[2], tpr[2], linestyle='--
',color='blue', label='Class 2 vs Rest')
plt.plot(fpr[3], tpr[3], linestyle='--
',color='yellow', label='Class 3 vs Rest')
```

```
plt.plot(fpr[4], tpr[4], linestyle='--
',color='purple', label='Class 4 vs Rest')
plt.plot(fpr[5], tpr[5], linestyle='--
',color='magenta', label='Class 5 vs Rest')

plt.title('Multiclass ROC curve for Decision Tree with SMOTE Upsampling
')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.savefig('Multiclass ROC curve for Decision Tree with SMOTE Upsampli
ng',dpi=300)

DT_classifier_NM_PCA = tree.DecisionTreeClassifier(criterion='gini')
DT_classifier_NM_PCA.fit(X_train_dres_nm_PCAreduced,y_train_dres_nm)
y_pred_NM_PCA = DT_classifier_NM_PCA.predict(X_test_NM_PCA_transformed)
pickle.dump(DT_classifier_NM_PCA, open('DT_classifier_NM_PCA.pkl', 'wb'
))
print('Performance of Decision Tree Algorithm with Near Miss Downsampli
ng and PCA:\n')
print(confusion_matrix(y_test, y_pred_NM_PCA))
print(confusion_matrix(y_test, y_pred_NM_PCA))
print(classification_report(y_test, y_pred_NM_PCA))
print('Accuracy:',accuracy_score(y_test, y_pred_NM_PCA))
print('Precision:',precision_score(y_test, y_pred_NM_PCA, average='weig
hted'))
print('Recall:',recall_score(y_test, y_pred_NM_PCA, average='weighted')
)
print('F1 Score:',f1_score(y_test, y_pred_NM_PCA, average='weighted'))
print('Cohen Kappa Score:',cohen_kappa_score(y_test, y_pred_NM_PCA))

fpr = dict()
tpr = dict()
thresh = dict()

for i in range(6):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, y_pred_NM_PCA, pos_la
bel=i)

plt.plot(fpr[0], tpr[0], linestyle='--
',color='orangered', label='Class 0 vs Rest')
plt.plot(fpr[1], tpr[1], linestyle='--
',color='green', label='Class 1 vs Rest')
plt.plot(fpr[2], tpr[2], linestyle='--
',color='blue', label='Class 2 vs Rest')
plt.plot(fpr[3], tpr[3], linestyle='--
',color='yellow', label='Class 3 vs Rest')
```

```
plt.plot(fpr[4], tpr[4], linestyle='--
',color='purple', label='Class 4 vs Rest')
plt.plot(fpr[5], tpr[5], linestyle='--
',color='magenta', label='Class 5 vs Rest')

plt.title('Multiclass ROC curve for Decision Tree with Near Miss Downsa
mpling and PCA')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.savefig('Multiclass ROC curve for Decision Tree with Near Miss Down
sampling and PCA',dpi=300)

DT_classifier_SMOTE_PCA = tree.DecisionTreeClassifier(criterion='gini')
DT_classifier_SMOTE_PCA.fit(X_train_ures_SMOTE_PCAreduced,y_train_ures_
SMOTE)
y_pred_SMOTE_PCA = DT_classifier_S
pickle.dump(DT_classifier_SMOTE_PCA, open('DT_classifier_SMOTE_PCA.pkl'
, 'wb'))

print('Performance of Decision Tree Algorithm with SMOTE Upsampling and
 PCA:\n')
print(confusion_matrix(y_test, y_pred_SMOTE_PCA))
print(confusion_matrix(y_test, y_pred_SMOTE_PCA))
print(classification_report(y_test, y_pred_SMOTE_PCA))
print('Accuracy:',accuracy_score(y_test, y_pred_SMOTE_PCA))
print('Precision:',precision_score(y_test, y_pred_SMOTE_PCA, average='w
eighted'))
print('Recall:',recall_score(y_test, y_pred_SMOTE_PCA, average='weighte
d'))
print('F1 Score:',f1_score(y_test, y_pred_SMOTE_PCA, average='weighted'
))
print('Cohen Kappa Score:',cohen_kappa_score(y_test, y_pred_SMOTE_PCA))
fpr = dict()
tpr = dict()
thresh = dict()

for i in range(6):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, y_pred_SMOTE_PCA, pos
_label=i)

plt.plot(fpr[0], tpr[0], linestyle='--
',color='orangered', label='Class 0 vs Rest')
plt.plot(fpr[1], tpr[1], linestyle='--
',color='green', label='Class 1 vs Rest')
plt.plot(fpr[2], tpr[2], linestyle='--
',color='blue', label='Class 2 vs Rest')
```

```
plt.plot(fpr[3], tpr[3], linestyle='--
',color='yellow', label='Class 3 vs Rest')
plt.plot(fpr[4], tpr[4], linestyle='--
',color='purple', label='Class 4 vs Rest')
plt.plot(fpr[5], tpr[5], linestyle='--
',color='magenta', label='Class 5 vs Rest')

plt.title('Multiclass ROC curve for Decision Tree with SMOTE Upsampling
 and PCA')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.savefig('Multiclass ROC curve for Decision Tree with SMOTE Upsampli
ng and PCA',dpi=300)
DT_classifier_NM_LDA = tree.DecisionTreeClassifier(criterion='gini')
DT_classifier_NM_LDA.fit(X_train_dres_nm_LDAreduced,y_train_dres_nm)
y_pred_NM_LDA = DT_classifier_NM_LDA.predict(X_test_NM_LDA_transformed)
print('Performance of Decision Tree Algorithm with Near Miss Downsampli
ng and LDA:\n')
print(confusion_matrix(y_test, y_pred_NM_LDA))
print(confusion_matrix(y_test, y_pred_NM_LDA))
print(classification_report(y_test, y_pred_NM_LDA))
print('Accuracy:',accuracy_score(y_test, y_pred_NM_LDA))
print('Precision:',precision_score(y_test, y_pred_NM_LDA, average='weig
hted'))
print('Recall:',recall_score(y_test, y_pred_NM_LDA, average='weighted')
)
print('F1 Score:',f1_score(y_test, y_pred_NM_LDA, average='weighted'))
print('Cohen Kappa Score:',cohen_kappa_score(y_test, y_pred_NM_LDA))

fpr = dict()
tpr = dict()
thresh = dict()

for i in range(6):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, y_pred_NM_LDA, pos_la
bel=i)

plt.plot(fpr[0], tpr[0], linestyle='--
',color='orangered', label='Class 0 vs Rest')
plt.plot(fpr[1], tpr[1], linestyle='--
',color='green', label='Class 1 vs Rest')
plt.plot(fpr[2], tpr[2], linestyle='--
',color='blue', label='Class 2 vs Rest')
plt.plot(fpr[3], tpr[3], linestyle='--
',color='yellow', label='Class 3 vs Rest')
plt.plot(fpr[4], tpr[4], linestyle='--
',color='purple', label='Class 4 vs Rest')
```

```
plt.plot(fpr[5], tpr[5], linestyle='--
',color='magenta', label='Class 5 vs Rest')

plt.title('Multiclass ROC curve for Decision Tree with Near Miss Downsa
mpling and LDA')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.savefig('Multiclass ROC curve for Decision Tree with Near Miss Down
sampling and LDA',dpi=300)

DT_classifier_SMOTE_LDA = tree.DecisionTreeClassifier(criterion='gini')
DT_classifier_SMOTE_LDA.fit(X_train_ures_SMOTE_LDAreduced,y_train_ures_
SMOTE)
y_pred_SMOTE_LDA = DT_classifier_SMOTE_LDA.predict(X_test_SMOTE_LDA_tra
nsformed)
pickle.dump(DT_classifier_SMOTE_LDA, open('DT_classifier_SMOTE_LDA.pkl'
, 'wb'))
print('Performance of Decision Tree Algorithm with SMOTE Upsampling and
 LDA:\n')
print(confusion_matrix(y_test, y_pred_SMOTE_LDA))
print(confusion_matrix(y_test, y_pred_SMOTE_LDA))
print(classification_report(y_test, y_pred_SMOTE_LDA))
print('Accuracy:',accuracy_score(y_test, y_pred_SMOTE_LDA))
print('Precision:',precision_score(y_test, y_pred_SMOTE_LDA, average='w
eighted'))
print('Recall:',recall_score(y_test, y_pred_SMOTE_LDA, average='weighte
d'))
print('F1 Score:',f1_score(y_test, y_pred_SMOTE_LDA, average='weighted'
))
print('Cohen Kappa Score:',cohen_kappa_score(y_test, y_pred_SMOTE_LDA))
fpr = dict()
tpr = dict()
thresh = dict()

for i in range(6):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, y_pred_SMOTE_LDA, pos
_label=i)

plt.plot(fpr[0], tpr[0], linestyle='--
',color='orangered', label='Class 0 vs Rest')
plt.plot(fpr[1], tpr[1], linestyle='--
',color='green', label='Class 1 vs Rest')
plt.plot(fpr[2], tpr[2], linestyle='--
',color='blue', label='Class 2 vs Rest')
plt.plot(fpr[3], tpr[3], linestyle='--
',color='yellow', label='Class 3 vs Rest')
```

```
plt.plot(fpr[4], tpr[4], linestyle='--
',color='purple', label='Class 4 vs Rest')
plt.plot(fpr[5], tpr[5], linestyle='--
',color='magenta', label='Class 5 vs Rest')

plt.title('Multiclass ROC curve for Decision Tree with SMOTE Upsampling
 and LDA')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.savefig('Multiclass ROC curve for Decision Tree with SMOTE Upsampli
ng and LDA',dpi=300)

DT_classifier.get_depth()
params = {
          'max_depth': [40, 50, 60, 70, 80],
#           'max_samples_leaf': [, 20, 50, 100],
          'max_features':['log2','sqrt',None]
}
grid_search = GridSearchCV(estimator=DT_classifier,
                           param_grid=params,
                           cv=4, n_jobs=-
1, verbose=1, scoring = "accuracy")
%%time
grid_search.fit(X_train,y_train)
score_df = pd.DataFrame(grid_search.cv_results_)
score_df.nlargest(5,"mean_test_score")
DT_classifier = tree.DecisionTreeClassifier(criterion='gini', max_depth
=70)
DT_classifier.fit(X_train,y_train)
y_pred_DT = DT_classifier.predict(X_test)

print('Performance of Decision Tree Algorithm without resampling - Afte
r Hyperparameter Tuning:\n')
print(confusion_matrix(y_test, y_pred_DT))
print(classification_report(y_test, y_pred_DT))
print('Accuracy:',accuracy_score(y_test, y_pred_DT))
print('Precision:',precision_score(y_test, y_pred_DT, average='weighted
'))
print('Recall:',recall_score(y_test, y_pred_DT, average='weighted'))
print('F1 Score:',f1_score(y_test, y_pred_DT, average='weighted'))
print('Cohen Kappa Score:',cohen_kappa_score(y_test, y_pred_DT))
fpr = dict()
tpr = dict()
thresh = dict()

for i in range(6):
```

```
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, y_pred_DT, pos_label=
i)

plt.plot(fpr[0], tpr[0], linestyle='--
',color='orangered', label='Class 0 vs Rest')
plt.plot(fpr[1], tpr[1], linestyle='--
',color='green', label='Class 1 vs Rest')
plt.plot(fpr[2], tpr[2], linestyle='--
',color='blue', label='Class 2 vs Rest')
plt.plot(fpr[3], tpr[3], linestyle='--
',color='yellow', label='Class 3 vs Rest')
plt.plot(fpr[4], tpr[4], linestyle='--
',color='purple', label='Class 4 vs Rest')
plt.plot(fpr[5], tpr[5], linestyle='--
',color='magenta', label='Class 5 vs Rest')

plt.title('Multiclass ROC curve for Decision Tree without resampling -
After Hyperparameter Tuning')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.savefig('Multiclass ROC curve for Decision Tree without resampling
- After Hyperparameter Tuning',dpi=300)
knn_classifier = KNeighborsClassifier(n_neighbors=5, p=2, metric='minko
wski')
knn_classifier.fit(X_train, y_train)
y_pred_knn = knn_classifier.predict(X_test)
print('Performance of KNN Algorithm without resampling:\n')
print(confusion_matrix(y_test, y_pred_knn))
print(classification_report(y_test, y_pred_knn))
print('Accuracy:',accuracy_score(y_test, y_pred_knn))
print('Precision:',precision_score(y_test, y_pred_knn, average='weighte
d'))
print('Recall:',recall_score(y_test, y_pred_knn, average='weighted'))
print('F1 Score:',f1_score(y_test, y_pred_knn, average='weighted'))
print('Cohen Kappa Score:',cohen_kappa_score(y_test, y_pred_knn))
fpr = dict()
tpr = dict()
thresh = dict()

for i in range(6):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, y_pred_knn, pos_label
=i)

plt.plot(fpr[0], tpr[0], linestyle='--
',color='orangered', label='Class 0 vs Rest')
plt.plot(fpr[1], tpr[1], linestyle='--
',color='green', label='Class 1 vs Rest')
```

```
plt.plot(fpr[2], tpr[2], linestyle='--
',color='blue', label='Class 2 vs Rest')
plt.plot(fpr[3], tpr[3], linestyle='--
',color='yellow', label='Class 3 vs Rest')
plt.plot(fpr[4], tpr[4], linestyle='--
',color='purple', label='Class 4 vs Rest')
plt.plot(fpr[5], tpr[5], linestyle='--
',color='magenta', label='Class 5 vs Rest')

plt.title('Multiclass ROC curve for KNN without resampling')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.savefig('Multiclass ROC curve for KNN without resampling',dpi=300)
k_range = list(range(1, 10))
param_grid = dict(n_neighbors=k_range)

grid = GridSearchCV(knn_classifier, param_grid, cv=3, scoring='accuracy
', return_train_score=False,verbose=1)
grid_search=grid.fit(X_train, y_train)
score_df = pd.DataFrame(grid_search.cv_results_)
score_df.nlargest(5,"mean_test_score")

knn_classifier = KNeighborsClassifier(n_neighbors=1, p=2, metric='minko
wski')
knn_classifier.fit(X_train, y_train)
y_pred_knn = knn_classifier.predict(X_test)
print('Performance of KNN Algorithm without resampling - After Hyperpar
ameter Tuning:\n')
print(confusion_matrix(y_test, y_pred_knn))
print(classification_report(y_test, y_pred_knn))
print('Accuracy:',accuracy_score(y_test, y_pred_knn))
print('Precision:',precision_score(y_test, y_pred_knn, average='weighte
d'))
print('Recall:',recall_score(y_test, y_pred_knn, average='weighted'))
print('F1 Score:',f1_score(y_test, y_pred_knn, average='weighted'))
print('Cohen Kappa Score:',cohen_kappa_score(y_test, y_pred_knn))
fpr = dict()
tpr = dict()
thresh = dict()

for i in range(6):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, y_pred_knn, pos_label
=i)

plt.plot(fpr[0], tpr[0], linestyle='--
',color='orangered', label='Class 0 vs Rest')
```

```
plt.plot(fpr[1], tpr[1], linestyle='--
',color='green', label='Class 1 vs Rest')
plt.plot(fpr[2], tpr[2], linestyle='--
',color='blue', label='Class 2 vs Rest')
plt.plot(fpr[3], tpr[3], linestyle='--
',color='yellow', label='Class 3 vs Rest')
plt.plot(fpr[4], tpr[4], linestyle='--
',color='purple', label='Class 4 vs Rest')
plt.plot(fpr[5], tpr[5], linestyle='--
',color='magenta', label='Class 5 vs Rest')

plt.title('Multiclass ROC curve for KNN without resampling - After Hype
rparameter Tuning')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.savefig('Multiclass ROC curve for KNN without resampling - After Hy
perparameter Tuning',dpi=300)
knn_classifier_SMOTE = KNeighborsClassifier(n_neighbors=1, p=2, metric=
'minkowski')
knn_classifier_SMOTE.fit(X_train_ures_SMOTE, y_train_ures_SMOTE)
y_pred_knn_SMOTE = knn_classifier_SMOTE.predict(X_test)
pickle.dump(knn_classifier_SMOTE, open('knn_classifier_SMOTE.pkl', 'wb'
))
print('Performance of KNN Algorithm with SMOTE Upsampling:\n')
print(confusion_matrix(y_test, y_pred_knn_SMOTE))
print(classification_report(y_test, y_pred_knn_SMOTE))
print('Accuracy:',accuracy_score(y_test, y_pred_knn_SMOTE))
print('Precision:',precision_score(y_test, y_pred_knn_SMOTE, average='w
eighted'))
print('Recall:',recall_score(y_test, y_pred_knn_SMOTE, average='weighte
d'))
print('F1 Score:',f1_score(y_test, y_pred_knn_SMOTE, average='weighted'
))
print('Cohen Kappa Score:',cohen_kappa_score(y_test, y_pred_knn_SMOTE))
fpr = dict()
tpr = dict()
thresh = dict()

for i in range(6):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, y_pred_knn_SMOTE, pos
_label=i)

plt.plot(fpr[0], tpr[0], linestyle='--
',color='orangered', label='Class 0 vs Rest')
plt.plot(fpr[1], tpr[1], linestyle='--
',color='green', label='Class 1 vs Rest')
```

```
plt.plot(fpr[2], tpr[2], linestyle='--
',color='blue', label='Class 2 vs Rest')
plt.plot(fpr[3], tpr[3], linestyle='--
',color='yellow', label='Class 3 vs Rest')
plt.plot(fpr[4], tpr[4], linestyle='--
',color='purple', label='Class 4 vs Rest')
plt.plot(fpr[5], tpr[5], linestyle='--
',color='magenta', label='Class 5 vs Rest')

plt.title('Multiclass ROC curve for KNN with SMOTE Upsampling')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.savefig('Multiclass ROC curve for KNN with SMOTE Upsampling',dpi=30
0)
knn_classifier_NM = KNeighborsClassifier(n_neighbors=1, p=2, metric='mi
nkowski')
knn_classifier_NM.fit(X_train_dres_nm, y_train_dres_nm)
y_pred_knn_NM = knn_classifier_NM.predict(X_test)
pickle.dump(knn_classifier_NM, open('knn_classifier_NM.pkl', 'wb'))
print('Performance of KNN Algorithm with NM Downsampling:\n')
print(confusion_matrix(y_test, y_pred_knn_NM))
print(classification_report(y_test, y_pred_knn_NM))
print('Accuracy:',accuracy_score(y_test, y_pred_knn_NM))
print('Precision:',precision_score(y_test, y_pred_knn_NM, average='weig
hted'))
print('Recall:',recall_score(y_test, y_pred_knn_NM, average='weighted')
)
print('F1 Score:',f1_score(y_test, y_pred_knn_NM, average='weighted'))
print('Cohen Kappa Score:',cohen_kappa_score(y_test, y_pred_knn_NM))
fpr = dict()
tpr = dict()
thresh = dict()

for i in range(6):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, y_pred_knn_NM, pos_la
bel=i)

plt.plot(fpr[0], tpr[0], linestyle='--
',color='orangered', label='Class 0 vs Rest')
plt.plot(fpr[1], tpr[1], linestyle='--
',color='green', label='Class 1 vs Rest')
plt.plot(fpr[2], tpr[2], linestyle='--
',color='blue', label='Class 2 vs Rest')
plt.plot(fpr[3], tpr[3], linestyle='--
',color='yellow', label='Class 3 vs Rest')
plt.plot(fpr[4], tpr[4], linestyle='--
',color='purple', label='Class 4 vs Rest')
```

```
plt.plot(fpr[5], tpr[5], linestyle='--
',color='magenta', label='Class 5 vs Rest')

plt.title('Multiclass ROC curve for KNN with Near Miss Downsampling')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.savefig('Multiclass ROC curve for KNN with Near Miss Downsampling',
dpi=300)
RF_classifier = RandomForestClassifier(n_estimators = 20, max_depth=70,
 random_state=0)
RF_classifier.fit(X_train, y_train)
y_pred_RF = RF_classifier.predict(X_test)
print('Performance of RF Algorithm without resampling:\n')
print(confusion_matrix(y_test, y_pred_RF))
print(classification_report(y_test, y_pred_RF))
print('Accuracy:',accuracy_score(y_test, y_pred_RF))
print('Precision:',precision_score(y_test, y_pred_RF, average='weighted
'))
print('Recall:',recall_score(y_test, y_pred_RF, average='weighted'))
print('F1 Score:',f1_score(y_test, y_pred_RF, average='weighted'))
print('Cohen Kappa Score:',cohen_kappa_score(y_test, y_pred_RF))
fpr = dict()
tpr = dict()
thresh = dict()

for i in range(6):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, y_pred_RF, pos_label=
i)

plt.plot(fpr[0], tpr[0], linestyle='--
',color='orangered', label='Class 0 vs Rest')
plt.plot(fpr[1], tpr[1], linestyle='--
',color='green', label='Class 1 vs Rest')
plt.plot(fpr[2], tpr[2], linestyle='--
',color='blue', label='Class 2 vs Rest')
plt.plot(fpr[3], tpr[3], linestyle='--
',color='yellow', label='Class 3 vs Rest')
plt.plot(fpr[4], tpr[4], linestyle='--
',color='purple', label='Class 4 vs Rest')
plt.plot(fpr[5], tpr[5], linestyle='--
',color='magenta', label='Class 5 vs Rest')

plt.title('Multiclass ROC curve for Random Forest without resampling')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
```

```
plt.savefig('Multiclass ROC curve for Random Forest without resampling'
,dpi=300)
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 50, num
= 10)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
bootstrap = [True, False]
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'bootstrap': bootstrap}
RF_random = RandomizedSearchCV(estimator = RF_classifier, param_distrib
utions = random_grid, n_iter = 20, cv = 3, verbose=2, random_state=0, n
_jobs = -1)
RF_random.fit(X_train, y_train)
RF_random.best_params_
RF_classifier = RandomForestClassifier(n_estimators = 50, max_depth=80,
 bootstrap=False, max_features='sqrt', random_state=0)
RF_classifier.fit(X_train, y_train)
y_pred_RF = RF_classifier.predict(X_test)
pickle.dump(RF_classifier, open('RF_classifier.pkl', 'wb'))
print('Performance of RF Algorithm without resampling - After Hyperpara
mter Tuning:\n')
print(confusion_matrix(y_test, y_pred_RF))
print(classification_report(y_test, y_pred_RF))
print('Accuracy:',accuracy_score(y_test, y_pred_RF))
print('Precision:',precision_score(y_test, y_pred_RF, average='weighted
'))
print('Recall:',recall_score(y_test, y_pred_RF, average='weighted'))
print('F1 Score:',f1_score(y_test, y_pred_RF, average='weighted'))
print('Cohen Kappa Score:',cohen_kappa_score(y_test, y_pred_RF))
fpr = dict()
tpr = dict()
thresh = dict()

for i in range(6):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, y_pred_RF, pos_label=
i)

plt.plot(fpr[0], tpr[0], linestyle='--
',color='orangered', label='Class 0 vs Rest')
plt.plot(fpr[1], tpr[1], linestyle='--
',color='green', label='Class 1 vs Rest')
plt.plot(fpr[2], tpr[2], linestyle='--
',color='blue', label='Class 2 vs Rest')
plt.plot(fpr[3], tpr[3], linestyle='--
',color='yellow', label='Class 3 vs Rest')
```

```
plt.plot(fpr[4], tpr[4], linestyle='--
',color='purple', label='Class 4 vs Rest')
plt.plot(fpr[5], tpr[5], linestyle='--
',color='magenta', label='Class 5 vs Rest')

plt.title('Multiclass ROC curve for Random Forest without resampling -
After Hyperparameter Tuning')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.savefig('Multiclass ROC curve for Random Forest without resampling
- After Hyperparameter Tuning',dpi=300)

# svm_classifier = SVC(kernel='poly', degree=3, C = 1.0)
# svm_classifier.fit(X_train, y_train)
# y_pred_svm = svm_classifier.predict(X_test)

# print('Performance of SVM Algorithm without resampling:\n')
# print(confusion_matrix(y_test, y_pred_svm))
# print(classification_report(y_test, y_pred_svm))
# print('Accuracy:',accuracy_score(y_test, y_pred_svm))
# print('Precision:',precision_score(y_test, y_pred_svm, average='weigh
ted'))
# print('Recall:',recall_score(y_test, y_pred_svm, average='weighted'))
# print('F1 Score:',f1_score(y_test, y_pred_svm, average='weighted'))
# print('Cohen Kappa Score:',cohen_kappa_score(y_test, y_pred_svm))
# svm_classifier = SVC(kernel='rbf', C = 1.0)
# svm_classifier.fit(X_train, y_train)
# y_pred_svm_rbf = svm_classifier.predict(X_test)
# print('Performance of SVM Algorithm with RBF Kernel without resamplin
g:\n')
# print(confusion_matrix(y_test, y_pred_svm_rbf))
# print(classification_report(y_test, y_pred_svm_rbf))
# print('Accuracy:',accuracy_score(y_test, y_pred_svm_rbf))
# print('Precision:',precision_score(y_test, y_pred_svm_rbf, average='w
eighted'))
# print('Recall:',recall_score(y_test, y_pred_svm_rbf, average='weighte
d'))
# print('F1 Score:',f1_score(y_test, y_pred_svm_rbf, average='weighted'
))
# print('Cohen Kappa Score:',cohen_kappa_score(y_test, y_pred_svm_rbf))
svm_classifier_nm = SVC(kernel='poly', degree=3, C = 1.0)
svm_classifier_nm.fit(X_train_dres_nm, y_train_dres_nm)
y_pred_svm_nm = svm_classifier_nm.predict(X_test)
pickle.dump(svm_classifier_nm, open('svm_classifier_nm.pkl', 'wb'))
print('Performance of SVM Algorithm with Near Miss downsampling:\n')
print(confusion_matrix(y_test, y_pred_svm_nm))
print(classification_report(y_test, y_pred_svm_nm))
```

```
print('Accuracy:',accuracy_score(y_test, y_pred_svm_nm))
print('Precision:',precision_score(y_test, y_pred_svm_nm, average='weig
hted'))
print('Recall:',recall_score(y_test, y_pred_svm_nm, average='weighted')
)
print('F1 Score:',f1_score(y_test, y_pred_svm_nm, average='weighted'))
print('Cohen Kappa Score:',cohen_kappa_score(y_test, y_pred_svm_nm))

fpr = dict()
tpr = dict()
thresh = dict()

for i in range(6):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, y_pred_svm_nm, pos_la
bel=i)

plt.plot(fpr[0], tpr[0], linestyle='--
',color='orangered', label='Class 0 vs Rest')
plt.plot(fpr[1], tpr[1], linestyle='--
',color='green', label='Class 1 vs Rest')
plt.plot(fpr[2], tpr[2], linestyle='--
',color='blue', label='Class 2 vs Rest')
plt.plot(fpr[3], tpr[3], linestyle='--
',color='yellow', label='Class 3 vs Rest')
plt.plot(fpr[4], tpr[4], linestyle='--
',color='purple', label='Class 4 vs Rest')
plt.plot(fpr[5], tpr[5], linestyle='--
',color='magenta', label='Class 5 vs Rest')

plt.title('Multiclass ROC curve for SVM with Near Miss Downsampling')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.savefig('Multiclass ROC curve for SVM with Near Miss Downsampling',
dpi=300)

all_labels = pd.DataFrame()
all_labels['Actual_label'] = y_test
all_labels['y_pred_DT'] = y_pred_DT
all_labels['y_pred_NM'] = y_pred_NM
all_labels['y_pred_SMOTE'] = y_pred_SMOTE
all_labels['y_pred_NM_PCA'] = y_pred_NM_PCA
all_labels['y_pred_SMOTE_PCA'] = y_pred_SMOTE_PCA
all_labels['y_pred_NM_LDA'] = y_pred_NM_LDA
all_labels['y_pred_SMOTE_LDA'] = y_pred_SMOTE_LDA
all_labels['y_pred_KNN'] = y_pred_knn
all_labels['y_pred_KNN_SMOTE'] = y_pred_knn_SMOTE
all_labels['y_pred_KNN_NM'] = y_pred_knn_NM
```

```
all_labels['y_pred_NB'] = y_pred_NB
all_labels['y_pred_RF'] = y_pred_RF
# all_labels['y_pred_svm_nm'] = y_pred_svm_nm
# all_labels[['y_pred_DT','Agglomerative_min_labels','Agglomerative_max
_labels']]=independent_variables[['Agglomerative_labels','Agglomerative
_min_labels','Agglomerative_max_labels']]
data = [
{'Algorithm':'Decision Tree without resampling' ,'Accuracy':0.763336889
83729,'Precision':0.7623049559359242,'Recall':0.76333688983729,'F1 Scor
e':0.7628094905920674,'Cohen Kappa Score':0.596681340983346},
{'Algorithm':'Decision Tree with Near Miss Downsampling','Accuracy':0.2
2480540265282864,'Precision':0.5431846016633978,'Recall':0.224805402652
82864,'F1 Score':0.2626001987113276,'Cohen Kappa Score':0.0787595709141
8129},
{'Algorithm':'Decision Tree with SMOTE Upsampling','Accuracy':0.7642280
365673271,'Precision':0.7725880165635359,'Recall':0.7642280365673271,'F
1 Score':0.7679865604188995,'Cohen Kappa Score':0.6072223355459014},
{'Algorithm':'Decision Tree with Near Miss Downsampling and PCA','Accur
acy':0.18901606084854952,'Precision':0.5208938985991317,'Recall':0.1890
1606084854952,'F1 Score':0.22407231601991243,'Cohen Kappa Score':0.0597
2201042990921},
{'Algorithm':'Decision Tree with SMOTE Upsampling and PCA','Accuracy':0
.6911580461860537,'Precision':0.721815047933934,'Recall':0.691158046186
0537,'F1 Score':0.7032986584993112,'Cohen Kappa Score':0.50450374824989
97},
{'Algorithm':'Decision Tree with Near Miss Downsampling and LDA','Accur
acy':0.20474752863389833,'Precision':0.5142486750813464,'Recall':0.2047
4752863389833,'F1 Score':0.24971268616453107,'Cohen Kappa Score':0.0577
7192884677773},
{'Algorithm':'Decsion Tree with SMOTE Upsampling and LDA','Accuracy':0.
6028092339775455,'Precision':0.6746275672595141,'Recall':0.602809233977
5455,'F1 Score':0.6283270075892382,'Cohen Kappa Score':0.39472278972161
27},
{'Algorithm':'KNN without resampling','Accuracy':0.7986513575337262,'Pr
ecision':0.7982935187700809,'Recall':0.7986513575337262,'F1 Score':0.79
84710410835046,'Cohen Kappa Score':0.6574980649397748},
{'Algorithm':'KNN with SMOTE Upsampling','Accuracy':0.7952666165522927,
'Precision':0.801758151083889,'Recall':0.7952666165522927,'F1 Score':0.
7981975830544615,'Cohen Kappa Score':0.6578269982214404},
{'Algorithm':'KNN with Near Miss Upsampling','Accuracy':0.2325084669043
058,'Precision':0.5664887557511156,'Recall':0.2325084669043058,'F1 Scor
e':0.2688785664243745,'Cohen Kappa Score':0.09355157402001324},
{'Algorithm':'Naive Bayes without resampling','Accuracy':0.585143917165
7896,'Precision':0.4499104487639562,'Recall':0.5851439171657896,'F1 Sco
re':0.4804411924156227,'Cohen Kappa Score':0.08074571293428756},
{'Algorithm':'Random Forest without resampling','Accuracy':0.8089591567
852438,'Precision':0.7969254562173812,'Recall':0.8089591567852438,'F1 S
core':0.7986904178915076,'Cohen Kappa Score':0.6549810654516983},
```

```
{'Algorithm':'SVM with Near Miss Downsampling','Accuracy':0.2995344213002255,'Precision':0.5123237426347645,'Recall':0.2995344213002255,'F1 Score':0.36286713356946726,'Cohen Kappa Score':0.07811063221491454}]
performance_metrics = pd.DataFrame(data)
performance_metrics.sort_values(by=['Accuracy', 'Cohen Kappa Score'], ascending=False)
```

REFERENCS:

1.)https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b94[Author:Ayush Pant
Date:11/01/2019]
2.)https://www.run.ai/guides/machine-learning-engineering/machine-learning-workflow[Author:Group Al learning Europe
Date:12/3/2017]
3.)https://ml-ops.org/content/end-to-end-ml-workflow[Author:Ml ops Date:-]
4.)https://www.kaggle.com/general/280496[Author:Kaggle group]
5.)https://aws.amazon.com/blogs/machine-learning/architect-and-build-the-full-machine-learning-lifecycle-with-amazon-sagemaker/%5BAuthor:Aws%5D/