DEVELOPMENT OF A MACHINE LEARNING ALGORITHM TO MINIMIZE RUNOFF THROUGH AN AUTOMATED SMART IRRIGATION SYSTEM

A Thesis

by

SAMBANDH BHUSAN DHAL

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Ulisses Braga Neto
Co-Chair of Committee,	Jorge Alvarado
Committee Members,	Nicholas G. Duffield
	Stavros Kalafatis
	Benjamin Wherley
Head of Department,	Miroslav M. Begovic

December 2019

Major Subject: Electrical Engineering

ABSTRACT

The study of proper water management practices is of prime importance due to the ever-increasing population and rapid industrialization which results in shortage of portable water supplies throughout the world. The current water supplies are not expected to meet the increasing demand in the upcoming decades which could in result affect the socio-economic stability and have a detrimental effect on human livelihood.

About 30% of the current municipal supplies in the world are used for outdoor irrigation activities such as gardening and landscaping purposes. These numbers are on the rise due to the ever increasing human population. Due to the current inefficient landscape practices, substantial amount of water is lost in the form of runoff. This poses a great threat to the environment with its potential for transporting fertilizers and pesticides into storm sewers and, eventually, surface waters. Thus, this study focuses on designing a Machine Learning approach which would act as a Decision Support System (DSS) to irrigate turfgrasses to minimize runoff in the plots while maintaining the quality of the turfgrasses.

For this, a robust Machine Learning approach named as Radial Basis Function - Support Vector Machine (RBF-SVM) was proposed which was trained on the synthetic data generated from the datapoints recorded during the year 2015-16 and 2016-17 at the Turfgrass Laboratory in Texas A & M University, College Station. For each of the approaches, the target variable was changed and the number of features were varied in each case to see which gives the best results. Among all the target variables, predicting the Soil Wetting Efficiency Index, devised by Wherley, *et. al.*[33] was the most applicable as it is one of the most generic approaches since it is not site-specific and gave the highest validation testing accuracy of 90%. Thus, the latter approach was

used in the ASIS controller to observe the robustness of the algorithm in controlling the effectiveness of the irrigation cycle.

Until now, only few irrigation cycles have been scheduled and experimental data are still being collected from the facility. Preliminary results suggested that the Machine Learning algorithm has the potential to save water as it helped in efficient regulation of irrigation cycles and even achieved a goal of zero runoff in two of the irrigation runs. The Green Cover percentage of the plots where the proposed ASIS controller was mounted showed an increment of about 12%, thereby validating the fact that the quality of turfgrasses was also maintained. With more irrigation cycles which would be scheduled over time, the proposed Machine Learning approach is expected to perform better with increase in observations and may nullify runoff eventually.

DEDICATION

To my mother, father, brother and my guru,

Puspanjali, Bibhuti, Siddhant and Prof. Alvarado,

My strengths, my faith, my joy

ACKNOWLEDGEMENTS

Firstly, I would like to thank my advisor and co-advisor, Dr. Ulisses Braga Neto and Dr. Jorge Alvarado for their patience, guidance and for being a constant source of support throughout my thesis. I thank them for believing in me and giving me a chance to work on this project. They have always motivated me to work hard, do better and implement ideas out of the box. I am impressed with their work ethics and group dynamics and I look forward to incorporate these qualities in my professional life.

The other student in Dr. Alvarado's group, Tomas Reyes, a Senior in Electrical and Computer Engineering, has been the key contributor to the project. The design of the entire smart controller was done by him single-handedly and this project would not have been a success of this kind without him. He has been cooperative and diligent throughout and his inputs have been very instrumental in designing my approach. As the project included scrapping of historical weather data, Shubham Jain, a PhD student in Geosciences, has been instrumental in helping me collect the data to begin with. I would also like to express my sincere gratitude to Rupali Sahu, a PhD student in Aerospace Engineering for helping me with her presentation and artistic skills, which have been a huge help for my thesis and for being my support and inspiration ceaselessly.

I would like to thank Dr. Benjamin Wherley, Dr. Nicholas G. Duffield and Dr. Stavros Kalafatis for serving on my committee. Since the project needed a lot of domain knowledge about irrigating turfgrasses, Dr. Wherley's expertise has not only been instrumental in understanding the practical side of the project but his constant intervention has always assured that we progressed on the right track. Dr. Duffield taught me Data Mining and Analysis which deepened my interest in the field and gave me the confidence to carry out different approaches in the project. Dr.Kalafatis'

research on optimization tasks related to the field of architecture enlightened me about its application in various scenarios and related research in the field.

I would like to extend my hearty thanks to all my friends who made my time in College Station enjoyable and fun. Being an international student, transition to the culture in the United States was not easy. Given the amount of course load and research in graduate school, life would not have been easy without you all. Thank you Radhika, Abhishek, Sahil, Soniya, Ayush, Nikita, Priya, Siddharth, Jay, Aishwarya, Hemant, Mahesh, Anirudh, Ajinkya, Akash, Keyur, Rahul, Anoop, Arjun, Keerthi, Dwarkanath and Devansh for standing by me in my tough times and for lifting my spirit up. Also, I would like to thank my friends from India; Abhishek, Chandan, Vineeta and Utkarsh who have always kept in touch and provided me the encouragement and support to pursue my dreams.

Lastly, I thank my mother Dr. Puspanjali Jena for motivating me to do research. Also, I would like to thank my father Bibhuti Bhusan Dhal and my younger brother Siddhant for believing in me and pursuing me to study abroad and pursue research. The thesis, for all its worth, is dedicated to them.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Dr. Ulisses Braga-Neto [advisor] of the Department of Electrical and Computer Engineering and Dr. Jorge Alvarado [co-advisor] of the Department of Engineering Technology and Industrial Distribution and also Dr. Benjamin Wherley of the Department of Soil and Crop Sciences and Dr. Nicholas Duffield and Prof. Stavros Kalafatis of Electrical and Computer Engineering.

The design and programming of the smart controller in Chapter 4 was provided by Tomas Reyes, an undergraduate student in the Department of Electrical and Computer Engineering.

All other work for the thesis was completed by me independently.

Funding Sources

This work was also made possible in part by Account Number 28-163043-00001 under Water Seed Grant by Dr. Alvarado and in part by Account Number 114252-95180 under Dr. Wherley.

NOMENCLATURE

DI	Deficit Irrigation
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface
ML	Machine Learning
TAMU	Texas Agricultural and Mechanical University
SVM	Support Vector Machine
ET	Evapotranspiration
TM	Thermatic Mapper
ANN	Artificial Neural Network
UNWC	United Nations Watercourses Convention
UNECE	United Nations Economic Commission for Europe
SDG	Sustainable Development Goals
LID	Low Impact Development
BMP	Best Management Practices
SWMM5	Storm Water Management Model 5
GAP	Good Agricultural Practices
TSS	Total Suspended Solids
TN	Total Nitrogen
CW	Constructed Wetlands
RNN	Recurrent Neural Network
LM	Levenberg-Marquardt

BP	Back Propagation
DTDNN	Distributed Time Delay Neural Network
EM	Expected Maximization
MI	Multiple Imputation
KNN	K Nearest Neighbors
MRI	Magnetic Resonance Imaging
MDS	Multi-Dimensional Scaling
PCA	Principal Component Analysis
MCMC	Markov Chain Monte Carlo
MNIST	Modified National Institute of Standards and Technology
SMOTE	Synthetic Minority Over-Sampling Technique
LIRMS	Landscape Irrigation Runoff Mitigation System
GRIDMET	University of Idaho Gridded Surface Meteorological Dataset
ET0	Reference Evapotranspiration Rate
VPD	Vapor Pressure Deficit
SWEI	Soil Wetting Efficiency Index
MDG	Millennium Development Goals
SGD	Stochastic Gradient Decent
SysFor	A Systematically Developed Forest of Multiple Decision Trees
LargeVis	Algorithm for Visualizing Large-scale and High-Dimensional Data
ELM	Extreme Learning Machine
MSE	Mean Squared Error
CCA	Complete Case Analysis

ACA	Available Case Analysis
GC	Green Cover
SI	Synthetic Imagery
DA	Data Augmentation
RBF-SVM	Radial Basis Function – Support Vector Machine

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 Background

Almost 71% of the surface of the Earth is covered with water. However, less than even a percent of it is fit for consumption. The Earth is home to nearly 6.5 million terrestrial organisms and 2.2 million aquatic organisms. With such less amount of portable water, water scarcity is an alarming issue which has garnered the interest of the scientific community for the last three decades or so. In Texas alone, as of 2010, the water consumption by the agricultural sector accounted to 58% of the total freshwater usage but it has accounted for only 0.6% of the state's economy. Due to this, more economically viable ways for water conservation must be practiced; otherwise, it could lead to a drought like situation in the next half a century.

Chang, *et. al.* [1] recommended to market the usage of water on theoretical grounds, but for any implementation of these policies, it is required to have a proper administrative control and structure to be in place. Talking of Texas alone, this state faces an important issue in harnessing market forces to address the problem of scarcity and depletion of groundwater.

Agam, *et. al.* [2] suggested that it is very important to have an estimation of the water budget of any particular area for planning purposes. One of the main variables in the management of water resources in irrigation applications is the evaporation loss, often referred to as evapotranspiration. Other physical characteristics are also important including water inflow into land masses. Specific management techniques can be practiced so as to decrease these kinds of losses, to reduce the impact on water resources. However, there are still other forms of water losses which are unavoidable such as soil-related permeation. Certain methods such as Deficit Irrigation

(DI) [3] have been proposed for improving the economic viability of agricultural systems, but these kinds of practices impose a lot of adjustments in the entire system. This forces us to think of alternative data modelling for water resources management to optimize the water needed for irrigation with minimal wastage of water (minimal runoff). In summary, irrigation management should rely on data mining and mathematical modeling to prescribe optimal irrigation practices.

1.2 Previously implemented approach for runoff minimization

The previously implemented approach for runoff minimization was conceived by Udaya Bhaskar Kothapalli [4] as part of a project funded by the Water Seed Grant Program at Texas A&M University. The designed controller relied on HTTP GET method to store all the irrigation activity on to a server named irrigation.db. The system also consisted of a continuous monitoring and storing of the weather data approach for every 15 minutes of operation. A Python Script was implemented on the server, which recorded the current and future weather data from the Open Weather API. Adding to this, he also used irrigation activity-based features, which were taken into account in the analysis of the data.

A total of 36 important features were determined due the process of data analysis and were used for training the Machine Learning (ML) classifiers. The set of predictors used in the analysis have been shown in the tables 1.1, 1.2, 1.3 and 1.4.

Table 1.1: Past weather parameters that were used as features in this approach by Udaya Bhaskar Kothapalli. Reprinted from [4]

PREDICTOR NAME	DESCRIPTION & UNITS
PAvg_Humidity	Average humidity for the last 3 days (%)
PAvg_Pressure	Average pressure for the last 3 days (hPa)
PAvg_Temp_min	Average minimum temperature for the last 3 days (K)
PAvg_Temp_max	Average maximum temperature for the last 3 days (K)
PAvg_Wind_speed	Average wind speed for the last 3 days (m/s)
PAvg_Clouds	Average cloudiness for the last 3 days (%)
P_Rain	Rain volume for the last 3 days (inches)
P_Snow	Snow volume for the last 3 days (inches)
Numday_last_rain	The number of days since the last rain event was recorded
Numday_last_irrigated	The number of days since it was last irrigated

Table 1.2: Current weather parameters that were used as features in this approach by Udaya Bhaskar Kothapalli. Reprinted from [4]

PREDICTOR NAME	DESCRIPTION & UNITS
C_Temp	Current temperature (K)
C_Humidity	Current humidity (%)
C_Pressure	Current atmospheric pressure (hPa)
C_Temp_min	Min temperature last recorded by weather station (K)
C_Temp_max	Max temperature last recorded by the weather station (K)
C_Wind_speed	Current wind speed (meter/sec)
C_Wind_deg	Current wind direction (degrees)
C_Clouds	Current cloudiness (%)
C_Rain	Rain volume recorded in the last 3 hours (inches)
C_Snow	Snow volume recorded in the last 3 hours (inches)

Table 1.3: Forecasted weather parameters that were used as features in this approach by Udaya Bhaskar Kothapalli. Reprinted from [4]

PREDICTOR NAME	DESCRIPTION & UNITS
F_Humidity	Average humidity forecast for the next day (%)
F_Pressure	Average pressure forecast for the next day (hPa)
F_Temp_min	Minimum temperature forecast for the next day (K)
F_Temp_max	Maximum temperature forecast for the next day (K)
F_Wind_Speed	Average wind speed forecast for the next day (m/s)
F_Clouds	Average cloudiness forecast for the next day (%)
F_Rain	Rain forecast for the next day (%)
F_Snow	Snow forecast for the next day (%)

Table 1.4: Irrigation parameters that were used as features in this approach by Udaya Bhaskar Kothapalli. Reprinted from [4]

PREDICTOR NAME	DESCRIPTION & UNITS
Effective_Irrigation_Time	The time for which the last irrigation cycle was run (min)
Number_Runoffs	The number of runoffs before irrigation is completed
Total_runoff_time	Total runoff time observed in the irrigation cycle (min)
Time_to_First_Runoff	Irrigation time until the first instance of runoff was observed (min)
Time_to_Second_Runoff	Irrigation time until the second instance of runoff was observed (min)
First_Runoff_Interval	The time interval between the start and end of first runoff (min)
Second_Runoff_Interval	The time interval between the start and end of second runoff (min)
Soil_Moisture	The pre soil moisture content of the soil (%)

Around 400 synthesized datapoints were generated from the limited previous irrigation activity recorded in 2015 and 2016 field tests at the TAMU Turfgrass Lab on plots 15, 17 and 18. Multinomial Logistic Regression, Multilayer Perceptron and SVM classification schemes were used on these synthesized feature vectors. These Machine Learning algorithms output a certain irrigation code and a set of irrigation rules were devised by the author in consultation with Dr.Wherley, a faculty in Soil and Crop Sciences, Texas A & M University for each output irrigation code throughout the entire year and have been included in the Tables 1.5 and 1.6.

Table 1.5: Prescribed irrigation rules for the Spring and Fall Seasons used in this approach by Udaya Bhaskar Kothapalli. Reprinted from [4]

Irrigation Settings Code	Start Time	Pause Time	Maximum Operational Time	Effective Irrigation Time	Suitable for lawn/weather condition
M0	0	0	0	0	No Irrigation Needed
M1	12 AM	3 hours	6 hours	15 min	Too Wet
M2	12 AM	3 hours	10 hours	20 min	Wet
M3	7 PM	3 hours	12 hours	20 min	Wet
M4	12 AM	2 hours	8 hours	25 min	Moderate
M5	10 PM	3 hours	8 hours	25 min	Moderate
M6	12 AM	2 hours	8 hours	30 min	Dry
M7	12 AM	2 hours	6 hours	30 min	Dry

Table 1.6: Prescribed irrigation rules for the Summer season used in this approach by Udaya Bhaskar Kothapalli. Reprinted from [4]

Irrigation Settings Code	Start Time	Pause Time	Maximum Operational Time	Effective Irrigation Time	Suitable for lawn/weather condition
M0	0	0	0	0	No irrigation needed
M1	7 PM	3 hours	12 hours	30 min	Wet
M2	12 AM	3 hours	10 hours	30 min	Wet
M3	10 PM	3 hours	8 hours	35 min	Moderate
M4	12 AM	2 hours	8 hours	35 min	Moderate
M5	12 AM	2 hours	6 hours	35 min	Dry
M6	12 AM	2 hours	8 hours	40 min	Too Dry

Before training the classifiers, the entire dataset was normalized, and 75 percent of the data was randomly used for training and 25 percent of the data was used for testing and validation of the classifiers. The MultiLayer Perceptron used in this case consist of three layers: the input layer, the hidden layer and the output layer. The specifications of the MultiLayer Perceptron used in this case are as follows:

1. Activation function : Logistic sigmoid function

2. Learning strategy : Stochastic Gradient Descent (SGD)
3. Learning rate : Adaptive learning rate
4. Number of epochs : 200

The classification accuracies for all the three classifiers were recorded and it was concluded that MultiLayer Perceptron gave the highest accuracy on the testing data (94.5%) followed by SVM Classification and Multinomial Logistic Regression with an accuracy of 85% and 83% respectively.

1.3 Motivation for current work

As mentioned before, it is very crucial to monitor and manage the regional budgeting of water in specific areas including turfgrass lawns. In the management of turfgrasses, water loss due to evaporation is one of the most important factors which should be taken into account when prescribing controlled irrigation events. Other factors such as insolation forcing, Vapor Pressure Deficits and regional dry air advection should be considered as well. To fully account for water loss due to natural mechanism including evaporation, many regional-scale Evapotranspiration (ET) models [2] have been proposed, which use remote-sensing data in order to gauge the soil temperature and surface soil moisture. Another approach that has been proposed is the generation of monthly or bi-monthly sub-field-space spatial resolution maps which are acquired with the help of Thematic Mapper (TM) [2].

Irrigated agriculture consumes about 70-80% of the water where the aridity of the soil is high. Therefore, the concept of Deficit Irrigation (DI) [3] has been proposed for irrigation under limited water conditions.

Apart from these abovementioned approaches, there have been data-driven models which take climatic predictors as inputs and predicts the amount of water required for irrigation. Capraro, *et. al.* [5] proposed a prediction model i.e. a Neural Network within the control algorithm to take the soil moisture level to the desired level. Karasekreter, *et. al.* [6] proposed a similar Artificial Neural Network (ANN) which takes soil moisture, soil type, product type and time intervals as inputs. This model determines the amount of water required for irrigation and the irrigation time intervals for strawberry orchards in Turkey with improved efficiency in the range of 24%. Khan, *et al.* [7] proposed various data mining techniques for predicting the irrigation water needs for different types of crops using suitable prediction algorithm. These models and previous work helped guide and motivate the existing work for developing similar models for turfgrass irrigation, taking into consideration climatic and field-specific predictors.

CHAPTER II

LITERATURE REVIEW

2.1 Addressing global freshwater shortage

Veera Gnaneswar Gude [10] recognized the act of ensuring freshwater as the most basic need for the survival of humanity. Over the last few decades, overpopulation coupled with rapid industrialization and increased living standards has resulted in an unprecedented demand for freshwater all over the world. More than 30% of the people in the world lack access to clean water sources for basic sanitation needs and it is projected that by 2060, more than 60-70% of the global population would go on to live in regions of absolute water scarcity. To address this issue, two main approaches related to management and technology development have been proposed: demand mitigation and supply enhancement. Demand mitigation refers to a set of practices where one tends to enforce a more responsible behavior by the users through economic pricing of natural water available. Supply enhancement is a set of water conservation practices which exist like rainwater harvesting.

Stuart E Bunn [11] recognized the need for massive investments to be made in the present world to offset the threat to human water scarcity. However, they have come at a considerable cost to the aquatic biodiversity and no attempts have been made to offset this damage. For this reason, certain global initiatives have been taken up in the last half of the century. The Millenium Development Goals (MDGs) were successful in reducing the biodiversity loss primarily with terrestrial and aquatic ecosystems. Yet, the omission of freshwater systems was a significant oversight owing to the fact that 40% of the world's vertebrae species stay in these ecosystems. Another such initiative was the setting up of United Nations Watercourses Convention (UNWC) and the United Nations Economic Commission for Europe (UNECE) Water Convention, which

played an important role in strengthening international laws on freshwater usage. Some of the initiatives and actions have also been taken up at the local and regional level. Most of the Australian cities have significantly reduced per capita water usage through a large number of initiatives pertaining to demand management while increasing the gross value of production. The Rio+20 commitment to Kyoto-compliant energy resources has resulted in building nearly about 3500 dams for effective fragmenting of the planet's free flowing rivers.

Another important point which has been addressed in this paper under the Water Sustainable Developmental Goals (SDGs) by the United Nations for the year 2030, is the problem of point and non-point pollution. There has been a billion-dollar investment to tackle the water pollution resulting from urban and industrial waste. Tackling non-point pollution remains a bigger challenge throughout the world. Riparian management guidelines have been prescribed for the rehabilitation of these streams and rivers, but they seem to be ineffective as they have failed to consider the social and economic aspect of the problem.

2.2 Detrimental effects of agricultural runoff

Willis, *et. al.* [12] reviewed how the quality of water in freshwater bodies is inversely affected due to the presence of pesticides in runoff. This paper iterated the fact that the use of pesticides has increased about 50-fold in the last few decades. In the United States alone, farmers use about 661 million pounds of pesticides and the United States amounts to about 65% of the total usage. The use of herbicides is on the rise; however, use patterns of insecticides have shifted from organochlorine to organophosphate and carbonate compounds. Sustainable use of pesticides is a prime concern and it requires knowledge of how they are transported, partitioned, detoxified and accumulated in the environment.

There are a few factors which affect the concentration of pesticides in runoff, a few of them being:

1. **Rainfall characteristics**: The duration and the frequency of rainfall are instrumental in determining the concentration of pesticides observed in agricultural runoff. The relationship between them is strictly directly proportional to one another.
2. **The time interval between application of pesticides and rainfall:** It is instrumental in determining the amount of runoff which is detected in agricultural fields. It is observed that the concentration of pesticides in runoff is higher when the application time between them is less.
3. **Properties of pesticides**: The chemical and physical composition, formulation and persistence of pesticides is consequential in determining the concentration of pesticides in runoff. The pesticides which are insoluble or partially soluble in water generally contribute to the long-term potential loss by runoff since the half-life of these compounds (time required for 50% of these pesticides to disappear) is very high.
4. **The rate of application of pesticides and the amount of pesticide loss in runoff:** Both of them are positively correlated, but the relationship between them is not strictly linear and this effect goes on to dwindle in due passage of time.
5. **Texture of soil and topography**: The steepness of the slope and the texture of the soil are instrumental in determining the amount of pesticides in runoff. Higher runoff generally tend to occur in places which have fine-textured soils rather than places, which have coarse-textured soils, since the infiltration rate of the former is low. The steepness of the slope also does play a role in the amount of runoff which the soil may incur. The higher the steepness, greater would be the runoff and vice-versa.

6. **Soil moisture level/content**: If the soil moisture content is high i.e. if the soil is wet/moist, then the soil infiltration rate is extremely low and there is a high chance of runoff, thereby increasing the pesticide concentration in the effluent stream.
7. **Ground cover type and quantity**: Ground cover, including crop residues, reduces the amount of soil erosion, thereby decreasing the transportation of pesticides in runoff. These ground covers aid in avoiding the transportation of water insoluble pesticides.
8. **Transport distance**: The greater the distance of transport of runoff across an untreated land, the lower the concentration of pesticides present in runoff. These lands absorb most of these pesticides, thereby reducing the pesticide concentration in runoff.

Out of all these factors, the major factors contributing to the concentration of pesticides in runoff are the first three factors [12].

2.3 Urban runoff mitigation techniques

Xie, *et. al.* [13] identified an appropriate drainage solution to minimize the potential flooding risk in urban areas for long rainfall duration, such as a combination of Rain Barrel, Previous Concrete and Green Roof. The experiment was carried out in a developed area in Shanghai under different considerations and it was concluded that increasing the pipe size could result in improved node flooding if the rainfall is for a shorter duration.

Besides extremely heavy rainfall events, the problem of impervious surfaces need to be addressed, which is one of the main hydraulic changes in the catchment processes due to urbanization. These impervious surfaces decrease the infiltration of runoff in urban areas, and indirectly affect downstream flooding. These were traditionally used as a means of controlling

runoff, but due to the cost involved in it, Green Infrastructure practices, have been considered as a better alternative.

As an alternative to the existing traditional gray infrastructure, Green Infrastructure techniques like Green Roof, Bio-Retention Cell, Vegetative Swale and Permeable Swale are being considered. These techniques alone do not help in the reduction of urban runoff. There have been certain Low Impact Development (LID) practices implemented before at an in-situ level, but they do not tackle the issue of surface runoff, which is one of the main causes of contamination of the urban water bodies. So, the main focus of future research is to formulate a framework to combine LID techniques and Green Infrastructure practices judiciously.

2.4 Smart irrigation strategies and initiatives

Gaborit, *et. al.* [14] identified stormwater detention ponds as one of the Best Management Practices (BMPs) for limiting runoff in urban landscape. These kinds of ponds store water during the rainy season, limiting the velocity of water and also improves its quality. The concept of dry retention ponds are widely practiced in the United States of America and Canada. The paper touches on finding techniques like SWMM5, which is a prediction-based approach for managing the routing of flow in the system. This designed model is helpful for simulation of runoff and the suspended solids in it over the catchment area. Although the simulation assumes ideal conditions, which results in an overestimation in the calculation of efficiency of the model, the overall model holds well for the hydrologic-hydraulic simulation process.

Tang, *et. al.* [15] studied the flow of water on and inside the surface of soil where the soil below has impenetrable rocks and tile drains. This work stresses on the evaluation of various mitigation strategies for reducing runoff, such as, constructing wetlands, ditches and long stripes

of vegetation. The vegetated strips/buffer strips used help in the reduction of pesticides from runoff in the following manner: (1) By facilitating the sorption of particles which reduce the overland runoff that contain the dissolved pesticides and (2) Enhancing pesticide retention by increasing the infiltration time. The performance of vegetative strips for filtering out pesticides and its effect on the environment has not been evaluated and still remains a point of contention among most of the researchers. A significant amount of research needs to be dedicated to investigating the effect of trapped pesticides through different physiological processes in these buffer zones, and the preventive measures that can be taken to avoid their release. This paper also discusses a few measures that need to be taken while applying pesticides like considering the nozzle diameter, the climate and soil restrictions and the variability in application distance. The application of pesticides must be reduced in high-risk areas because the differences in the composition of the pesticides play a more important role than the variability of field-specific characteristics. There is another Good Agricultural Practice (GAP), which needs to be practiced such as tillage practice. This process alters the soil hydraulic properties, thereby affecting the water flow pathways in the soil. The third most commonly used GAP is the process of Tile Drainage which is used in places where shallow groundwater table exists. This process reduces the runoff over the land by 38% and significantly reduces the pesticide losses by about 56%.

Davis, *et. al.* [16] prescribed Bioretention, Bio-infiltration and Rain Gardens as one of the most innovative and efficient state-of-the-art techniques for managing storm water in urban areas. These techniques are extremely important for managing other pollutants like pathogenic bacteria and thermal pollution, but many design questions also persist for this practice such as the depth of the pooling area and the fill media used, the options used before treatment is applied and the selection of vegetative buffers.

The general features of the prescribed bioretention system include two major steps:

- 0.7 m to 1 m of soil media to help in the infiltration of runoff
- Using different types of vegetative strips to have a few inches of runoff pooling

As this entire process is still evolving and there is enough evidence to substantiate its efficiency in reducing urban runoff, there are certain fields of resear/ch which need serious attention. A few of them have been listed below.

- The fill media depth and composition
- The configuration and basin geometry of drainage
- Vegetation selection

Total Suspended Solids (TSS) removal and Total Nitrogen (TN) removal using this abovementioned technique in select field and laboratories have been studied in detail at sites in College Park, Charlotte, Durham, Villanova and few other places and have been summarized in this paper.

Vymazal, *et. al.* [17] proposed the use of Constructed Wetlands (CWs) for controlling the concentration of pesticides in runoff in most developing nations. So far, there has been wide usage of Constructed Wetlands with open and free water surfaces. Current work is being carried out on subsurface flow Constructed Wetlands (CWs) which have proven to be quite instrumental in controlling the concentration of pesticides in runoff as there have been a large number of physiological and biological processes involved in these CWs. However, there is strong evidence to suggest that the vegetative strips used in these wetlands play the most significant role. In summary, many irrigation controls studies have considered passive techniques to minimize runoff and pesticide contamination into water streams.

2.5 Current prediction-based models for controlling irrigational runoff

Caparo, *et.al*. [18] prescribed a controller, which monitors the soil moisture around the desired level. A Recurrent Neural Network (RNN) is used, which takes the plant's in-roots soil moisture level as the input and its predicted future values. The use of Recurrent Neural Network in a control system has been proposed before, but this paper showcased its application to agricultural systems. These models which are developed from the existing irrigation programming models consider the predictors mentioned in Table 2.1.

Table 2.1: List of predictors used by Caparo, *et. al.* to train the RNN. Reprinted from [18]

1. Environmental variables	Temperature Pressure Solar radiation Wind speed and direction
2. Plant variables	Stem size Sap flow
3. Soil variables	Soil temperature Humidity Conductivity

The training process was carried out using the experimental data about the soil moisture content of the soil and the time period for which the irrigation cycles were run. As stated before, the Neural Network used in this case is a Recurrent Neural Network. The specifications of the RNN used in this case are as mentioned in Table 2.2.

Table 2.2: Specifications of the RNN used by Caparo, *et. al.* Reprinted from [18]

1. Input layer	10 inputs and 8 delays
2. Hidden layer	20 neurons with tanh(.) activation function
3. Output layer	1 neuron with linear activation function

Then, the output from the Recurrent Neural Network is normalized before feeding it to the input which is achieved by dividing the output by the maximum volumetric level.

Karasekreter, *et. al.* [6] developed techniques to predict Irrigation Ratios and Time Intervals using an Artificial Neural Network (ANN) where the Levenberg-Marquardt (LM) and the BackPropagation (BP) algorithms were used to train our Neural Network and achieved successful results in the classification process. The input parameters and specifications of the Distributed Time Delay Neural Network (DTDNN) are shown in Table 2.3.

Table 2.3: Specifications of the ANN used by Karasekreter, *et. al.* Reprinted from [6]

1. Input parameters	Soil moisture Soil type Product type Time interval
2. Input layer	4 Neurons
3. Intermediate layer	10 Neurons
4. Output Layer	1 Neuron

This design is primarily aimed at conserving water and promoting the concept of irrigation during nighttime as it helps in reducing the water losses resulting from evaporation. This test was performed at a strawberry orchard spanning over 0.24 acres of land in Turkey and resulted in 20.5% savings in water and 23.5% savings in energy.

Khan, *et. al.* [7] prescribed the irrigation water requirement for different types of crops with the aim to reduce wastage of water in irrigated agriculture. The input dataset consisted of T-max, T-min, humidity, wind speed, rainfall, solar radiation, soil type and crop type; and the crop water usage was generated as the output. An approach named Reference Evapotranspiration Based Estimate was proposed to carry out the preprocessing of the data and a set of different Machine Learning algorithms were applied both on the processed and unprocessed dataset and their accuracy was observed in both the cases. A 3-fold cross-validation procedure was used to validate the model and it showed that there was a minor difference in the prediction accuracy for major Data Mining techniques such as SysFor, Decision Tree and ANN. This resulted in obtaining a prediction accuracy of 78% using SysFor, followed by Decision Tree and SVM with 74% and 64% prediction accuracy respectively.

Umair, *et. al.* [8] collected the environmental data recorded by sensors and were passed to the next stage which converted these parameters to actual soil moisture. Then, this calculated soil moisture was compared with the required soil moisture with the help of an Artificial Neural Network (ANN) and a decision was made dynamically on how much water would be required to irrigate the plots. The topology of the ANN used in this case is mentioned in Table 2.4.

Table 2.4: Specifications of the ANN used by Umair, *et. al.* Reprinted from [8]

1. Type of ANN	Distributed Time Delay Neural Network
2. Training function used	Bayesian Regulation Function
3. Performance measure used	Sum Squared Error
4. Learning rate	0.5

2.6 Missing data imputation techniques for completeness of the historical ASIS dataset

For developing an efficient Machine Learning (ML) algorithm using the ASIS historical dataset for a smart controller, the ML predictors should not have any missing values. Therefore, an applicable and robust ML learning algorithm capable of dealing with missing values within the data set should be used.

Schafer, *et. al*. [19] proposed techniques to impute or assign multiple missing values in datasets collected from the Adolescent Alcohol Prevention Trial. The questionnaire had several questions, which the subjects were either unwilling to respond due to lack of time or interest. The questionnaire consisted of three parts, where every one-third of the correspondents received any two of the three sections randomly. Previous approaches included estimation of mean for the datapoints as it is one of the simplest and most straightforward means to impute missing values. However, this method increased the correlation between the predictors thereby interfering with the analysis. That is why, in this case, the author proposed Multiple Imputation techniques where Bayesian techniques were used to construct a predictive distribution for the features and missing data was imputed with multiple values derived from this distribution. These results were combined to get overall estimates for the missing-data uncertainty. It was observed that, at most, only 3-5 imputations were required to get the desired results by this approach. The efficiency of an estimate based on m imputations has been summed up as: $\left(1+\frac{\gamma}{m}\right)^{-1}$ where γ is the fraction of missing information for the quantity being estimated.

Therese D. Pigott [20] reviewed many methods for handling missing data while conducting a questionnaire-based survey for students exhibiting asthma symptoms. Four different reasons for missing data were figured out in the survey: absence from school during the time the survey was conducted, symptom severity data missing, response mechanism missing and the last being

students forgetting to fill up all the data fields in the survey. The scale and the distribution of all the variables used in the study are considered to be multivariate normal and then, seven commonly used missing data methods were implemented to impute the missing values. Using Complete-Case Analysis (CCA), all the observations that have even one missing value in the analysis were dropped and Linear Regression is used to impute all the missing values. The main disadvantage of this method is that in 90% of the cases, there was at least one missing predictor value; thereby, reducing the size of the training dataset to only 15 observations which is too low to train any model. The second approach used in their case is the Available Case Analysis (ACA), which does not take validation data, training data and testing data separately. The drawback of this approach is that it works only for the variables that are weakly correlated with the response. For the observations where the variables are either strongly or moderately correlated, this entirely could ruin the analysis. The third approach used in this case is based on the averaging values of the predictor with available data to fill up the missing values. This approach leads to underestimate the variance of these variables and under no condition, would this approach produce unbiased results. The fourth and fifth approaches focus on using model-based methods, namely imputation using EM algorithm and Multiple Imputation techniques. The EM algorithm focuses on finding the statistical estimates of a distribution like the mean and covariance matrix when there is no defined solution to maximize the likelihood. Using Multiple Imputation, the researcher aims to impute the missing data with possible values for each observation.

2.7 Data clustering techniques for visualization of ASIS dataset to determine target variables' thresholds

Machine Learning approaches involve data clustering techniques to be able to bin observations into different categories to enable suitable decision-making. For that purpose, Tang, *et. al*. [21] identified a technique for finding representation of huge dimensional data into lower spatial dimensions. The basic idea is to retain the inherent structure of the datapoints while projecting them in lower dimensions. In this paper, the author proposed a technique called LargeVis algorithm, which works by constructing a graph based on the KNN algorithm before projecting it out in lesser dimensions. This KNN Graph works well over all the other algorithms since it uses a principled probabilistic model with a synchronous gradient descent algorithm for optimization. The pipeline for the data visualization process as explained by LargeVis is shown in Figure 2.1.

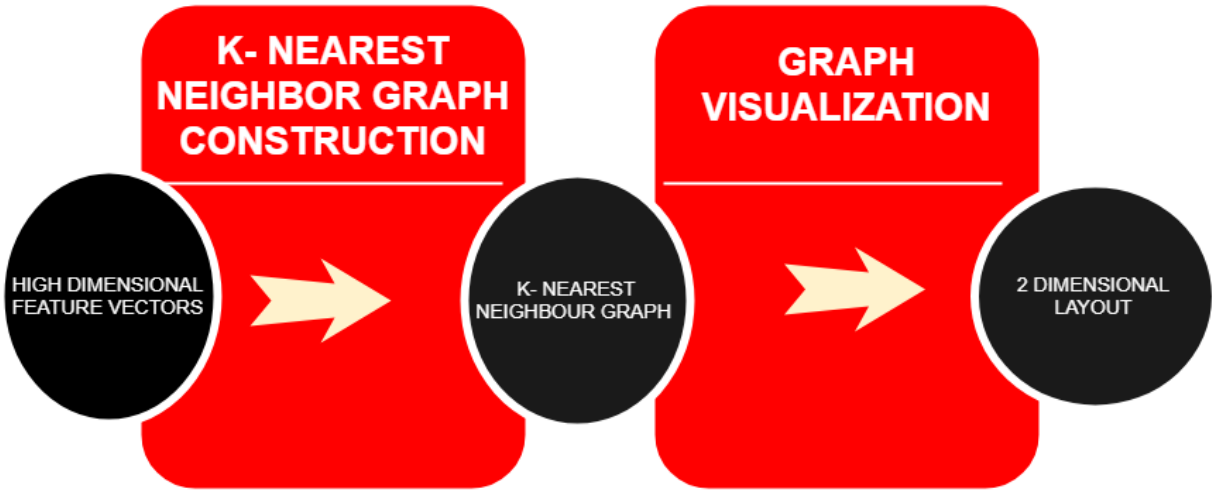

Figure 2.1: Pipeline of data visualization technique proposed by Tang, *et. al.* Reprinted from [21]

From the pipeline, it is clear that LargeVis algorithm constructs a K-Nearest Neighbor graph of all the datapoints and projects the same onto a lower-dimensional space. This algorithm is very efficient for constructing a principled probabilistic model for graph visualization, the objective of which can be optimized effectively. Most of the future work would be planned on building on how to use this algorithm for low-dimensional layouts, how to generate more intuitive and meaningful visualizations for high-dimensional data and how to handle data dynamically over time.

Pickett, *et. al.* [22] proposed the concept of graphic icons for visualizing high-dimensional data where these datapoints in multiple dimensions were reduced to two-dimensions, which were then structured in the form of gradients and contours for analysis. This work is one of the earliest works in the field of dimensionality reduction when working with image data, which were recorded by earth satellites and MRIs.

Daniel A Keim [23] proposed a classification technique on how to deal with one-dimensional data, two-dimensional data, multi-dimensional data, text and hypertext. All the techniques used are explained in detail, as follows:

(a) **Geometrically Transformed Displays**: These include techniques from exploratory statistics such as scatterplot matrices. There are certain other projection techniques, including Prosection Views, Hyperslice and the Parallel Coordinates visualization techniques. The parallel coordinates visualization technique maps the k-dimensional space into two dimensions.

(b) **Iconic Displays**: These map the attribute values of multidimensional data to the features of an icon.

(c) **Dense Pixel Displays**: These kinds of displays use different permutations and combinations of the pixels in an image to obtain correlation data among them, which is an instrumental step in feature reduction.

(d) **Stacked Displays**: The basic principle of this display is the juxtaposing of one coordinate system over the other and so on. Apart from the visualization techniques, some interaction and distortion techniques for effective data exploration have also been introduced in Keim [23]. The interaction techniques allow the analyst to interact with the visualizations and dynamically change them according to the exploration objectives. The distortion techniques help in retaining the structure of the data, but it also goes on to provide means of focusing on more details. The basic idea is to differentiate parts, which need to be showed with high amount of detail from the parts, which are showed with low amount of detail.

Buja, *et. al.* [24] implemented data visualization techniques with Multidimensional Scaling (MDS). This algorithm works by constructing maps in the sample space by interpreting the dissimilarities as distances. This approach has found recent application in the Machine Learning (ML) domain motivated by large databases. This approach of dimensionality reduction and visualization went on to become more popular with the emergence of kernelizing approaches inspired by Support Vector Machines (SVMs).

Ed H Chi [25] proposed an extension to taxonomize information visualization techniques by using the Data Space Model [Chi98]. This paper shows that this model not only helps researchers understand the space of design, but it also helps implementers understand how different data visualization techniques can be applied more broadly. The structure of the Data Space Model is shown in Figure 2.2.

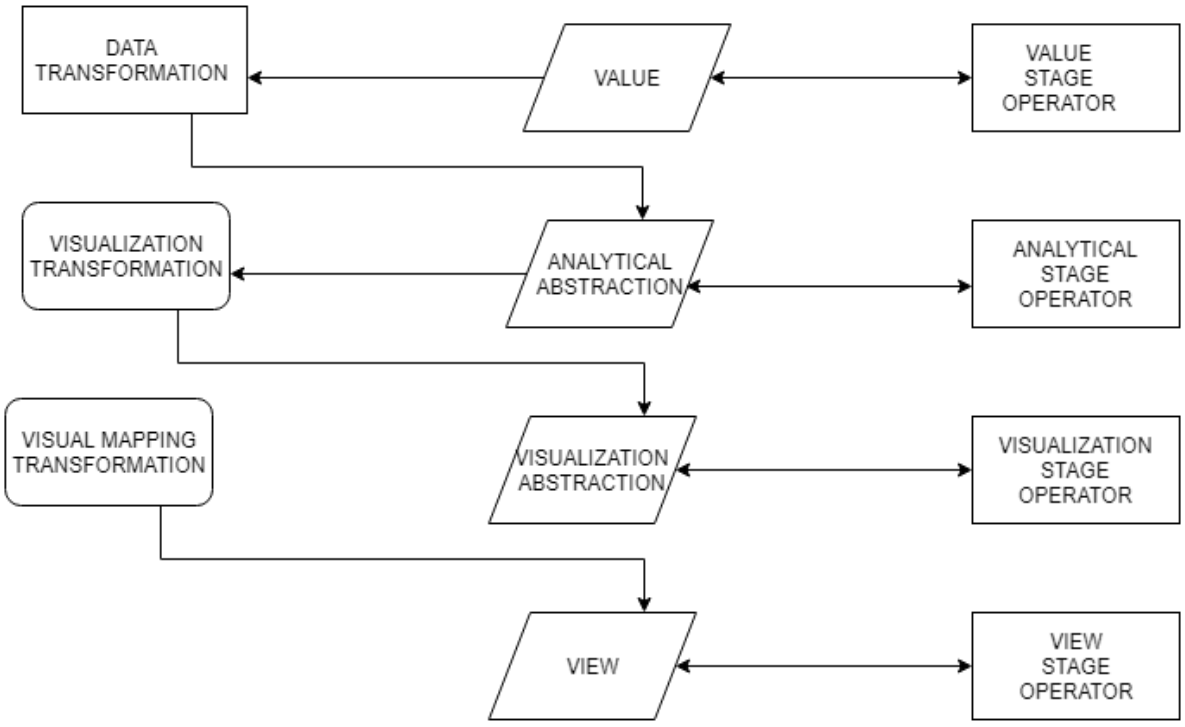

Figure 2.2: Proposed Data Space Model by Ed H Chi. Reprinted from [25]

2.8 Feature extraction techniques for better prediction of target variables in ASIS dataset

In applications where the number of observations is limited but have a large number of features, it can be challenging drawing inferences to postulate an accurate fitting model. Therefore, many feature selection techniques have been considered over the years.

In recent years, Lei Yu, *et. al.* [26] introduced a novel concept, which takes into account a predominant correlation, and a fast filter method to identify relevant features in the data without any pairwise correlation analysis. Using Symmetrical Uncertainty as the goodness measure, the technique was used to decide whether a feature was relevant to the class or not using an user-defined SU value. The SU value determines whether a relevant feature is redundant or not using pair-wise correlations between all features.

Jaworska, *et. al.* [27] stated how Multi-Dimensional Scaling (MDS) can be used as a tool in various psychological domains for exploratory data analysis. This works by condensing large amount of data into relatively simpler spatial maps that convey important relationships in the most economical manner. This tool can be used to model nonlinear relationships among variables, and it does not require multivariate normality. This insight has proven to be highly effective in reducing multi-dimensional outputs from MDS, and can be regressed with more objective variables, thereby providing more confidence in the emerging scaling solution and its interpretation.

Park, *et. al.* [28] suggested the feature extraction scheme, which uses class information to extract features by the Principal Component Analysis. The algorithm was tested on Yale face database and its performance was compared with the other algorithms. This algorithm consists of mainly 3 steps:

- Addition of labels to the datapoints
- Principal Component Analysis for feature extraction
- Determination of transformation matrix W

The pipeline of the PCA-FX algorithm is shown in figure 2.3. This algorithm achieved the lowest classification error (6.06%) on the testing dataset and is better than all other algorithms even though it uses the least number of features.

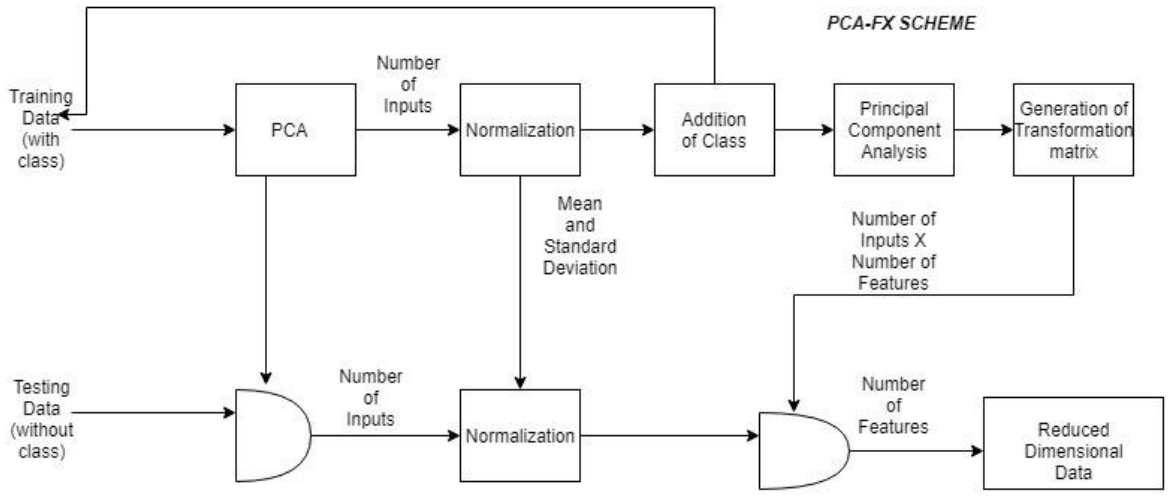

Figure 2.3: PCA-FX pipeline prescribed by Park, *et. al.* Reprinted from [28]

2.9 Data augmentation techniques using synthetic data for enabling Machine Learning approaches in irrigation control

In certain applications, lack of adequate data quantity can make any Machine Learning approach almost impossible to implement. Therefore, synthetic data generation is necessary for developing a robust ML approach. Dyk, *et. al.* [29] introduced an effective technique for augmenting data, which uses both marginal and conditional augmentation strategies with approximation methods. This strategy was applied to three common classes of models namely multivariate t, probit regression and mixed-effect models to obtain efficient Markov Chain Monte Carlo (MCMC) algorithms for posterior sampling. The basic feature of all these algorithms is that all of these are positive recurrent sub-chains of recurrent Markov Chains (MCs) constructed in larger spaces. These models rely heavily on the way data is distributed and which type of distribution the datapoints follow to ascertain a particular type of data distribution.

Wong, *et. al.* [30] stated that while transforming data from data space to feature-space, generic augmentation techniques achieved far better performance and reduced the chance of overfitting while training the model. The concept of data warping was introduced in this case for training large neural networks. The warped training data was created by applying affine transformation and elastic distortion to the character images (MNIST dataset) to reduce the imbalance in the training dataset. This concept of data warping was compared with another technique called as Synthetic Minority Over-Sampling technique (SMOTE) in which synthetic examples are created in the feature space from randomly selected pairs of real world examples from the minority class. Another alternative data augmentation technique called as Synthetic Imagery (SI) was also introduced in the process as an alternative to both the techniques. After using these data augmentation frameworks, the samples were fed to a convolutional backpropagation-trained neural network, convolutional support vector machine and a convolutional extreme learning machine classifier. While training the convolutional neural network (CNN), augmentation in data space using elastic deformations gave a better improvement in error percentage than augmentation in feature space. Using SMOTE algorithm for augmentation in the feature-space, the test error % decreased when the number of samples was increased to 50000 samples. Similarly, while using the Convolutional SVM, data augmentation in the feature-space using warping techniques provided only a very low improvement in testing error %, but augmentation in data space using SMOTE algorithm did not improve the performance of SVM. Similarly, when the Convolutional ELM was trained, the augmentation in data space did not show a monotonic trend, rather SMOTE showed more promising results on the dataset.

Wei, *et. al.* [31] presented two approximations for the analysis of missing-data problems namely the Poorman's Data Augmentation algorithm and the Asymptotic Data Augmentation

algorithm. Both algorithms were applied in the censored regression case to obtain semiparametric methodologies. The results showed that even when the size of the sample is as small as 40, the algorithms yielded reasonable results in terms of the model bias and the Mean Squared Error (MSE) of the models.

2.10 Advantages and challenges of recent ML studies in the context of irrigation control

The Machine Learning models, which have been discussed here mostly deal with irrigation of crops. Many of them have been used to model taking into account the geospatial characteristics of the area where the observations were recorded. However, runoff minimization has not been addressed or discussed in any of the cited literature.

All the studies cited above consider the distribution of data as part of synthetic data generation process. However, none of the papers have dealt with the estimation of statistical characteristics for synthetic data generation. Moreover, the data visualization and feature extraction approaches, which have been discussed do not take into account the case of scarcity of data in their applications. In summary, all these concerns need to be addressed in future studies.

CHAPTER III

ANALYSIS OF THE ASIS DATASET AND IMPLEMENTATION OF APPROPRIATE MACHINE LEARNING MODEL

3.1 Transformation and construction of field-based dataset for analysis

An irrigation dataset was generated as part of an on-going field study in which irrigation characteristics were measured and recorded. The dataset which has been used in this approach was recorded at the Turfgrass Laboratory, Texas A&M University, College Station. The data was collected using plots 15, 17 and 18 for the years 2015-16 and 2016-17, and similarly, the new data for the year 2018-19 has been recorded from the plots 10,12 and 13. Both the historical and the current data generated have been amalgamated together to be used for analysis and design of an appropriate Machine Learning approach. All the data used for analysis have been generated using CONTROL approaches in which the entire system was programmed to water the plots for a stipulated amount of time, and runoff was observed for each of these plots. All the plots used in the analysis are 13 ft by 27 ft, as shown in Fig. 3.1.

Figure 3.1: Overview of the plot used for analysis in TAMU Turfgrass Lab

The soil moisture, which is one of the most important parameters in this analysis has been measured with a Dynamax TH_2O moisture meter as shown in Fig. 3.2. The pre and the post soil moisture readings have been taken after each irrigation run, and appropriate inferences have been carried out which have been discussed in the later part of this chapter.

Figure 3.2: Dynamax TH_2O moisture meter used for measuring soil moisture content

The runoff volume observed in each of these plots was measured with the help of flowmeters which have been installed in each plot and stores the volume of runoff data recorded every two minutes. The data, which have been recorded were continuous time-series data and was summed over the time of an irrigation cycle. The unit in which this runoff volume was recorded was in

liters and is converted to gallons. The picture of the runoff meter used in the plots is as shown in Fig. 3.3.

Figure 3.3: Teledyne ISCO flow meter and runoff sampler

The irrigation volume observed in this case has been recorded using flow meters, which were installed under the soil at each of these plots. These meters record the volume of water observed in gallons when the irrigation system is run for a stipulated time. The pre and post readings were taken after each irrigation cycle to record the amount of water used in each case. The picture of the flow meter used in this case is as shown in Fig. 3.4.

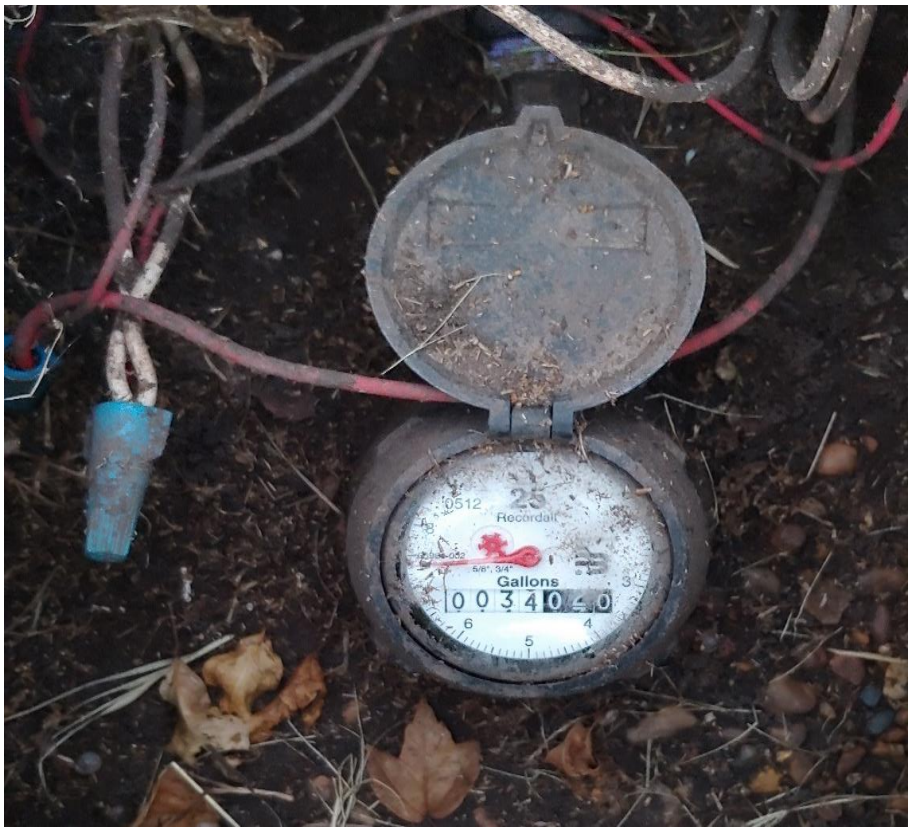

Figure 3.4: Flowmeter used in TAMU Turfgrass Laboratory for measuring applied irrigation volume

3.1.1 Review of the previous dataset

The previous dataset used for analysis for the year 2015-16 and for the year 2016-17 were recorded using both CONTROL and LIRMS approaches. The data recorded by the CONTROL approach has been used to design our model, but the performance of the LIRMS system designed by Udaya Bhaskar Kothapalli [4] was also used to help reduce runoff over these plots. An excerpt from the previous dataset has been included, as shown in Table 3.1.

Trial	Test Date	Scheduled Irrigation Time (mins)	Pause Time (mins)	Start Time (hr:min)	Allowable Irrigation Window (hrs)	Effective Irrigation Time (mins)	LIRMS Total Operation Time (hr:mins)	Irrigation Volume		Runoff Volume		Soil Moisture		Runoff Reduction by LIRMS (%)	Soil Wetting Efficiency Index		Comment
								LIRMS (gal)	CONTROL (gal)	LIRMS (gal)	CONTROL (gal)	LIRMS Pre/ Post(%)	CONTROL Pre/Post(%)		LIRMS	CONTROL	
1	8/13/2016	40min	35min	7:00:00 AM	1Hr	19min 42sec	7:19:42 AM	159	253	14.99	80.93	35.63/43.24	35.11/42.94	70.52	13.43	8.81	Good run
2	8/26/2016	40min	35min	7:00:00 AM	1Hr	15min 24sec	7:15:24 AM	126	254	20.35	75.16	35.23/43.54	35.89/44.15	45.42	18.72	9.06	Good run
3	9/2/2016	40min	30min	7:00:00 AM	1Hr	18min 53sec	7:18:57 AM	135	253	13.6	85.4	32.45/44.32	32.12/45.19	70.16	27.1	16.08	Good run
4	9/9/2016	40min	35min	7:00:00 AM	1Hr	16min 10sec	7:16:10 AM	129	251	25.3	62.74	36.32/42.23	34.54/43.01	21.54	12.61	9.77	Good run
5	9/23/2016	40min	35min	7:00:00 AM	1Hr	25min 17sec	7:25:17 AM	163	253	18.54	49.83	31.32/41.57	31.01/42.15	42.24	20.08	14.2	Good run
6	9/30/2016	40min	35min	7:00:00 AM	1Hr	19min 37sec	7:19:37 AM	146	251	32.7	101.4	33.5/42.4	32.7/43.1	44.6	18.2	12.6	Good run
7	10/7/2016	40min	35min	7:00:00 AM	1Hr	5min 10sec	7:05:10 AM	53	250	101.4	73.8	32.1/46.6	31.6/46.6	-548.6	85.5	18.9	Rain
8	10/14/2016	40min	35min	7:00:00 AM	1Hr	14min 23sec	7:14:25 AM	139	253	25.3	61.7	34.7/45.3	34/45.6	25.4	22.1	13.4	Good run
9	10/21/2016	40min	35min	7:00:00 AM	1Hr	24min 34sec	7:25:43 AM	162	250	22.3	44.6	36.3/43.7	35.5/44.5	22.9	12.7	10.1	Good run
10	10/28/2016	40min	35min	7:00:00 AM	1Hr	15min 12sec	7:15:13 AM	135	251	NAN	NAN	35.3/44.3	35.1/43.8	NAN	18.9	9.8	Flow Data Needed
11	11/4/2016	40min	35min	7:00:00 AM	1Hr	18min 45sec	7:18:45 AM	139	250	NAN	NAN	32.5/43.6	33.6/44.3	NAN	24.4	12.7	Flow Data Needed
12	11/11/2016	40min	35min	7:00:00 AM	1Hr	12min 57sec	7:12:57 AM	115	252	106.2	136.4	38.6/46.4	37.9/45.8	-70.6	17.7	8.3	Rain Or too moist
13	11/18/2016	40min	35min	7:00:00 AM	1Hr	19min 34sec	7:19:34 AM	146	253	32.8	93.3	34.3/43.3	33.9/43.1	39.1	18.0	10.8	Good run
14	11/25/2016	40min	35min	7:00:00 AM	1Hr	15min 16sec	7:15:16 AM	120	251	100.6	199.2	39.2/46.5	39.8/46.3	-5.6	15.5	6.6	Rain Or too moist
15	12/02/2016	40min	35min	7:00:00 AM	1Hr	13min 53sec	7:13:53 AM	119	252	NAN	NAN	37.3/45.9	37.5/45.1	NAN	19.2	8.1	Flow Data Needed
16	12/9/2016	40min	35min	7:00:00 AM	1Hr	19min 28sec	7:19:28 AM	144	251	NAN	NAN	35.5/44.3	34.8/43.9	NAN	17.4	10.5	Flow Data Needed

Table 3.1: An excerpt from the previous dataset recorded by Udaya Bhaskar Kothapalli. Reprinted from [4]

From the above table, the recorded data for the corresponding dates were analyzed. It was inferred from the table that irrigation cycles were carried on a weekly schedule on each of the plots using both approaches (LIRMS and CONTROL). There is a column which contains the information as to how much runoff was reduced using the LIRMS approach. This showed feasible results baring a few datapoints where the soil was either too moist or there was rain on that particular day. It also showed that the previous system did not take into account the weather forecast into consideration while running an irrigation cycle and thus, proved to be one of the major drawbacks of the LIRMS design.

In the dataset mentioned before, there were four datapoints which had missing values for the runoff volume observed for both the CONTROL and LIRMS approaches. However, when this dataset was used, these values were imputed by running a Linear Regression over rest of the predictors which have been picked from the dataset. As the number of observations was too small to begin with, dropping the observations while doing the analysis was not a viable option.

3.1.2 Addition of new features in the dataset

Baring the old features of the previous dataset, while constructing the new dataset for analysis, the weather data from the GRIDMET website was scrapped for each of the corresponding dates. The website provides the remote sensing data for the location used for data collection. The new climatic factors which were included in the dataset for initial analysis are as follows:

(a) Solar radiation [Unit: MJ/m^2day]

(b) Temperature [Unit: $°C$]

(c) Wind speed [Unit: $meter/sec$]

(d) Vapor Pressure Deficit [$Unit: KPa$]

(e) Relative Humidity [No Unit]

(f) Evapotranspiration Rate [Unit: KPa]

Some site-specific predictors were also included in our analysis. The predictors include the following:

(a) Texture of the Soil

(b) Effective Depth of Rooting [Unit: $inch$]

(c) Infiltration Rate of the soil [Unit: $inch/hr$]

(d) Type of turf being irrigated

Besides using the above two categories of predictors, three sprinklers parameters were also used for full data analysis. The sprinkler parameters are as follows:

(a) Application Rate of the sprinkler [Unit: $Gal\,\min^{-1} ft^{-2}$]

(b) Diameter of the nozzle [Unit: $inch$]

(c) Area to be irrigated [Unit: $square\ feet$]

3.1.3 Deciding the value of Evapotranspiration rate to be used in the analysis

Using the abovementioned predictors, mathematical equations were used to calculate some of the parameters used as inputs. The web-scrapped values were compared with the calculated values for validation purposes. The relative error was recorded to have a better idea on the authenticity of the values obtained.

One of the most important predictors in the study is the Evapotranspiration Rate (ET_0) which gives a rough idea about the water requirement of the turfgrasses. As suggested by Dr. Wherley and Dr. Nithya Rajan, the scrapped values of ET_0 were verified with the calculated values.

A set of procedures was carried out systematically to calculate the ET_0 and has been included below.

1. The value of Reference Evapotranspiration (ET_0) was calculated from the Penmann Monteith Equation as shown in Equation 3.1.

$$ET_o = \frac{0.408\Delta(R_n - G) + \gamma\left(\frac{900}{T+273}\right)u_2(e_s - e_a)}{\Delta + \gamma(1 + 0.34u_2)} \qquad (3.1)$$

The description of the parameters used for the calculation of ET_0 are as follows:

$\mathrm{ET_0} = \text{Daily evapotranspiration rate (Unit: mm day}^{-1}\text{)}$

$\mathrm{R_n} = \text{Net radiation at crop surface (Unit: MJ m}^{-2}\text{day}^{-1}\text{)}$

$\mathrm{G} = \text{soil heat flux density (Unit: MJ m}^{-2}\text{day}^{-1}\text{)}$

$\mathrm{T} = \text{air Temperature at 2 m height (Unit: °C)}$

$\mathrm{u_2} = \text{Wind speed at 2 m height (Unit: m s}^{-1}\text{)}$

$\mathrm{e_s} = \text{Vapour pressure of air at saturation (Unit: kPa)}$

$\mathrm{e_a} = \text{actual vapour pressure (Unit: kPa)}$

$\Delta =$ slope of vapour pressure curve (Unit: kPa ° C^{-1})

$\gamma =$ Psychrometric constant (Unit: kPa ° C^{-1})

2. Some of the hyperparameters used in the equation were site-specific and were calculated for the Turfgrass Lab separately.

The value of the psychometric constant (γ) has been calculated as follows:

The height above the sea level for the TAMU Turfgrass Laboratory site (ZIP 77845) was set at 91 m (z = 91m).

The formula for atmospheric pressure (P) was used, as shown in Equation 3.2.

$$\mathrm{P} = (101.3)\left(\frac{293 - 0.0065\mathrm{z}}{293}\right)^{5.26} \qquad (3.2)$$

Considering z = 91 m, the value of atmospheric pressure (P) was estimated to be 100.23 kPa.

Considering this value of pressure, the value of γ was estimated as $(0.665).(10^{-3}).P = 0.067\ kPa\ °C^{-1}$.

Similarly, the value of Δ was calculated from Equation 3.3 which was then substituted in Equation 3.1 to get the value of ET rate.

$$\Delta = \frac{4098\left[0.618\exp\left(\frac{17.27T}{T+273}\right)\right]}{(T+273)^2} \qquad (3.3)$$

where the value of temperature (T) varied every day.

The value of wind speed at 2 m height (u_2) was calculated from the wind speed data recorded at 10 m height (u_z) by using Equation 3.4.

$$u_2 = u_z \times \frac{4.87}{\ln(67.8z-5.42)} \qquad (3.4)$$

where,

$u_2 =$ wind speed at 2m height (Unit: m s^{-1})

$u_z =$ measured wind speed at height z (Unit: m s^{-1})

$z =$ height of wind measurements above ground surface (Unit: m)

After doing the calculations, the results were included in Table 3.2.

Table 3.2: Comparison of calculated and measured ET values with relative error

Test Date	Solar Radiation (MJ/m^2)	Temperature (deg c)	Delta	Wind Speed (m/sec)	Wind Speed (m/sec_2m)	Vapour Pressure Deficit	Relative Humidity	ETO Calculated (mm/day)	ETO Web-scrapped (mm/day)	ETO Relative Error
2/9/16	19.99	27.1	0.21	3.1	2.32	3.57	0.72	10.22	4.91	51.98
3/25/16	20.66	13.9	0.10	2.6	1.94	1.59	0.56	7.08	3.89	45.08
4/1/16	7.09	12.5	0.10	6.8	5.09	1.44	0.71	6.56	2.79	57.44
4/8/16	18.83	19	0.14	3.5	2.62	2.20	0.43	8.49	5.1	39.93
4/15/16	23.13	19.3	0.14	2.8	2.09	2.23	0.77	8.97	4.07	54.63
4/22/16	23.98	19.7	0.14	4.2	3.14	2.29	0.7	10.23	4.94	51.69
4/29/16	21.49	24.9	0.19	7	5.24	3.14	0.73	13.26	5.31	59.97
5/6/16	26.60	21.2	0.16	2.2	1.65	2.51	0.57	9.75	5.45	44.12
5/13/16	23.89	25.4	0.19	3.4	2.54	3.23	0.75	11.13	5.3	52.39
5/20/16	26.50	21.9	0.16	2.1	1.57	2.62	0.78	9.79	4.8	50.97
5/27/16	12.85	19.1	0.14	5.8	4.34	2.20	1	8.84	1.68	81.00
6/3/16	20.04	24.3	0.18	2.8	2.09	3.03	0.79	9.34	4.23	54.70
6/10/16	27.18	28	0.22	3.4	2.54	3.77	0.75	12.62	6.17	51.12
6/17/16	31.05	29.6	0.24	3.5	2.62	4.13	0.68	14.17	7.47	47.28
6/18/16	30.46	28.7	0.23	2.6	1.94	3.92	0.74	12.84	6.74	47.51
6/24/16	29.34	28.6	0.23	4.7	3.52	3.90	0.73	14.58	7.04	51.70
8/13/16	18.02	30	0.25	2.8	2.09	4.23	0.7	9.90	5	49.49
8/26/16	21.90	26.7	0.21	2.4	1.80	3.49	0.78	9.87	4.69	52.49
9/9/16	22.41	28.3	0.23	3.4	2.54	3.83	0.74	11.45	5.61	50.99
9/23/16	16.18	27.7	0.22	3.8	2.84	3.70	0.72	10.13	4.79	52.74
9/30/16	23.39	20.5	0.15	1.9	1.42	2.41	0.57	8.56	4.27	50.12
10/7/16	18.68	25.3	0.19	4.5	3.37	3.21	0.74	10.85	3.99	63.24
10/14/16	14.82	24.2	0.18	2.6	1.94	3.01	0.78	7.79	3.31	57.51
10/21/16	18.61	18.3	0.13	3.3	2.47	2.10	0.66	8.12	3.36	58.63
10/28/16	16.90	22.3	0.17	1.8	1.35	2.68	0.7	7.14	3.67	48.59
11/4/16	6.70	21.6	0.16	3.2	2.39	2.57	0.81	6.03	1.64	72.79
11/11/16	7.01	16.5	0.12	1.6	1.20	1.87	0.62	3.78	1.98	47.56
11/18/16	8.19	20.7	0.15	5.4	4.04	2.43	0.74	8.12	2.12	73.88
11/25/16	5.18	17	0.12	2.6	1.94	1.93	0.81	4.43	1.34	69.73
12/2/16	5.60	13.1	0.10	4.3	3.22	1.50	0.64	5.20	2.19	57.86
12/9/16	10.30	3.6	0.06	3.8	2.84	0.79	0.53	3.86	1.68	56.49

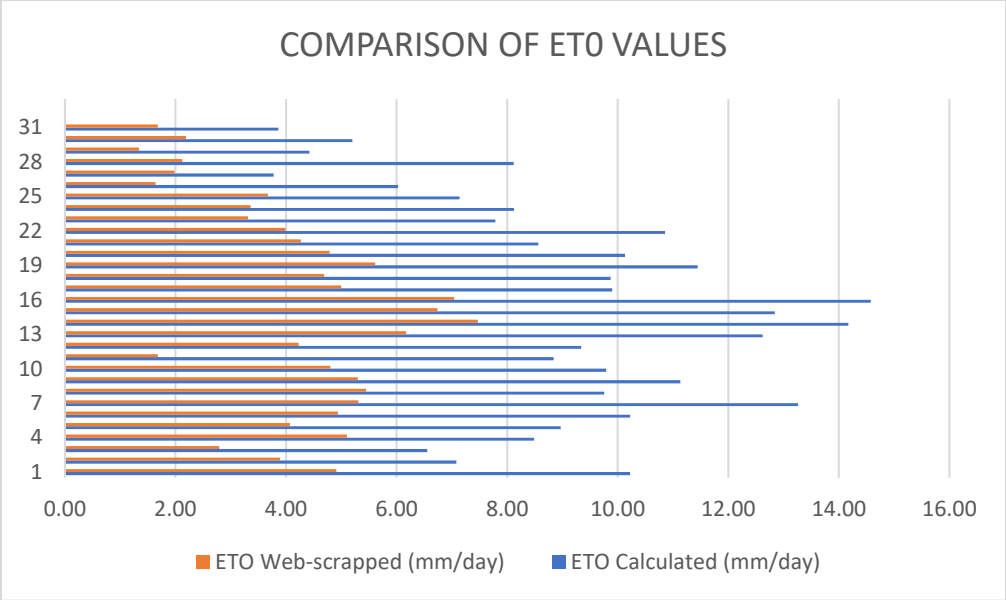

Figure 3.5: Plot comparing the calculated and web-scrapped values of ET0

From Table 3.2 and Figure 3.5, it was clear that while calculating the Evapotranspiration rate using the Penmann-Monteith equation, there was a scaling and normalization error while doing the analysis. A strong correlation coefficient of 90% was observed between the web-scrapped and the calculated values of ET_0. However, as the Penmann-Monteith equation uses an empirical formula to calculate the Evapotranspiration rate, a decision was taken to proceed with the web-scrapped values of ET_0 for future analysis.

3.1.4 Description of the features used

The description of all the predictors used in the dataset while constructing an initial approach using the CONTROL method is as follows:

i. **Test date (Unit: MM/DD/YYYY)**: The date on which the irrigation run was conducted.

ii. **Start time (Unit: hr:min – 24 hour format)**: The time when the irrigation run was scheduled to start.

iii. **Scheduled irrigation time (Unit: min)**: The time for which the irrigation cycle was scheduled to run.

iv. **Solar radiation (*Unit*: MJ/m^2day)**: The radiant energy emitted by the sun from a nuclear fusion reaction that creates electromagnetic energy. This is one of the most important factors in the calculation of Evapotranspiration.

v. **Temperature (Unit: °*C*)**: The degree of hotness or coldness of the atmosphere on the °C scale.

vi. **Daily average wind speed (Unit: miles/hr)**: The mean estimate of the wind speed recorded every two minutes by any weather station over a particular area for an entire day.

vii. **Vapor Pressure Deficit (Unit: KPa)**: The difference between the saturation point moisture content and the actual amount of moisture in the air.

viii. **Area to be irrigated (Unit: sq. ft.)**: The size of the plot to be irrigated in square feet.

ix. **Nozzle diameter of the sprinkler (Unit: mm)**: It is an important parameter in the determination of pressure when the irrigation process is carried out.

x. **Application rate of the sprinkler (*Unit*: $Gal\ min^{-1}ft^{-2}$)**: The average rate at which water is sprayed onto crops. It depends on the size of sprinkler nozzles, the operating pressure and the distance between sprinklers. The basic infiltration rate of the soil should be higher than the average application rate of the sprinkler, so higher absorption of water can nullify or minimize runoff.

xi. **Flow rate of the sprinkler (Unit: Gal/min)**: The sum of the flow rates through the individual sprinklers of the entire sprinkler system which depends on water pressure.

The dimensions of the sprinkler opening and frictional losses in the sprinkler line also affect the overall flow rate.

xii. **Texture of the soil**: The type of soil which is used to irrigate turfgrasses.

xiii. **Effective depth of rooting (Unit: cm)**: The depth of soil used by the main body of the plant roots to obtain most of the stored moisture and plant food under proper irrigation.

xiv. **Infiltration rate (Unit: cm)**: The velocity at which the water enters the soil. It is measured by the depth of the water layer that can enter into the soil in one hour.

xv. **Pre soil moisture (Unit: %)**: The soil moisture level of the soil before an irrigation cycle.

xvi. **Post soil moisture (Unit: %)**: The soil moisture level of the soil after an irrigation cycle.

xvii. **Type of turf being irrigated**: The breed of turfgrass which is being irrigated. In this case, St. Augustine turfgrass was used.

xviii. **Evapotranspiration rate (Unit: mm/day)**: The rate at which water was transferred from the land to the atmosphere by evaporation from the soil and other surfaces and by transpiration from plants.

xix. **Runoff volume observed (Unit: Gallons)**: The amount of runoff recorded by the runoff meter after running the irrigation cycle for a stipulated amount of time.

xx. **Precipitation (Unit: inch)**: A major part of the hydrologic cycle, which is responsible for depositing most of the fresh water on the planet.

xxi. **Irrigation volume observed (Unit: Gallons):** The amount of water used for irrigation while running an irrigation cycle. It is measured using meters installed below the soil where the pre and the post readings are taken and their difference gives the total irrigation volume.

3.1.5 Final set of features used for analysis

There were a certain set of features which were needed to be removed since the number of observations were too little to work with a total of 21 features in the analysis. This increased the case of overfitting the model and giving erroneous outputs on the testing data. In Table 3.3, the list of features which had been removed from the analysis have been stated along with the reason for their removal.

Table 3.3: List of features removed from the analysis

Feature No	Predictor Name	Reason For Removal
i	**Test date**	Irrelevant
ii	**Start time**	Already encoded into 4 classes (4:Evening, 2:Morning, 1:Night, 3:Afternoon)
iii	**Solar radiation**	Significant correlation with Evapotranspiration Rate
iv	**Temperature**	Significant correlation with Evapotranspiration Rate
v	**Vapor Pressure Deficit**	Significant correlation with Evapotranspiration Rate
vi	**Area to be irrigated**	Variance = 0 (since all the 6 plots used are of same area)
vii	**Application rate**	Variance = 0 (since the sprinkler parameter used is the same)
viii	**Flow rate**	Variance = 0 (since the sprinkler parameter used is the same)
ix	**Texture of the soil**	Variance = 0 (since the soil used is the same)
x	**Effective depth of rooting**	Variance = 0 (since the soil used is the same)
xi	**Infiltration rate**	Variance = 0 (since the soil used is the same)
xii	**Pre soil moisture**	Significant correlation with Soil Wetting Efficiency Index
xiii	**Post soil moisture**	Significant correlation with Soil Wetting Efficiency Index
xiv	**Type of turf being irrigated**	Variance = 0 (since the turfgrass used is the same in all the 6 plots)

The final list of features used in the analysis were as follows:

i. **Reference Evapotranspiration rate**

ii. **Average wind speed**

iii. **Scheduled irrigation time**

iv. **Precipitation**

v. **Start time class:** The start time variable has been encoded into 4 classes: 1:Night, 2:Morning, 3:Afternoon and 4:Evening

These were the final list of features used in the analysis and the response or target variable was changed to see which approach worked the best for the application. Details about this have been discussed in section 3.2.

3.2 Selection of appropriate response or target variable for runoff minimization

3.2.1 Approach I

In this approach, the above 5 predictors were used and the runoff observed was estimated. The model was trained on the data generated by the CONTROL approach. Here, 'Runoff Volume Observed' was treated as a categorical variable which is binned on the basis of its distribution as shown in Fig. 3.6.

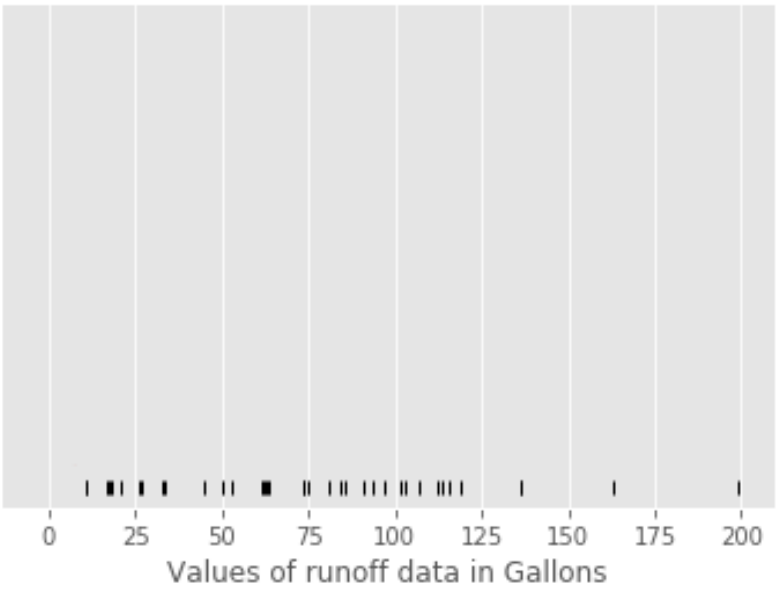

Figure 3.6: Distribution of the runoff volume observed by CONTROL approach

The values of runoff data below 105 gallons show up as one cluster and the values greater than 105 gallons show up as another to classify them either as Class 0 or Class 1.

3.2.2 Approach II

Here, the time when the first instance of runoff was observed (Unit: seconds) was estimated. Using CONTROL approach, an irrigation cycle was carried out for a stipulated time, and runoff was observed for the first time in a few minutes of its operation. The distribution of the variable is shown in Fig. 3.7.

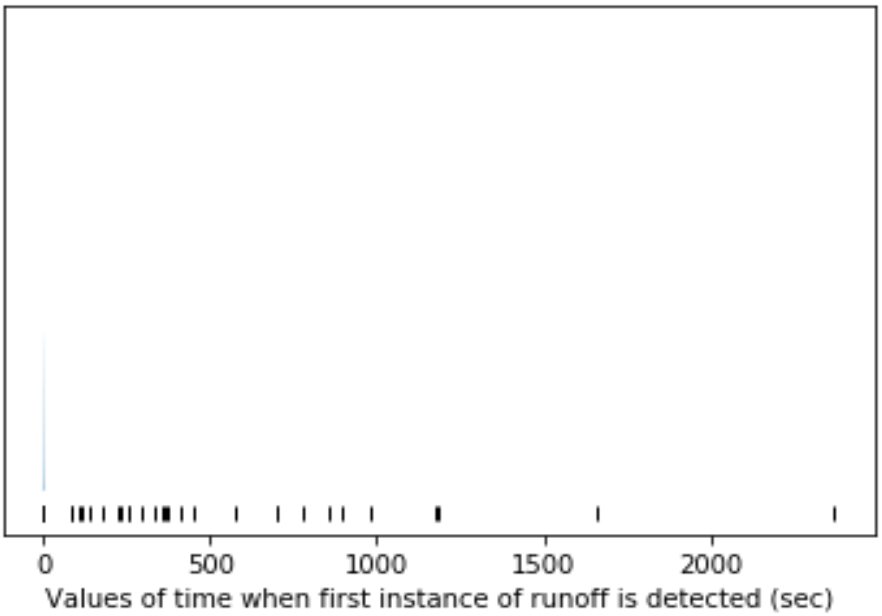

Figure 3.7: Distribution of the variable showing the first instance of runoff in seconds

This variable was binned into 3 categories. For all the cases when the first instance of runoff was detected at 0 seconds was categorized as Category 0. Similarly, all the values between 0 and 600 seconds and any value beyond have been categorized as Category 1 and Category 2, respectively.

3.2.3 Approach III

Here, the Soil Wetting Efficiency Index (SWEI) was used as the target variable. For this, the soil moisture reading was recorded after each irrigation cycle with the help of a soil moisture meter as shown in Fig. 3.2 and the irrigation volume before and after an irrigation cycle was recorded using flowmeters. The formula for Soil Wetting Efficiency Index, proposed by Wherley, *et. al* [33] is as follows:

$$SOIL\ WETTING\ EFFICIENCY\ INDEX = \frac{\left[\left[\frac{SM_{post} - SM_{pre}}{SM_{pre}}\right] * 1000\right]}{IRRIGATION\ VOLUME} \qquad (3.5)$$

By far, this seems to be the most reliable target variable for analysis since this variable is not site-specific and is a more generic approach compared to others.

The distribution of the Soil Wetting Efficiency Index is shown in Fig. 3.8.

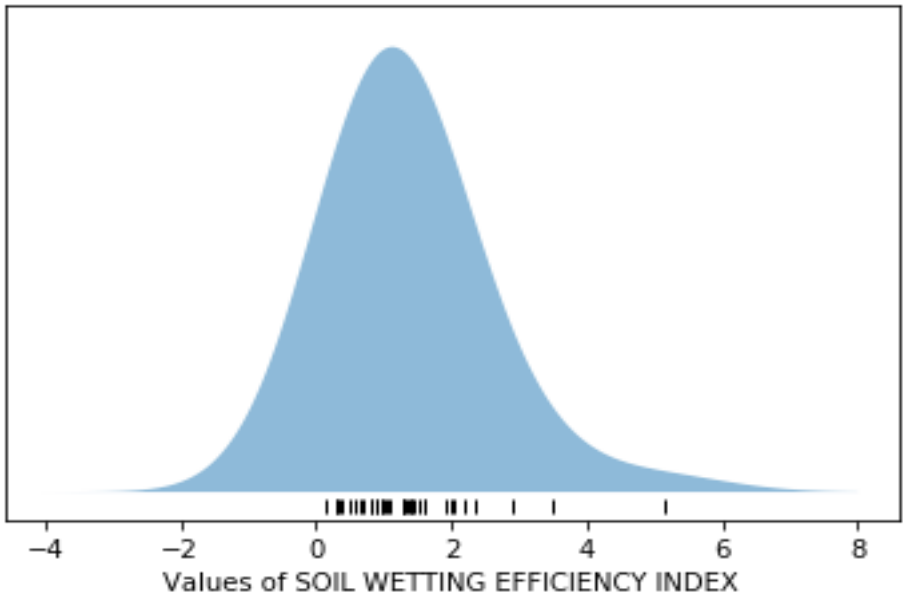

Figure 3.8: Distribution of the variable showing the Soil Wetting Efficiency Index

Using this approach, the Soil Wetting Efficiency Index was categorized into 2 or 3 classes as follows.

(a) Value of Soil Wetting Efficiency Index below 1.7 was categorized as Class 0 and the rest as Class 1 respectively.

(b) Value of Soil Wetting Efficiency Index below 1.5 was categorized as Class 0, values of Soil Wetting Efficiency Index between 1.5 and 2.2 was categorized as Class 1 and the rest as Class 2, respectively.

Both these sub-approaches were used to generate synthetic data and test our classifiers, which have been discussed in sections 3.4 and 3.5.

3.3 Data visualization

Data visualization is a very important tool to check whether data is linearly separable or not. For this purpose, Principal Component Analysis was used to see whether data show up in clusters so that it can be used for further analysis of the data.

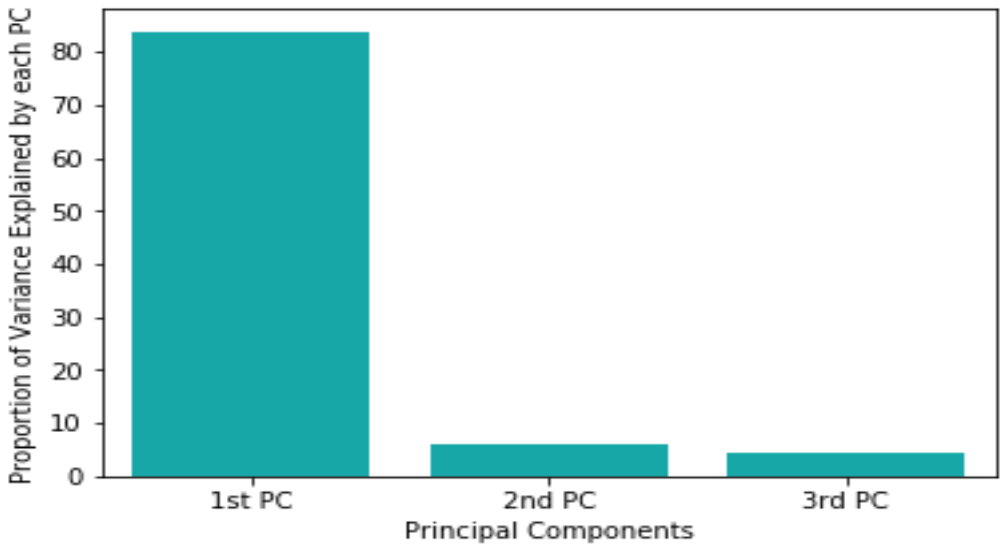

Figure 3.9: Plot showing Proportion of Variance Explained Vs the Number of PCs used in our analysis

From Fig. 3.9, using the Principal Components for analysis, 84% of the total variance in the dataset was explained by the first Principal Component, 6% by the second Principal Component and 4% by the third Principal Component respectively.

Table 3.4: Loading matrix of the first 3 Principal Components used in the analysis

	ETO (mm/day)	Avg wind speed (mph)	Precipitation (inch)	Effective Irrigation Time (min)	Start time class
PC – 1	-0.08	0.07	0.07	-0.03	0.99
PC – 2	0.42	0.44	-0.08	-0.79	-0.02
PC – 3	-0.43	0.21	0.85	-0.2	-0.12

From the loading matrix shown above, the features transformed into Principal Components were observed. The first Principal Component which explained 84% of the total variance in the dataset showed 'Start time class' as being the most strongly correlated predictor in PC-1 with a correlation coefficient of 0.99. The second and the third Principal Components showed 'Effective Irrigation Time (min)' and 'Precipitation (inch)' as the most important predictors respectively as they showed strong correlation with the respective PCs.

As the size of the dataset was small to begin with, dimensionality reduction was a very important step and from Fig. 3.9, it was inferred that three Principal Components were enough to explain 94% of the total variance in the data. Therefore, there was a need to observe how the three Principal Components clustered up against each other, and a decision was made whether to use this information for further analysis of the data.

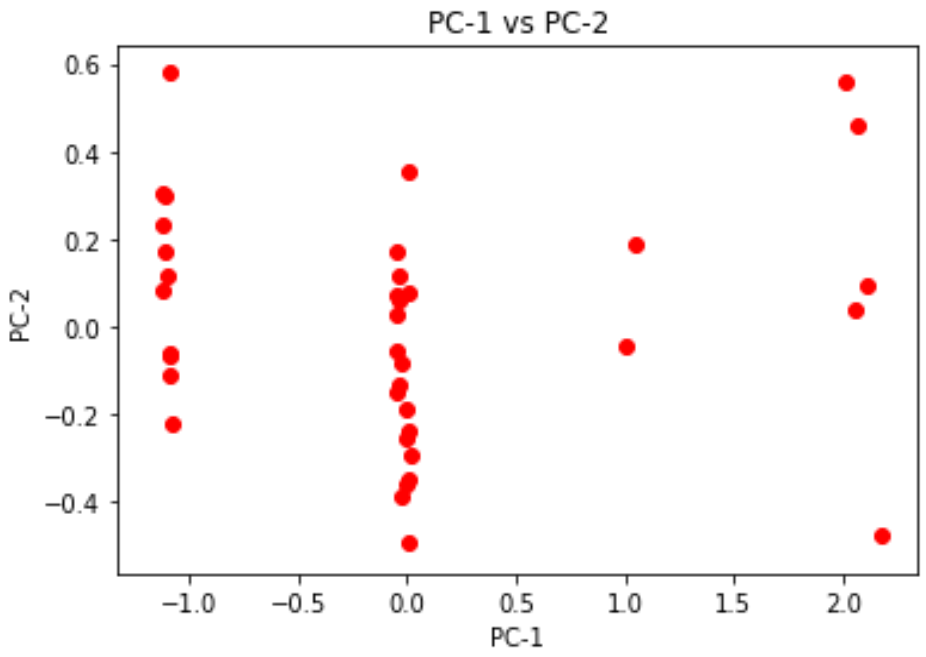

Figure 3.10: Clustering of the datapoints considering the 1st two Principal Components

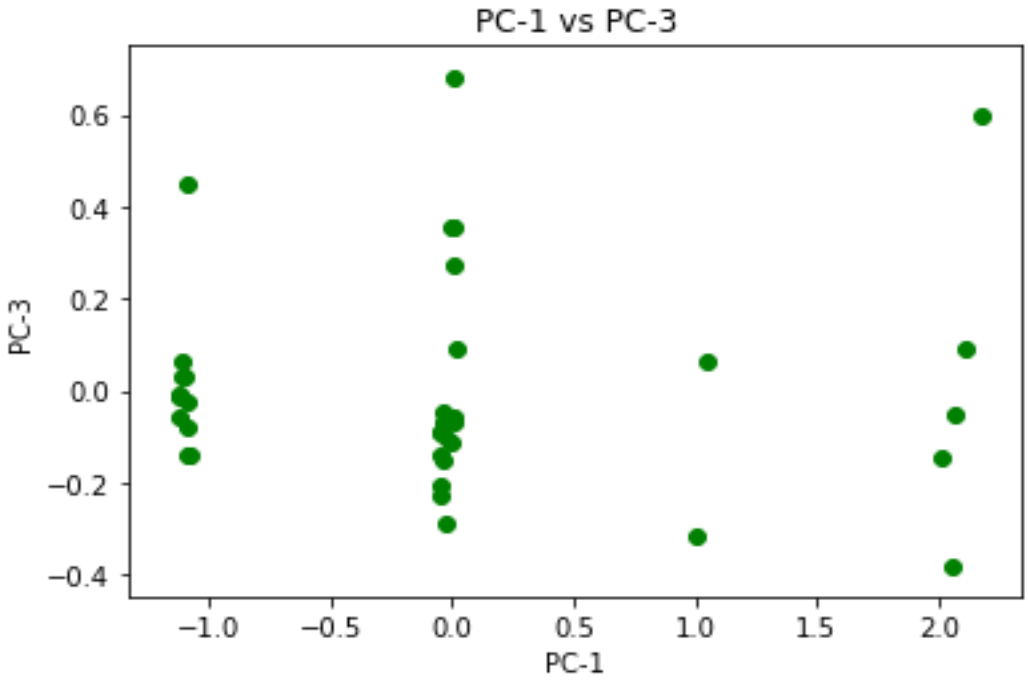

Figure 3.11: Clustering of the datapoints considering the 1st and the 3rd Principal Components used in the analysis

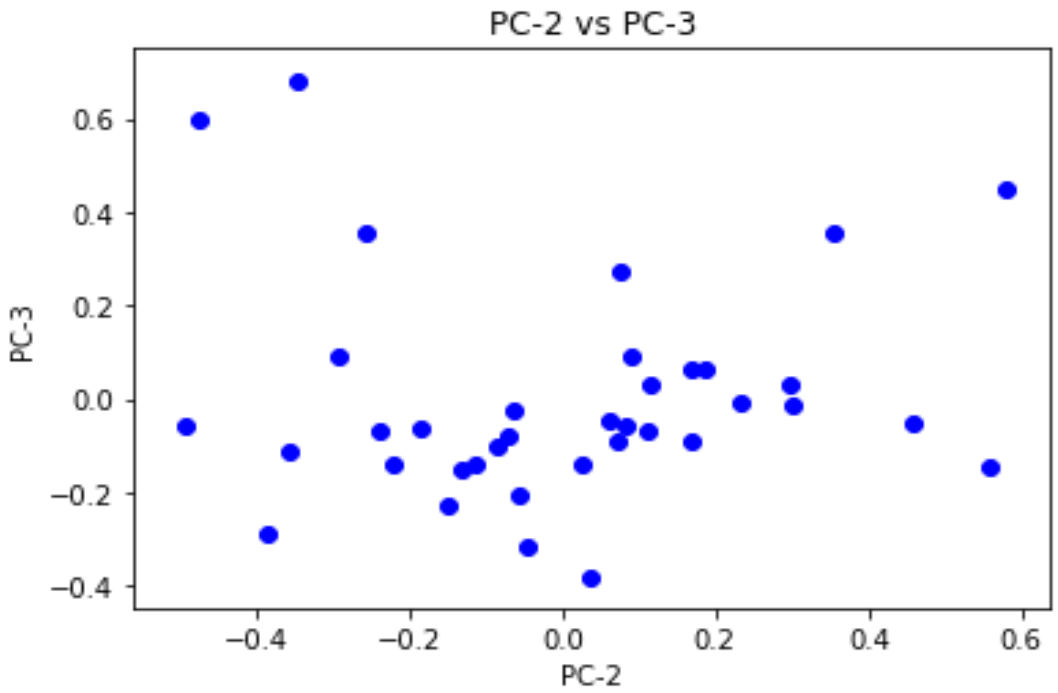

Figure 3.12: Clustering of the datapoints considering the 2nd and the 3rd Principal Components used in the analysis

From Fig. 3.10 and 3.11, on observing the plots between PC-1 vs PC-2 and PC-1 vs PC-3, four distinct elongated clusters were observed. It was inferred from the plots that PC-1 is the most dominant Principal Component. To further assert this inference, the feature 'Start Time Class' which was grouped into four categories while analyzing the data, dominated the plots as it had the highest coefficient in the first Principal Component.

From Fig. 3.12, on observing the plot between PC-2 vs PC-3, there was no clustering of the datapoints at all. Therefore, it was inferred that both the Principal Components explained equal amount of variance in the dataset and none of them dominated over the other.

3.4 Generation of synthetic data

There are many ways of generating synthetic data, which have been discussed before in the literature review section. In this study, the number of observations at hand are few, therefore, a special case of the Monte Carlo (MC) approach for synthetic data generation was used. For each binning of the target into different classes, a shared and a distinct mean and covariance matrix was generated among the classes. The simulations were run to generate data according to the distribution of the matrices.

Before generating synthetic data for all these approaches, all the datapoints were normalized. Normalizing the data entails subtracting the mean of the predictors from each of the datapoints and then dividing the values by their corresponding Standard Deviation.

(a) Approach I [Any observed volume of runoff below 105 Gallons was classified as Class 0 and the rest as Class 1, respectively]

Generation of data using distinct mean and covariance matrix for the classes: [500 observations each], as follows:

START TIME CLASS = 0 and RUNOFF VOLUME CLASS = 0

START TIME CLASS = 1 and RUNOFF VOLUME CLASS = 0

START TIME CLASS = 2 and RUNOFF VOLUME CLASS = 0

START TIME CLASS = 3 and RUNOFF VOLUME CLASS = 0

START TIME CLASS = 0 and RUNOFF VOLUME CLASS = 1

START TIME CLASS = 1 and RUNOFF VOLUME CLASS = 1

START TIME CLASS = 2 and RUNOFF VOLUME CLASS = 1

START TIME CLASS = 3 and RUNOFF VOLUME CLASS = 1

Total number of observations sharing distinct mean and covariance matrix among the classes = 4000 observations X 5 predictors

Generation of data using shared mean and covariance matrix for the classes: [1000 observations each], as follows:

START TIME CLASS = 0 and RUNOFF VOLUME CLASS = 0,1
START TIME CLASS = 1 and RUNOFF VOLUME CLASS = 0,1
START TIME CLASS = 2 and RUNOFF VOLUME CLASS = 0,1
START TIME CLASS = 3 and RUNOFF VOLUME CLASS = 0,1

Total number of observations using shared mean and covariance matrix among the classes = 4000 observations X 5 predictors

Total number of observations = 8000 observations X 5 predictors

(b) Approach II [Any observed value of time until first runoff = 0 seconds was classified as Class 0, any observed value between 0 and 600 seconds was classified as Class 1 and anything beyond was classified as Class 2 respectively]

Generation of data using distinct mean and covariance matrix for the classes: [500 observations each], as follows:

START TIME CLASS = 0 and TIME UNTIL FIRST RUNOFF CLASS = 0
START TIME CLASS = 1 and TIME UNTIL FIRST RUNOFF CLASS = 0
START TIME CLASS = 2 and TIME UNTIL FIRST RUNOFF CLASS = 0
START TIME CLASS = 3 and TIME UNTIL FIRST RUNOFF CLASS = 0
START TIME CLASS = 0 and TIME UNTIL FIRST RUNOFF CLASS = 1
START TIME CLASS = 1 and TIME UNTIL FIRST RUNOFF CLASS = 1

START TIME CLASS = 2 and TIME UNTIL FIRST RUNOFF CLASS = 1

START TIME CLASS = 3 and TIME UNTIL FIRST RUNOFF CLASS = 1

START TIME CLASS = 0 and TIME UNTIL FIRST RUNOFF CLASS = 2

START TIME CLASS = 1 and TIME UNTIL FIRST RUNOFF CLASS = 2

START TIME CLASS = 2 and TIME UNTIL FIRST RUNOFF CLASS = 2

START TIME CLASS = 3 and TIME UNTIL FIRST RUNOFF CLASS = 2

Total number of observations sharing distinct mean and covariance matrix among the classes = 6000 observations X 5 predictors

Generation of data using shared mean and covariance matrix for the classes: [1000 observations each], as follows:

START TIME CLASS = 0 and TIME UNTIL FIRST RUNOFF CLASS = 0,1,2

START TIME CLASS = 1 and TIME UNTIL FIRST RUNOFF CLASS = 0,1,2

START TIME CLASS = 2 and TIME UNTIL FIRST RUNOFF CLASS = 0,1,2

START TIME CLASS = 3 and TIME UNTIL FIRST RUNOFF CLASS = 0,1,2

Total number of observations using shared mean and covariance matrix among the classes = 4000 observations X 5 predictors

Total number of observations = 10000 observations X 5 predictors

(c) Approach III:

CASE I: [Any value of Soil Wetting Efficiency below 1.7 was classified as Class 0 and the rest as Class 1 respectively]

Generation of data using distinct mean and covariance matrix for the classes: [500 observations each], as follows:

START TIME CLASS = 0 and SOIL WETTING EFFICIENCY INDEX CLASS = 0

START TIME CLASS = 1 and SOIL WETTING EFFICIENCY INDEX CLASS = 0

START TIME CLASS = 2 and SOIL WETTING EFFICIENCY INDEX CLASS = 0

START TIME CLASS = 3 and SOIL WETTING EFFICIENCY INDEX CLASS = 0

START TIME CLASS = 0 and SOIL WETTING EFFICIENCY INDEX CLASS = 1

START TIME CLASS = 1 and SOIL WETTING EFFICIENCY INDEX CLASS = 1

START TIME CLASS = 2 and SOIL WETTING EFFICIENCY INDEX CLASS = 1

START TIME CLASS = 3 and SOIL WETTING EFFICIENCY INDEX CLASS = 1

Total number of observations sharing distinct mean and covariance matrix among the classes = 4000 observations X 5 predictors

Generation of data using shared mean and covariance matrix for the classes: [1000 observations each], as follows:

START TIME CLASS = 0 and SOIL WETTING EFFICIENCY INDEX CLASS = 0,1

START TIME CLASS = 1 and SOIL WETTING EFFICIENCY INDEX CLASS = 0,1

START TIME CLASS = 2 and SOIL WETTING EFFICIENCY INDEX CLASS = 0,1

START TIME CLASS = 3 and SOIL WETTING EFFICIENCY INDEX CLASS = 0,1

Total number of observations using shared mean and covariance matrix among the classes = 4000 observations X 5 predictors

Total number of observations = 8000 observations X 5 predictors

CASE II: [Any value of Soil Wetting Efficiency Index below 1.5 was classified as Class 0, any value between 1.5 and 2.2 was classified as Class 1 and the rest as Class 2, respectively]

Generation of data using distinct mean and covariance matrix for the classes: [500 observations each], as follows:

START TIME CLASS = 0 and SOIL WETTING EFFICIENCY INDEX CLASS = 0

START TIME CLASS = 1 and SOIL WETTING EFFICIENCY INDEX CLASS = 0

START TIME CLASS = 2 and SOIL WETTING EFFICIENCY INDEX CLASS = 0

START TIME CLASS = 3 and SOIL WETTING EFFICIENCY INDEX CLASS = 0

START TIME CLASS = 0 and SOIL WETTING EFFICIENCY INDEX CLASS = 1

START TIME CLASS = 1 and SOIL WETTING EFFICIENCY INDEX CLASS = 1

START TIME CLASS = 2 and SOIL WETTING EFFICIENCY INDEX CLASS = 1

START TIME CLASS = 3 and SOIL WETTING EFFICIENCY INDEX CLASS = 1

START TIME CLASS = 0 and SOIL WETTING EFFICIENCY INDEX CLASS = 2

START TIME CLASS = 1 and SOIL WETTING EFFICIENCY INDEX CLASS = 2

START TIME CLASS = 2 and SOIL WETTING EFFICIENCY INDEX CLASS = 2

START TIME CLASS = 3 and SOIL WETTING EFFICIENCY INDEX CLASS = 2

Total number of observations sharing distinct mean and covariance matrix among the classes = 6000 observations X 5 predictors

Generation of data using shared mean and covariance matrix for the classes: [1000 observations each], as follows:

START TIME CLASS = 0 and SOIL WETTING EFFICIENCY INDEX CLASS = 0,1,2

START TIME CLASS = 1 and SOIL WETTING EFFICIENCY INDEX CLASS = 0,1,2

START TIME CLASS = 2 and SOIL WETTING EFFICIENCY INDEX CLASS = 0,1,2

START TIME CLASS = 3 and SOIL WETTING EFFICIENCY INDEX CLASS = 0,1,2

Total number of observations using shared mean and covariance matrix among the classes = 4000 observations X 5 predictors

Total number of observations = 10000 observations X 5 predictors

3.5 Results

Radial Basis Function - Support Vector Machine (RBF-SVM) was used in each case to test the approach. For different values of penalty parameter, the value of Gamma parameter was tuned using param_grid ( ). Two different types of tests were used to support the approach and the best model was selected based on the results. Accordingly, appropriate irrigation rules were defined for the model, which produced best results.

The two types of tests which have been used in our analysis are as follows:

(a) The synthetic dataset for both training and testing the model was split in the ratio X (training):Y (testing), where both X and Y vary between 10 and 100 in intervals of 10 units used.

(b) The synthetic dataset was used for training the model and the real dataset was used for validation purposes.

Now, for all the three approaches, the results were obtained, and the best model was chosen for implementation which is explained in the next section.

3.5.1 Approach I [Classification of runoff as the target class]

The values of accuracy on the test data for different values of penalty parameter and the best Gamma are as follows. The Python code used to design this approach is shown in APPENDIX A. The values of test size for validating the model are shown in each case, and the values with the highest accuracy have been highlighted in red as shown in Table 3.5.

Table 3.5: Testing accuracies for different Test:Train Split Ratio when the target variable is 'Runoff volume observed'

	Test Size = 0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
C = 1	81.33	80.47	80.84	80.23	80.08	79.67	79.3	78.08	78.68
C = 10	82	81.93	81.91	82.16	81.89	81.24	81.4	79.83	80.92
C = 20	82.93	81.87	81.91	82.1	81.94	82.06	81.77	80.31	80.63
C = 30	82.67	81.93	81.96	82	82.13	81.91	81.7	80.35	80.72
C = 40	82.67	82	82.13	82.3	82.21	82.22	82.15	80.4	80.52
C = 50	82.67	82.13	82.27	82.3	82.24	82.17	82.08	80.36	80.49
C = 60	82.67	82.2	82.31	82.17	82.10	82.13	81.77	80.26	80.46
C = 70	82.67	82.13	82.35	82.1	82.03	82.15	82.08	80.18	80.4
C = 80	83.2	82.2	82.76	82.27	82.10	82.08	82.08	80.23	80.28
C = 90	83.2	82.27	82.8	82.23	82.13	82.13	82.09	80.03	80.22
C = 100	83.06	82.33	82.84	82.3	82.16	81.93	82.17	80.02	80.38

Once the model was trained using synthetic data generated, the classifier was tested on the real dataset.

Table 3.6: Testing accuracies when the RBF-SVM is trained on the synthetic data and target variable is 'Runoff volume observed'

Value of Penalty	Testing Accuracy
C =1	83.78
C = 10	83.78
C = 20	89.2
C = 30	89.2
C = 40	89.2
C = 50	89.2
C = 60	89.2
C = 70	89.2
C = 80	89.2
C = 90	89.2
C = 100	89.2

The highest accuracy was achieved on the testing dataset when the value of penalty parameter was 20 and the score was 89.2% (Table 3.6).

3.5.2 **Approach II** [Classification of time until first runoff as the target class]

Here, the best value of Gamma for each train:test split has been chosen for different penalty parameters and the best testing accuracies have been highlighted in red as in Approach I, which is shown in Table 3.7. The Python code used to design this approach is shown in APPENDIX B.

Table 3.7: Testing accuracies for different Test:Train Split Ratio when the target variable is 'The time until the 1st instance of Runoff was observed'

	Test Size = 0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
C = 1	78	77.14	77.62	76.57	76.48	76.2	75.31	74.05	71.4
C = 10	77	77.36	78.62	77.32	77.29	77.45	76.71	76.02	73.2
C = 20	77.29	77.29	77.81	77.21	77.11	77.19	76.63	75.92	72.95
C = 30	78	77.29	77.9	77.32	77.09	77.19	76.5	75.78	72.79
C = 40	77.71	77.29	77.76	77.17	77.2	77.3	76.57	75.78	72.41
C = 50	77.43	77.43	77.66	77.18	77.2	77.42	76.59	75.67	72.28
C = 60	78	77.29	77.52	77.82	77.25	77.48	76.69	75.59	72.19
C = 70	78.14	77.43	77.86	77.82	77.31	77.5	76.69	75.52	72.11
C = 80	78	77.29	77.67	77.75	77.37	77.48	76.73	75.48	72.05
C = 90	78.29	77.43	77.67	77.79	77.34	77.45	76.61	75.44	71.82
C = 100	78.29	77.5	77.43	77.82	77.37	77.48	76.41	75.45	72.43

Once the model was trained using synthetic data just as done in Approach I, the model was tested on the real data and the testing accuracies have been shown below.

Table 3.8: Testing accuracies when the RBF-SVM is trained on the synthetic data and target variable is 'The time until 1st instance of runoff is observed'

Value of Penalty	Testing Accuracy
C =1	75
C = 10	75
C = 20	75
C = 30	77.8
C = 40	75
C = 50	77.78
C = 60	77.78
C = 70	77.78
C = 80	77.78
C = 90	77.78
C = 100	77.78

The highest accuracy was achieved on the testing dataset when the value of penalty parameter was 30 and the score obtained was 77.8% (Table 3.8).

3.5.3 Approach III

CASE I: Classification of Soil Wetting Efficiency Index into two classes

The Python code used to design this approach is shown in APPENDIX C(a). Here, the size of the testing dataset was adjusted as done in both of the abovementioned approaches, and the results were included in Table 3.9, as follows:

Table 3.9: Testing accuracies for different Test:Train Split Ratio when the target variable is SWEI (Binned into 2 classes)

	Test Size = 0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
C = 1	82.17	80.92	82	81.75	81.3	80.75	80.07	77.64	74.19
C = 10	83.5	81.92	83.5	83.5	83.43	83.25	82.76	81.73	79.98
C = 20	83.67	81.42	83.5	83.21	82.87	83.33	82.83	82.21	80.5
C = 30	83.67	81.33	83.2	83.71	83.13	83.16	82.54	82.42	80.46
C = 40	83.67	81.25	83.28	83.67	83.77	83.25	83.05	82.27	80.59
C = 50	83.5	81.67	83.17	83.46	83.8	83.28	82.88	82.38	80.56
C = 60	83.5	81.25	83.22	83.58	83.76	83.19	82.83	82.44	80.3
C = 70	83	81.58	83.17	83.62	83.76	83.33	82.97	82.38	80.28
C = 80	83	81.75	83.11	83.5	83.76	83.42	82.69	82.52	80.37
C = 90	82.83	81.67	83.05	83.67	83.73	83.19	82.67	82.44	80.37
C = 100	82.83	81.67	83	83.58	83.73	83.19	82.67	82.44	80.41

Once the model was trained using synthetic data generated in each case, the classifier was tested on the real dataset. The highest accuracy was achieved on the testing dataset when the value of penalty parameter was 100 and the score obtained was 86.49% (Table 3.10).

Table 3.10: Testing accuracies when the RBF-SVM is trained on the synthetic data and target variable is SWEI (Binned into 2 classes)

Value of Penalty	Testing Accuracy
C =1	72.97
C = 10	72.97
C = 20	72.97
C = 30	75.67
C = 40	75.67
C = 50	78.38
C = 60	81.08
C = 70	81.08
C = 80	83.78
C = 90	83.78
C = 100	86.49

CASE II: Classification of Soil Wetting Efficiency Index into three classes

The Python code used to design this approach is shown in APPENDIX C(b). The values of accuracy on the test data for different values of penalty parameter and the best Gamma are shown in Table 3.11. The best testing accuracies have been highlighted in red.

The model was also trained using synthetic data generated in each case, and the classifier was tested on the real dataset which is shown in Table 3.12.

Table 3.11: Testing accuracies for different Test:Train Split Ratio when the target variable is SWEI (Binned into 3 classes)

	Test Size = 0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
C = 1	63.7	64.85	63.85	63.07	62.70	62.43	62.42	61.05	57.93
C = 10	65.54	66.15	65.95	65.6	64.89	65.64	65.03	64.37	61.89
C = 20	65.23	66.46	65.8	65.7	64.58	65.64	65.38	64.5	62.75
C = 30	65.38	66.38	65.8	65.9	64.58	65.6	65.49	64.52	62.84
C = 40	64.92	66.31	65.7	65.7	64.58	65.31	65.43	64.52	62.97
C = 50	65.08	66.38	65.64	65.6	64.74	65.21	65.49	64.6	63
C = 60	65.23	66.23	65.54	66	64.86	65.26	65.56	64.6	62.96
C = 70	64.46	66.53	65.74	65.92	64.74	65.23	65.36	64.6	62.92
C = 80	64.46	66.53	65.53	65.85	64.8	65.28	65.54	64.46	63.1
C = 90	64.46	66.30	65.6	65.77	64.92	65.26	65.38	64.5	63.04
C = 100	64.92	66.31	65.49	65.88	64.89	65.31	65.62	64.6	63

Table 3.12: Testing accuracies when the RBF-SVM is trained on the synthetic data and target variable is SWEI (Binned into 3 classes)

Value of Penalty	Testing Accuracy
C =1	67.57
C = 10	70.27
C = 20	72.97
C = 30	70.27
C = 40	70.27
C = 50	72.97
C = 60	72.97
C = 70	72.97
C = 80	72.97
C = 90	72.97
C = 100	72.97

The highest accuracy was achieved on the testing dataset when the value of penalty parameter was 20 and the score obtained was 72.97%.

3.6 Choosing the best model for implementation and deciding the irrigation rules

At first, after running data visualization to see the threshold for Soil Wetting Efficiency Index using historical dataset and the datapoints recorded for this year, the value of 1.7 was set as the threshold value [**Approach III: Case I**].

The values of SWEI predicted more than 1.7 were classified as observations which showed lower potential for runoff and vice-versa. Initially, the ML algorithm was proposedly used to predict the time of irrigation but it was decided that the time-to-irrigate should be calculated from the evapotranspiration rate and other climatic predictors directly by the controller, which was eventually used as a predictor in the analysis.

For the ML model, certain classification rules have been stated as follows:

(a) If SWEI predicted is more than 1.7,

Then

Irrigation time = 80% of the prescribed time

If first runoff is detected:

Pause Time = 30 min

Pulse Time = 2 min

If second runoff is detected:

Pause Time = 30 min

Pulse Time = 2 min

If third runoff is detected:

the loop is terminated

(b) If the SWEI predicted is less than 1.7,

Then

Irrigation time = 2 minutes

If first runoff is detected:

Pause Time = 1 hour

Pulse Time = 2 min

If second instance of runoff is detected:

the loop is terminated

With the approach described above, it is expected that runoff could be minimized during irrigation events.

CHAPTER IV

FIELD IMPLEMENTATION OF THE MACHINE LEARNING ALGORITHM IN THE CUSTOMIZED CONTROLLER

4.1 Proposed scheme/approach used by the controllers [Rachio, B-hyve and the customized ASIS Controller]

Before explaining the significance of Machine Learning algorithm in acting as a "Control" mechanism in the proposed controller, specific explanation is given about the other controllers which are being used actively in the industry. To have a fair assessment of the performance of the controllers, Rachio controller was installed in Plot 10, the proposed ASIS controller was installed in Plot 12, and the B-hyve controller was installed in Plot 13 respectively. To have a better idea about how the controllers regulate the irrigation cycle, the working of each controller is illustrated in detail below.

4.1.1 Irrigation scheme used by Rachio Controller

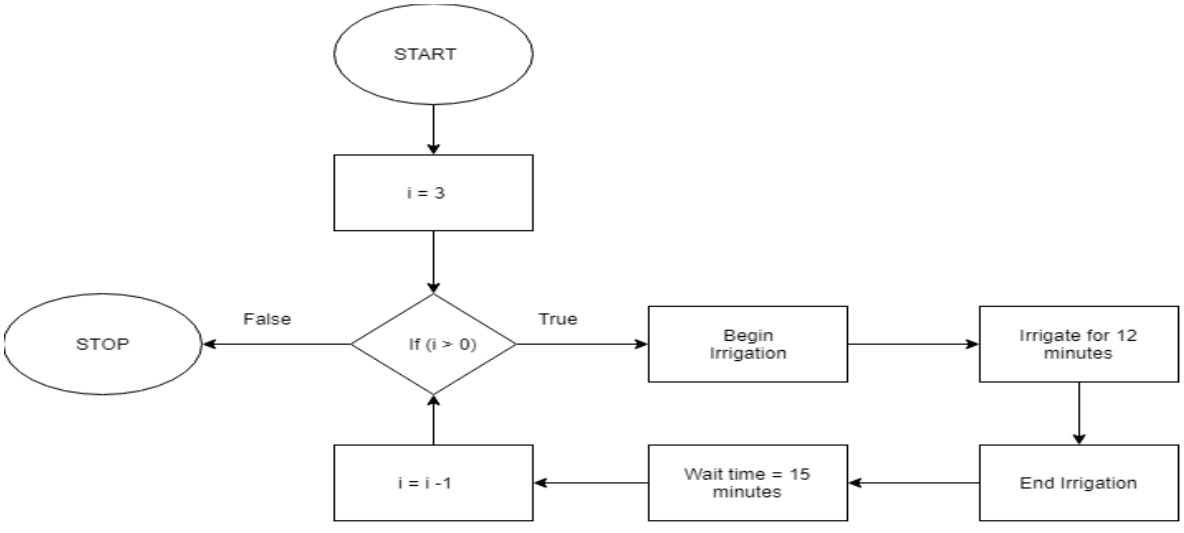

Figure 4.1: Schematic representation of the working of Rachio controller

From the flowchart, it is evident that the Rachio Irrigation controller performs a thrice executed cycle clock where it irrigates for 12-13 minutes per cycle and has a 15 minutes soak between each of the cycles. The controller decides the time to irrigate a month in advance by taking the weather forecast for the upcoming month. Although the timings for irrigation are not exactly predictable, it has been noticed from the irrigation runs that the irrigation cycles are usually scheduled two to three hours before sunrise.

4.1.2 Irrigation scheme used by B-hyve Controller

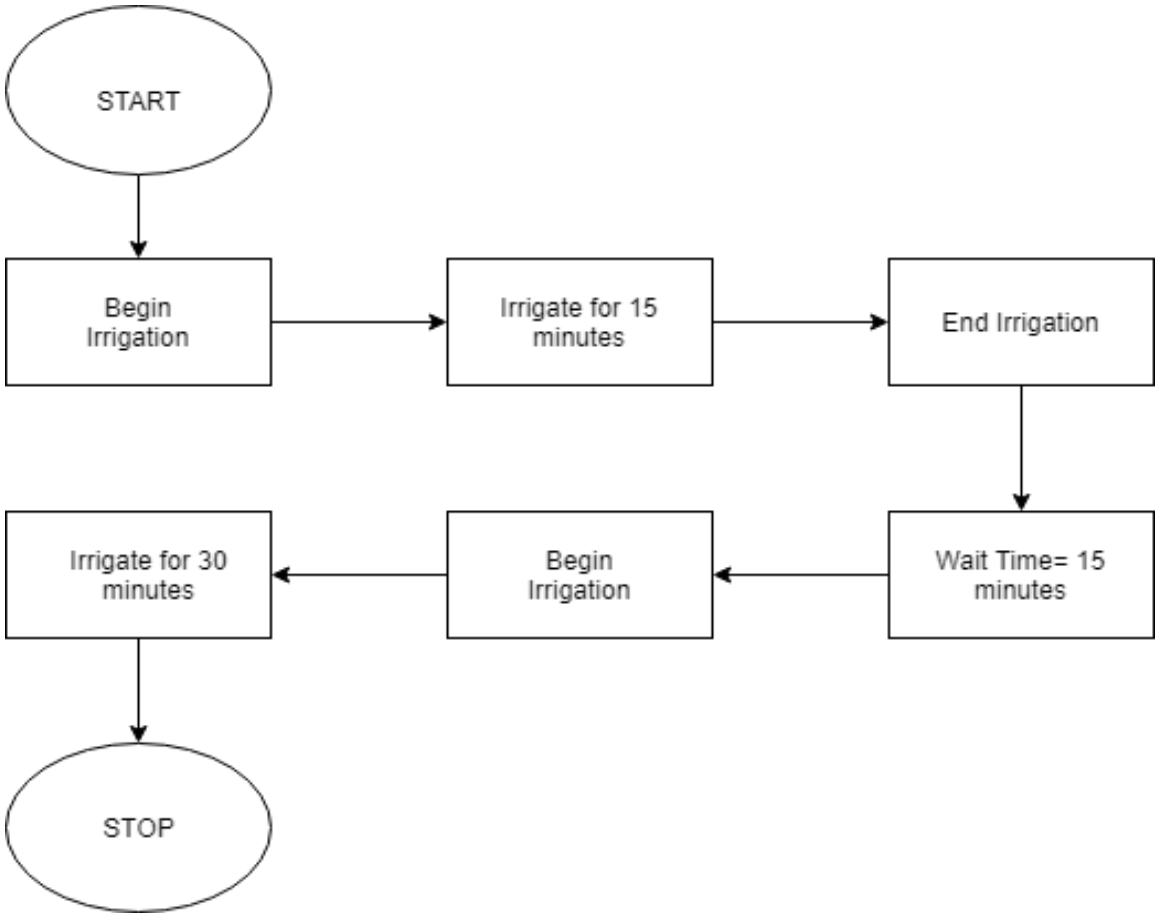

Figure 4.2: Schematic representation of the working of B-hyve controller

The irrigation scheme used by the B-hyve controller is much simpler compared to the other industrial controllers. It has a static operation time much like the abovementioned Rachio controller. It irrigates for a period of 15 minutes, followed by a 15 minutes soak period, repeats this again for a total of 30 minutes of irrigation and a total of 15 minutes for soaking. From the irrigation runs, it has been observed that the B-hyve controller begins the first irrigation cycle typically at 6 AM.

4.1.3 Irrigation scheme used by the proposed ASIS controller

The design of the entire ASIS controller was devised by Tomas Reyes, a senior in the Department of Electrical and Computer Engineering at Texas A&M University. The Machine Learning algorithm which acts as a Control algorithm in this case was devised by the author.

The design proposed by Tomas is discussed in detail in this section, whereas the impact of the Machine Learning approach in minimizing runoff used in the ASIS is discussed in Section 4.3.

To understand the irrigation scheme in detail, the entire algorithm has been broken up into four parts as follows:

a) Initialization of the algorithm
b) Algorithm for calculation of Evapotranspiration and Effective Rainfall
c) Machine Learning algorithm
d) Implementation and field deployment of the algorithm based on ML output

Each part has been explained in detail along with their block diagram.

(a) Initialization of the algorithm

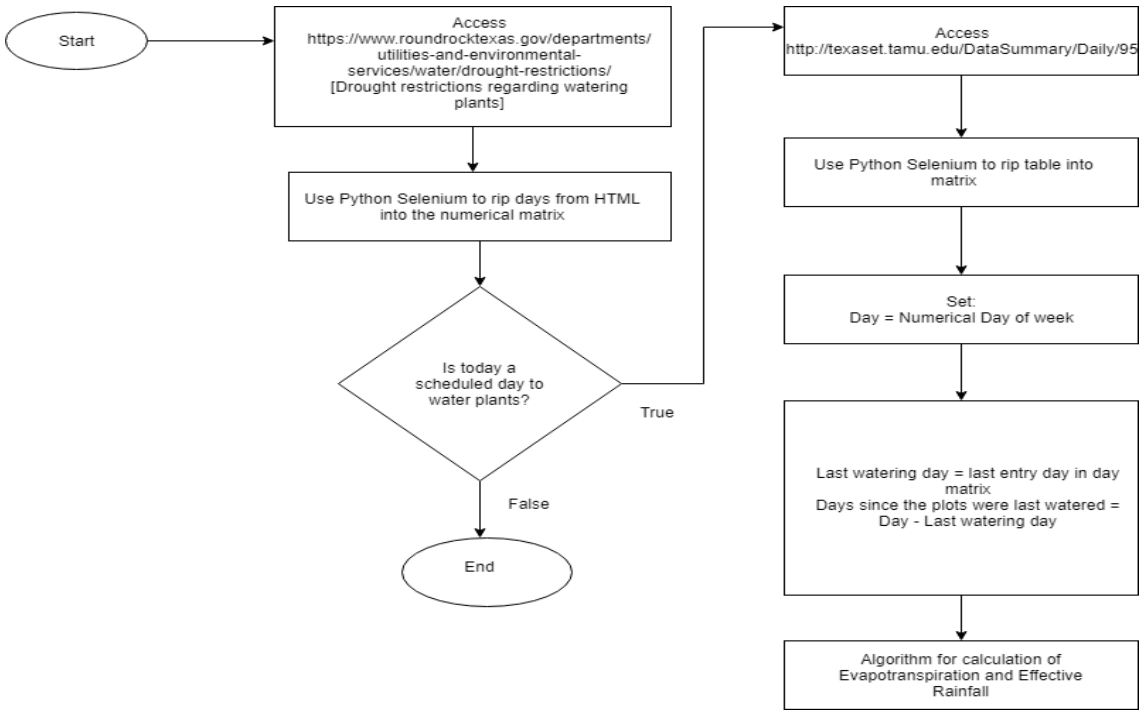

Figure 4.3: Schematic representation for the initialization algorithm used in the ASIS controller

Variable descriptions:

day: numerical index of the day of the week (0:Sunday, 1:Monday, 2:Tuesday,.. and so on)

last watering day: numerical index of the day when the plot was last watered (index of the last entry in the day matrix)

The initialization of the algorithm works as follows:

(i) The drought conditions and environmental restrictions are accessed for watering on a particular day, and 'Python Selenium' algorithm is used to rip those days into a matrix. All the days are stored in the matrix in numeric form e.g. Sunday: 0, Monday: 1, Tuesday: 2, Wednesday: 3, Thursday: 4, and so on.

(ii) The day scrapped from the website is compared with the present day's numeric index.

If both of them match, then:

- The data summary from Texas ET Network is scrapped for the day and is stored into a matrix using ‘Python Selenium’ algorithm.
- The ‘day’ is set to the numerical day of the week.
- Now, the ‘days since the plots were last watered’ is calculated which is the difference between ‘today’ and the ‘last watering day’ i.e. the numerical index when the plot was last watered is stored as the last entry day in day matrix.

(iii) Now, the algorithm for calculation of Evapotranspiration and Effective Rainfall is executed which is discussed in the next section.

(b) Algorithm for calculation of Evapotranspiration and Effective Rainfall

Branch I: Decision based on effective rainfall

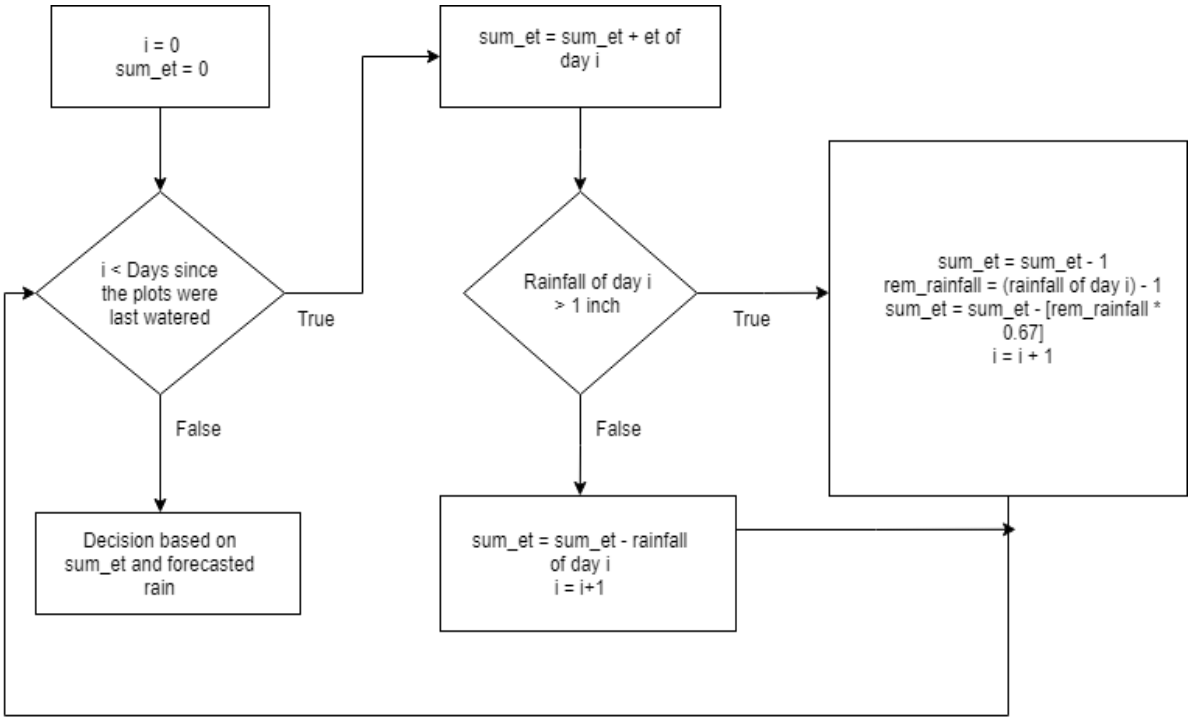

Figure 4.4: **Schematic representation for Evapotranspiration and Effective Rainfall based algorithm used in the ASIS controller**

Variable descriptions:

i: Iterative variable which stores the day index

sum_et: Cumulative Evapotranspiration rate until day i

rem_rainfall: Effective Rainfall

j: Iterative variable which is used for comparison of 'day' with 'the difference between the next watering day and today (with warp around)'

(i) At first, the value of 'i' is compared with 'the days since the plots were last watered'.

(ii) If the value of the iterative variable 'i' is more than 'the days since the plots were last watered', then Branch II i.e. the algorithm, which shows the decision based on cumulative Evapotranspiration and forecasted rain is executed which is discussed in the next sub-chapter **(Section 4.1.3 (b): Branch II).**

(iii) If the value of the iterative variable 'i' is less than 'the days since the plots were last watered',

- The cumulative Evapotranspiration rate (sum_et) is calculated by summing over the Evapotranspiration rates until day i.
- If the rainfall of day i [i.e. the day when the plot is supposed to be watered] is greater than 1 inch,
    - The cumulative Evapotranspiration rate (sum_et) is decremented by 1 inch.
    - The Effective Rainfall (rem_rainfall) is decremented by 1 inch.
    - The value of cumulative Evapotranspiration rate (sum_et) is decremented by 67% of the Effective Rainfall.
    - The iterative variable 'i' is incremented by 1.
    - Step (i) is repeated.
- If the rainfall of day i is less than 1 inch,

- The value of cumulative Evapotranspiration rate (sum_et) is calculated by subtracting the rainfall of day 'i' from the value of cumulative Evapotranspiration rate until day i.
- Step (i) is repeated.

Branch II: Decision based on Evapotranspiration and forecasted rain

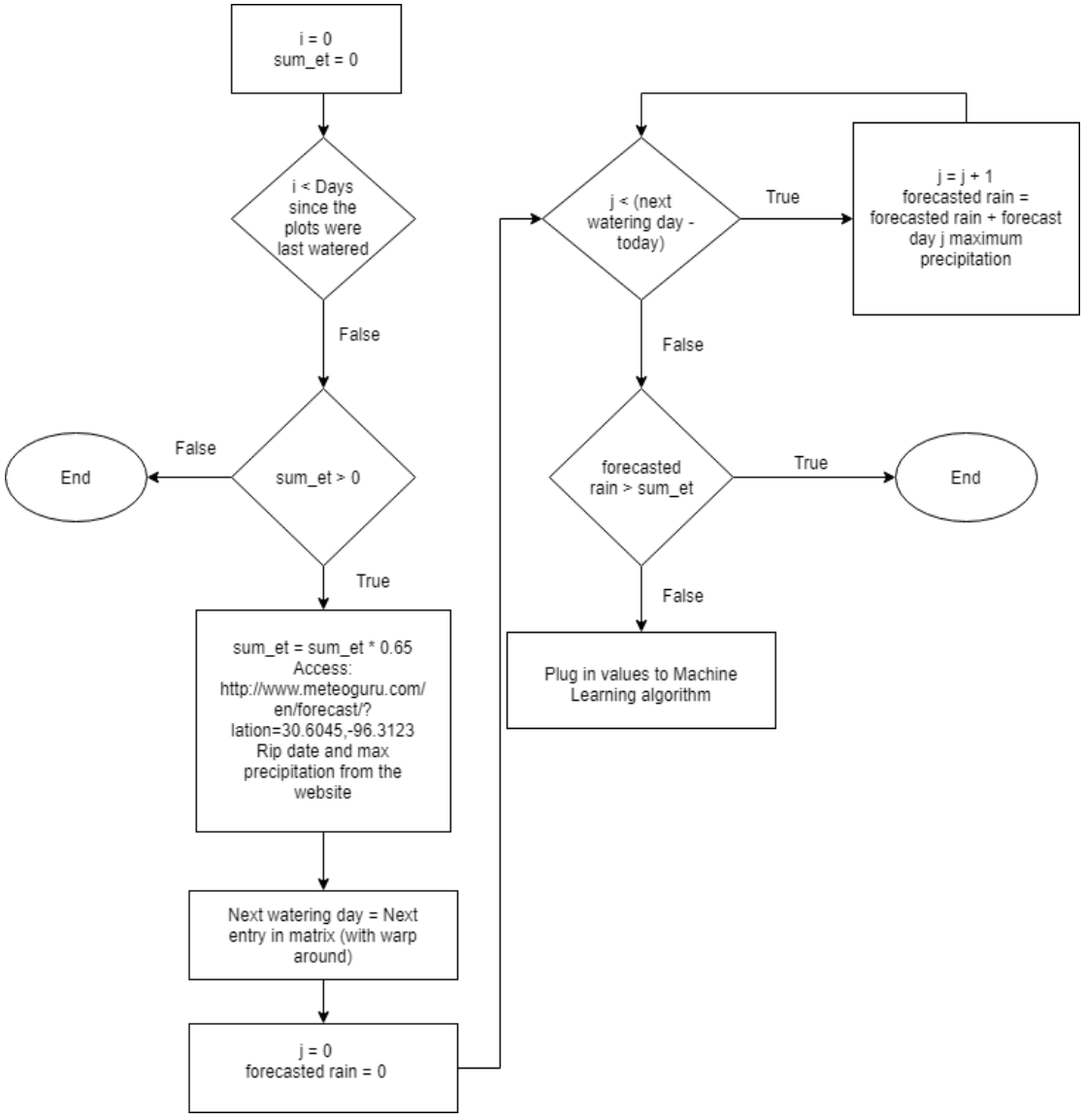

Figure 4.5: Schematic representation for cumulative Evapotranspiration rate and forecasted rain based algorithm used in the ASIS controller

This branch is executed when the value of the iterative variable ‘i’ is less than ‘the days since the plots were last watered’. The algorithm works in the following manner:

(i) If the value of cumulative Evapotranspiration rate (sum_et) is greater than zero,

- Cumulative Evapotranspiration rate (sum_et) is reduced to 65% of the value which is scrapped from the Texas ET Network.
- The date and maximum precipitation is ripped from ‘meteoguru.com’ website for that particular date.
- The ‘next watering day’ is the next entry in data matrix (with warp around).

(ii) If the value of iterative variable ‘j’ is less than the difference between the ‘next watering day’ and ‘today’,

- The value of iterative variable ‘j’ is incremented by 1.
- The value of ‘forecasted rain’ is incremented by the value of maximum precipitation expected on day ‘j’.
- Step (ii) is repeated.

(iii) If the value of iterative variable ‘j’ is greater than the difference between the ‘next watering day’ and ‘today’, the value of ‘forecasted rain’ is compared with the cumulative Evapotranspiration rate (sum_et).

If the value of ‘forecasted rain’ is greater than the ‘cumulative Evapotranspiration rate’, then the algorithm is terminated, else the values are plugged into the Machine Learning algorithm which has been briefly discussed in the next section (Section 4.1.3(c)).

(c) Machine Learning algorithm:

Radial Basis Function-Support Vector Machine (RBF-SVM) is applied and the Soil Wetting Efficiency Index is observed by binning them into two categories (0 and 1). Details about the algorithm have already been discussed in Chapter 3 of the thesis.

(d) Implementation and field deployment of the algorithm based on ML output

Variable descriptions:

watering time: This is calculated as the product between the cumulative Evapotranspiration rate and the max allowable irrigation time which is 35 minutes in this case

Active time: The length of time for which the controller has been running. The minimum value that this variable can attain is zero and the maximum value is the run time of the controller

Run time: It is calculated as 80% of the 'watering time'

Pulse time total: It is calculated from 20% of the 'Total watering time' and is added by the difference between 'Active time' and 'Run time'

Pause time: The length of time for which the controller has been running for the pulse. It keeps a track of the total time of the pulse

After the output from the Machine Learning algorithm is obtained, the field deployment of the entire system is done for the outputs of the ML algorithm: 0 and 1.

Sub-routine: Algorithm for pausing and pulsing of the sprinkler system

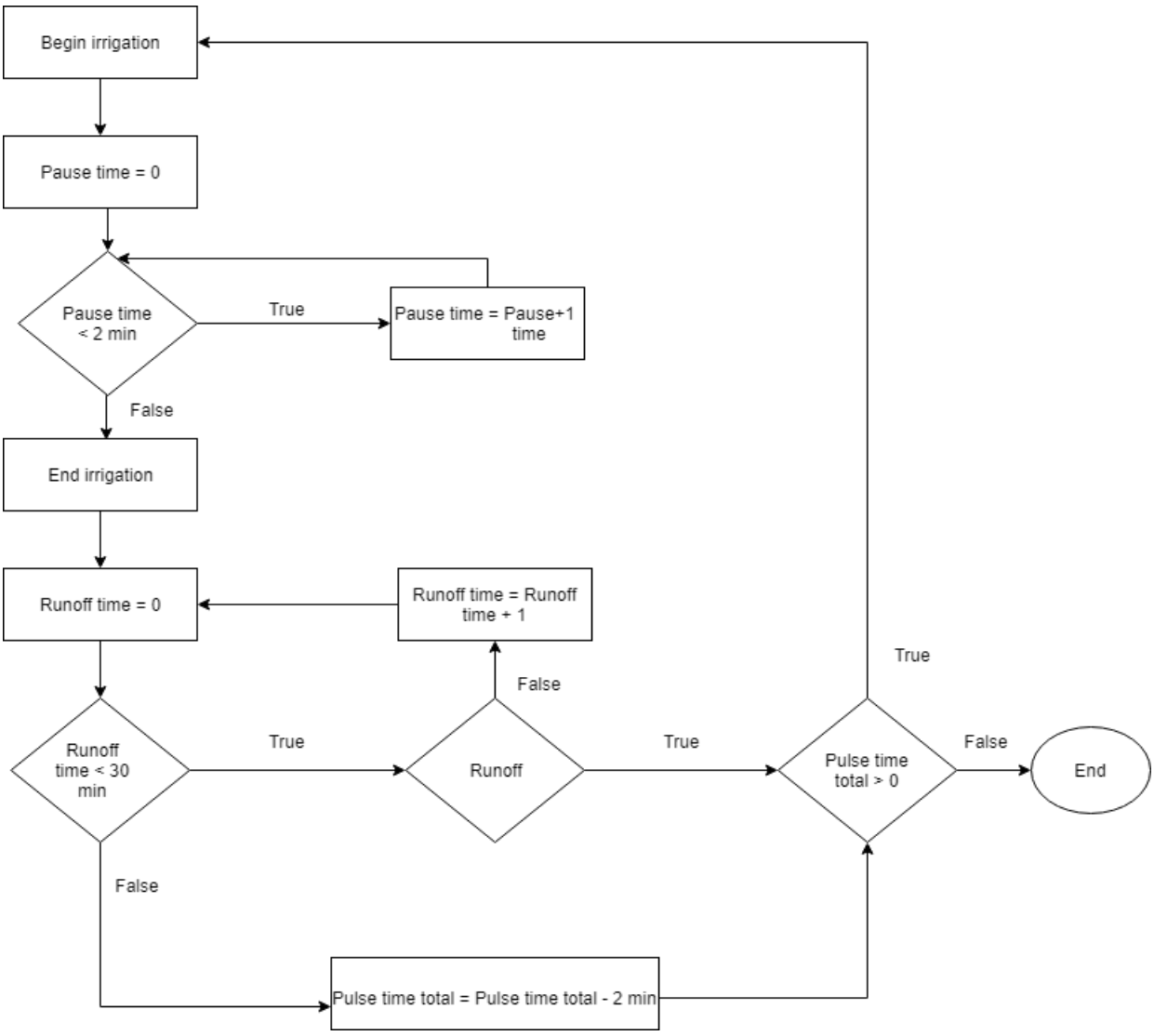

Figure 4.6: Schematic representation for pausing and pulsing for the sprinkler system used in the ASIS controller

At first, the algorithm for pausing and pulsing of the sprinkler system is explained in detail which is a sub-routine for both the cases, as follows.

(i) The irrigation system is started.

(ii) The ‘Pause time’ variable is initialized to zero.

(iii) If the value of ‘Pause time’ variable is less than 2 minutes, then the ‘Pause time’ variable is incremented by 1. This step is repeated until stopping criterion is met (i.e. the value of ‘Pause time’ variable exceeds 2 minutes).

(iv) When the value of ‘Pause time’ variable is more than 2 minutes,

- The irrigation cycle is stopped.
- The ‘Runoff time’ is initialized to zero.
- If the value of ‘Runoff time’ is less than 30 minutes,
    - The physical occurrence of runoff is checked.
    - If runoff is detected, then the algorithm checks for ‘Total pulse time’. If the value of ‘Total pulse time’ is positive, step (i) is initiated, else the algorithm is terminated.
    - If no runoff is detected, the value of ‘Runoff time’ is incremented by 1 and the same variable is initialized to zero. The value of ‘Runoff time’ is again checked until it meets the stopping criterion.
- If the value of ‘Runoff time’ is more than 30 minutes,
    - The ‘Total pulse time’ is decremented by 2 minutes.
    - If the value of ‘Total pulse time’ is positive, step (i) is initiated else the algorithm is terminated.

If the output from the ML algorithm is 1:

(i) At first, the algorithm for calculation of Evapotranspiration and Effective Rainfall is executed after initialization of the algorithm.

(ii) The ‘Total watering time’ is calculated from the cumulative Evapotranspiration rate by using the formula:

Total watering time = sum_et * 35 * 60

(iii) If the ML output is 1,

- The ‘Run time’ is calculated to be 80% of the ‘watering time’.
- The irrigation cycle is started.
- The ‘Active time’ variable is initialized to zero.
- The algorithm for comparison of ‘Active time’ and ‘Run time’ is executed.

Sub-branch 4.1.1: If Active time > Run time:

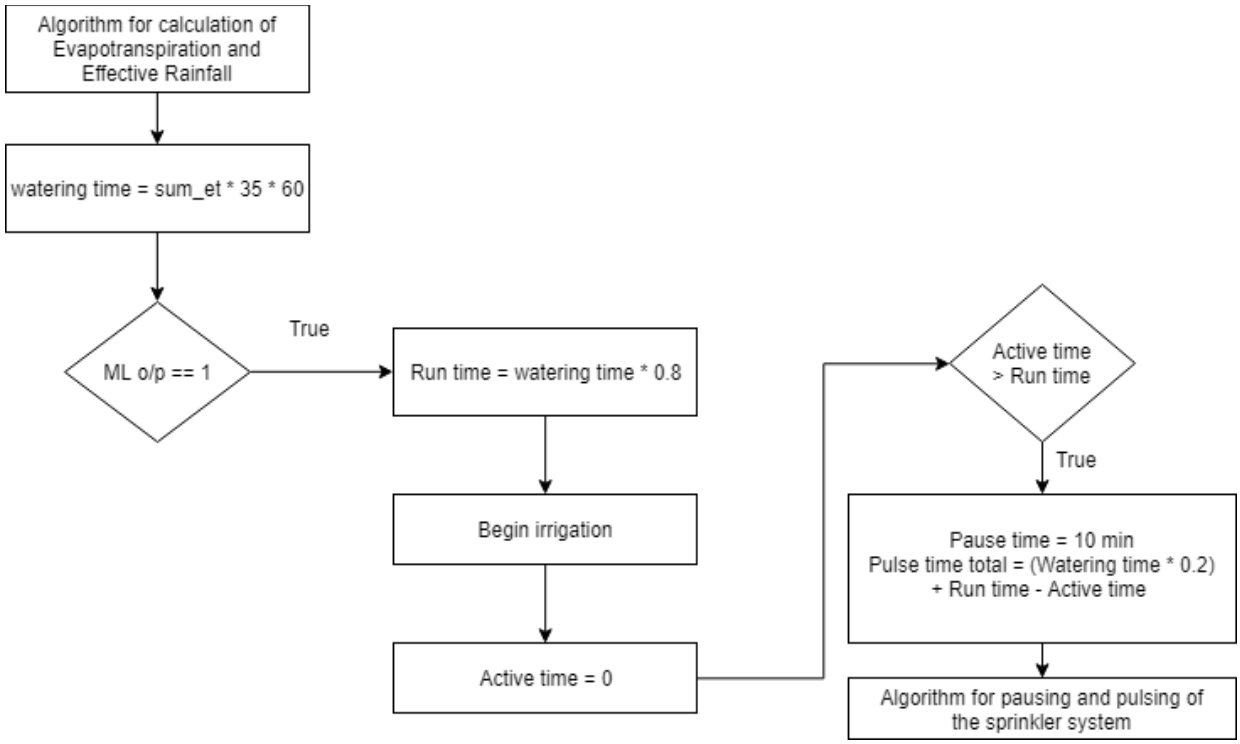

Figure 4.7: Schematic representation when the output from the ML algorithm is 1 (Active time > Run time) used in the ASIS controller

If ‘Active time’ is greater than ‘Run time’,

- The ‘Pause time’ is scheduled to 10 minutes.

The ‘Total pulse time’ of the system is calculated as:

Total pulse time = [20% of the watering time] + Run time + Active time

- Then, the algorithm for pausing and pulsing of the sprinkler system is executed which has been discussed before.

Sub-branch 4.1.2: If Active time < Run time:

If ‘Active time’ is less than ‘Run time’,

- It checks whether the runoff sensor is active.
- If runoff is detected,
  - The ‘pause time’ is set to 10 minutes.
  - The ‘Total pulse time’ of the system is calculated as:

    Total pulse time = [20% of the watering time] + Run time + Active time
  - Then, the algorithm for pausing and pulsing of the sprinkler system is executed which has been discussed before (Section 4.1.3(d)).

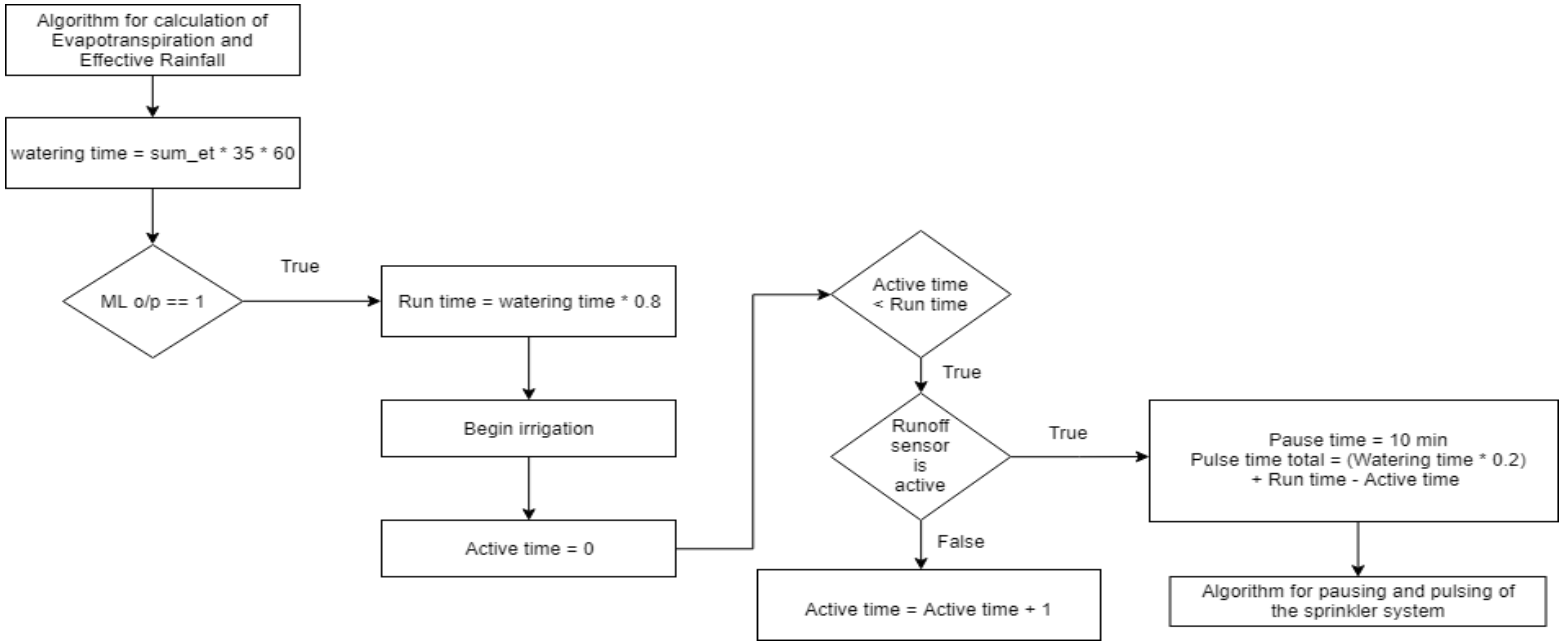

Figure 4.8: Schematic representation when the output from the ML algorithm is 1 (Active time < Run time) used in the ASIS controller

If the output from the ML algorithm is 0:

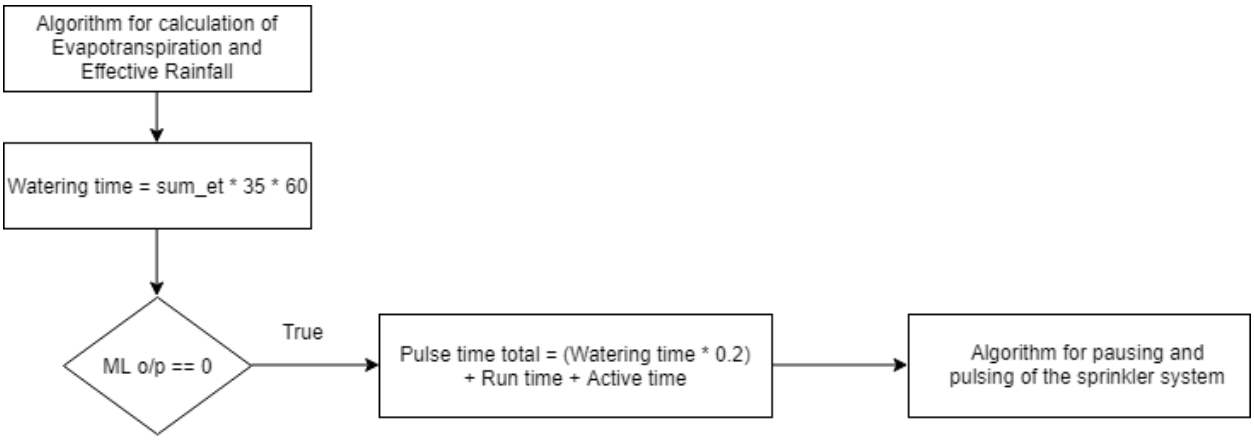

Figure 4.9: Schematic representation when the output from the ML algorithm is 0 used in the ASIS controller

- At first, the algorithm for calculation of Evapotranspiration and Effective Rainfall is executed.
- The 'watering time' is calculated from the cumulative Evapotranspiration rate in the following manner:

  watering time = sum_et * 35 * 60
- If the output from the ML algorithm is zero,

  the 'Total pulse time' of the system is calculated as:

  Total pulse time = [20% of the watering time] + run time + active time

  Then, the algorithm for pausing and pulsing of the sprinkler system is executed which has been discussed before (Section 4.1.3(d)).

4.2 Analysis of performance of controllers by observing their runoff profile

The irrigation runs for all the three controllers have been studied in detail from the observations recorded at the TAMU Turfgrass Laboratory. A detailed comparison of the efficiency of the controllers was made according to the runoff volume observed in all the three cases. Proper justification is given on how the proposed algorithm implemented in the ASIS controller helped in reducing runoff and would prove better than the existing systems in use.

4.2.1 Runoff profile for Rachio controller

Table 4.1: Irrigation profile of the runs carried out by Rachio controller

Date	Start Time	Start Time 2	Start Time 3	Run time	Gallons used	Runoff (Gal)
7/5/19	12:30 AM	2:36 AM	4:46 AM	35	218.81	32.40
7/8/19	12:30 AM	2:36 AM	4:46 AM	35	218.81	38.01
7/15/19	5:48 AM	NAN	NAN	35	218.81	10.34
7/22/19	5:12 AM	NAN	NAN	35	218.81	29.36
7/25/19	5:04 AM	5:44 AM	6:24 AM	30	187.55	NAN
7/26/19	3:20 AM	3:49 AM	4:35 AM	45	281.32	NAN
7/28/19	3:05 AM	3:50 AM	4:35 AM	45	281.32	NAN
8/1/19	5:08 AM	5:52 AM	NAN	28	175.04	5.93
8/8/19	5:12 AM	5:53 AM	6:35 AM	36	225.06	73.01
8/14/19	5:16 AM	5:57 AM	6:39 AM	36	225.06	25.59
8/21/19	5:20 AM	6:01 AM	6:43 AM	35	218.81	48.00
8/28/19	5:24 AM	6:05 AM	6:46 AM	33	206.30	44.77

The irrigation cycles using Rachio controller were carried out in Plot 10 at the TAMU Turfgrass Laboratory. For each of the dates when the irrigation cycles were run, it was observed that the Rachio started the first irrigation cycle at a time typically between midnight and early morning.

For most of the cases observed, Rachio ran a second and third irrigation cycle at irregularly spaced intervals. The cumulative run time of the controller ranged between 30 to 45 minutes.

For 30 to 35 minutes cumulative run time, it was observed that the runoff varied considerably. On 7/5, 7/8, 7/22 and 8/14, the runoff observed varied between 25 to 40 Gallons on each of the plots. For each of these dates, Rachio picked up the first start time either at midnight or in the earliest hours of the morning. On 7/15 and 8/1, when the value of runoff varied between 5 to 10 Gallons, it was observed that the first start time picked up by Rachio was between 5 AM to 6 AM.

Baring 7/25, 7/26 and 7/28, where there were missing values of runoff, the values of precipitation were scrapped from the GRIDMET website to draw inferences about how the controller decided the run time and the start time of operation. The precipitation data on each of the dates either showed zero or extremely low values. Therefore, it was difficult to find any concrete evidence to justify how the controller chose to operate.

4.2.2 Runoff profile for B-hyve controller

Table 4.2: Irrigation profile of the runs carried out by Bhyve controller

Date	Time 1	Time 2	Run Time	Gallons Used	Runoff (Gal)
6/21/19	6:00 AM	6:30 AM	30 mins	187.55	11.67
6/24/19	6:00 AM	6:30 AM	30 mins	187.55	5.20
6/28/19	6:00 AM	6:30 AM	30 mins	187.55	41.12
7/13/19	6:00 AM	6:30 AM	30 mins	187.55	11.03
7/15/19	6:00 AM	6:30 AM	30 mins	187.55	3.77
7/22/19	6:00 AM	6:30 AM	30 mins	187.55	7.83
7/25/19	6:00 AM	6:30 AM	30 mins	187.55	1.62
7/29/19	6:00 AM	6:30 AM	30 mins	187.55	12.49
8/1/19	6:00 AM	6:30 AM	30 mins	187.55	9.64
8/8/19	6:00 AM	6:30 AM	30 mins	187.55	27.27
8/14/19	6:00 AM	6:30 AM	30 mins	187.55	0.98
8/21/19	6:00 AM	6:30 AM	30 mins	187.55	5.93
8/28/19	5:59 AM	6:01 AM	1 min 2 sec	6.46	0.00

The irrigation cycles using B-hyve controller were carried out in Plot 13 at the TAMU Turfgrass Laboratory. For each of the dates when the irrigation cycles were run, it was observed that the BHyve started its first irrigation cycle at 6:00 AM and the second irrigation cycle at 6:30 AM. The cumulative irrigation run time was 30 minutes for all the cases. Unlike the Rachio controller, the BHyve irrigated at the scheduled time and for a fixed period.

Based on the values of runoff observed, efforts were being made to draw inferences about how the BHyve chose to irrigate by classifying the observations into two categories. The runoff

value of 10 Gallons was chosen as the threshold. The values of precipitation were observed for each day in the table. However, as there was no variance in the start time and the cumulative run time of the controller, there was no concrete evidence to justify how the controller chose to operate.

4.2.3 Runoff profile for ASIS controller

Table 4.3: Irrigation profile of the runs carried out by the proposed ASIS controller

Date	Time	ML Output	Total Time	Pre soil moisture	Run 1 Time	Time until runoff	Pulse Time	Gallons Used	Runoff (gal)	Post Soil Moisture
8/1/19	2:00 AM	1	17 min 31 sec	0.38	14 min 1 sec	9 min 1 sec	8 mins 31 seconds	109.50	4.79	0.36
8/8/19	2:00 AM	1	34 min 15 sec	0.2	27 min 24 sec	10 min 43 sec	23 min 32 sec	214.12	None recorded	0.2
8/14/19	7:00 PM	1	32 min 42 sec	0.18	26 min 9 sec	9 min 56 sec	22 min 46 sec	204.43	6.94	0.25
8/21/19	12:00 AM	1	39 min 57 sec	0.29	31 min 57 sec	31 min 57 sec	7 min 59 sec	249.75	26.6	0.36
8/28/19	12:00 AM	1	34 min 20 sec	0.09	27 min 28 sec	9 min 1 sec	25 min 19 sec	214.62	None recorded	0.38

The irrigation cycles using the devised ASIS controller were carried out in Plot 12 at the TAMU Turfgrass Laboratory. Details about how the start time, total run time and pulse time was decided has been discussed in Section 4.1.3. A proper explanation of the Machine Learning part has been discussed in Chapter 3 of the thesis and its usefulness in reducing runoff is explained in the next section.

Except for the irrigation run on 08/21 (the observation marked in red), when the runoff sensor was clogged, all other observations showed extremely low runoff on the plots and there have been two instances where the goal of obtaining zero runoff was achieved.

In summary, a conclusion can be drawn by observing the runoff profile of the three controllers. The devised ASIS controller performs better than the Rachio and BHyve controllers in terms of

minimizing runoff from the irrigational plots. Although more observations would be required to assert the proposed model, it is clear that the proposed algorithm is based on a more sophisticated approach than the existing ones since it considers the soil moisture content of the soil in supporting the decision-making process about watering the turfgrasses.

4.3 Analysis of efficiency of controllers by observing the percentage of Green Cover (GC) in turfgrasses

Although the main focus of the work is to minimize runoff from the plots, maintaining the quality of turfgrasses is also equally important. The quality of turfgrasses is measured by the percentage of Green Cover in these plots. To qualify the efficiency of the controllers, images were taken before and after each irrigation cycle for these plots and the percentage of Green Cover was recorded on each of the days to ensure that the quality of turfgrasses was also maintained.

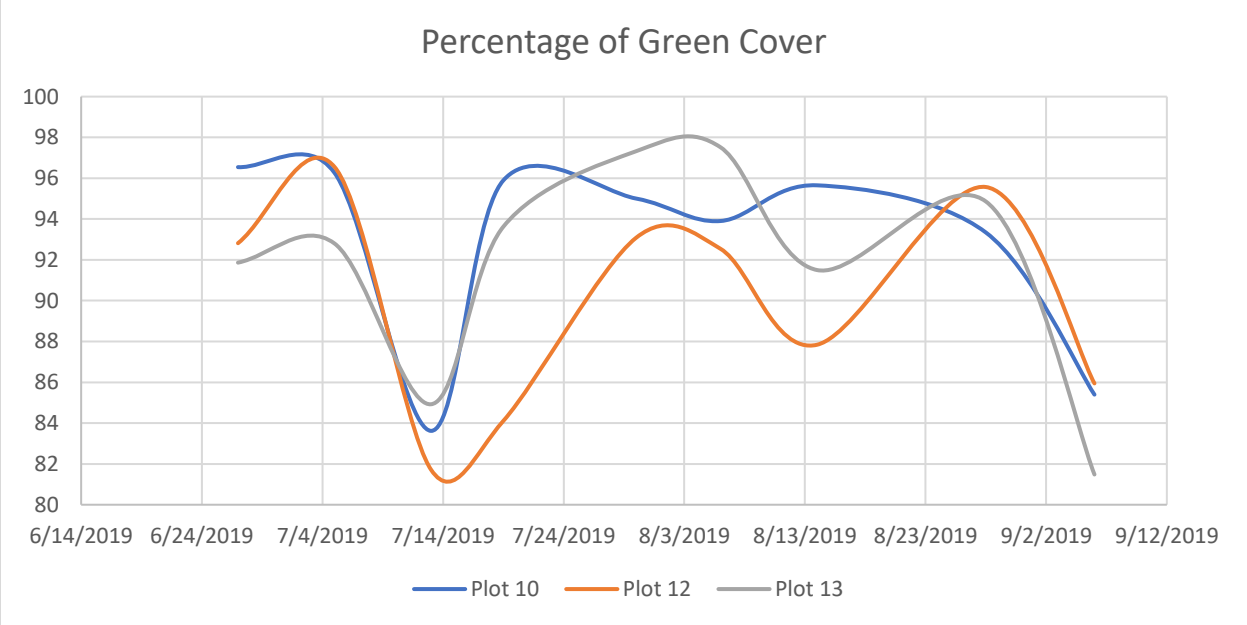

Figure 4.10: Plot showing percentage of Green Cover over the irrigation season

Figure 4.10 gives a clear view of how the percentage of Green Cover changed over the course of the irrigation season. When the irrigation cycle was started, it was decided to irrigate once a week but over the due course of time, it was noticed that the percentage of Green Cover dwindled significantly. Thus, it was decided to increase the frequency of the irrigation cycles thereby maintaining the turfgrass quality.

4.3.1 Evaluation of quality of turfgrasses irrigated by Rachio controller

The turfgrass in Plot 10 was irrigated by the Rachio controller in TAMU Turfgrass Lab, College Station. The first irrigation cycle was carried out on 07/05 and the last irrigation cycle was recorded on 08/28 as shown in Table 4.1. The images of the plots on each of these dates have been shown below to validate the efficiency of the Rachio controller.

Figure 4.11: Images showing the turfgrass quality of Plot 10 irrigated by Rachio controller (Left image: clicked on 07/05; Right image: clicked on 08/28)

The percentage of Green Cover on each of these dates was measured by Reagan Hejl, a technician in TAMU Turfgrass Laboratory using Sigmascan, Turf Analysis Macro software. As of 07/05, the percentage of green cover on the plots was measured to be 83.61% and that on 08/28 was 93.34%.

4.3.2 Evaluation of quality of turfgrasses irrigated by B-hyve controller

The turfgrass in Plot 13 was irrigated by the B-hyve controller in TAMU Turfgrass Lab, College Station. The first irrigation cycle was carried out on 06/21 and the last irrigation cycle was recorded on 08/28 as shown in Table 4.2. The images of the plots on each of these dates have been shown below to validate the efficiency of the B-hyve controller.

Figure 4.12: Images showing the turfgrass quality of Plot 13 irrigated by B-hyve controller (Left image: clicked on 06/21; Right image: clicked on 08/28)

As of 06/21, the percentage of green cover on the plots was measured to be 84.92% and that on 08/28 was 94.86%.

4.3.3 Evaluation of quality of turfgrasses irrigated by the proposed ASIS controller

The turfgrass in Plot 12 was irrigated by the proposed ASIS controller in TAMU Turfgrass Lab, College Station. The first irrigation cycle was carried out on 08/01 and the last irrigation cycle was recorded on 08/28 as shown in Table 4.3. The images of the plots on each of these dates have been shown below to validate the efficiency of the proposed ASIS controller.

Figure 4.13: Images showing the turfgrass quality of Plot 12 irrigated by the proposed ASIS controller (Left image: clicked on 08/01; Right image: clicked on 08/28)

As of 08/01, the percentage of green cover on the plots was measured to be 81.72% and that on 08/28 was 95.57%.

In summary, all the three controllers helped in improving the quality of turfgrasses as measured by the percentage of Green Cover on the plots. However, when the runoff profile for all the three controllers are compared, the proposed ASIS controller seemed to perform the best.

4.4 Impact of the ML algorithm in minimizing runoff

The devised controller in this case was the first to use predictive approaches for minimizing runoff. After training the Radial Basis Function-Support Vector Machine classifier (the proposed ML approach) on the synthetic dataset generated from the previously recorded data, it was decided that predicting the Soil Wetting Efficiency Index was the best approach. This target variable was chosen after measuring the testing accuracy which was approximately about 89% (the details about

choosing the target variable has already been discussed in Chapter 3 of the thesis). This is the most generic approach to be used since it is not site-specific.

The ASIS controller uses the Machine Learning approach as a Decision Support System (DSS) while watering the turfgrasses. The ML approach here does not prescribe the amount of water that the plots need, rather it prescribes the frequency at which the plots need to be irrigated.

The frequency of irrigation is one of the most essential components that needs to be kept in mind to minimize runoff. One of the major reasons for attaining high runoff is that soils become saturated after irrigating continuously for long duration of time. Therefore, it is very important to irrigate plots at time intervals depending on the soil moisture content of the soil. The ML algorithm used in this case classified the observations into either of the two categories (SWEI of 1.7 is set as threshold). The predicted values of SWEI less than 1.7 showed higher potential for runoff and vice-versa. A set of irrigation rules based on the ML output have already been defined in Chapter 3 (Section 3.5). It is expected that with increasing amount of observations, the value of threshold may change and in due process of time, a more robust approach would be in place which may nullify runoff entirely.

CHAPTER V

CONCLUSION AND FUTURE WORK

5.1 Conclusion

This work evaluates the performance of irrigiation controllers that attempt to minimize the water usage in turfgrass irrigation. The devised ASIS controller used with an in-built Machine Learning algorithm was tested and evaluated to show that it had the potential to nullify runoff from the plots. The tests were performed at the Texas A&M Turfgrass Research Laboratory, which is well equipped to support the observations. To further quantify and qualify the approach, the Rachio and BHyve controllers were installed in the adjacent plots and the runoff from the plots were recorded.

Since the Machine Learning algorithm acted as a Decision Support System (DSS), it prescribed the frequency of irrigating plots rather than the amount of water that would be required for irrigation. The ML algorithm was trained on the synthetic data generated from the previous field experiments carried out in the laboratory. The accuracy of the Radial Basis Function - Support Vector Machine tested on the recorded dataset was around 90%. Only few irrigation cycles were scheduled so far and experimental data are still being collected from the facility. Preliminary results show that the proposed irrigation controller with a built-in Machine Learning approach which uses Soil Wetting Efficiency Index as the target variable has the capacity to minimize the runoff in these plots. With more irrigation cycles which would be scheduled over time, the proposed Machine Learning approach would perform better with more datapoints and may nullify runoff eventually.

5.2 Future work

Based on the preliminary results obtained from limited number of tests, it can be validated that the system has the potential to save water. However, more tests are required to quantify the overall water savings. The testing is scheduled to continue for the next few months. The entire LIRMS system is planned to be implemented in Round Rock, Texas to gather more data and to see how well the system performs while irrigating in rocky terrains.

The data on which the ML model was trained had been collected during a moderately wet season and for sandy-loamy soil. The characteristics of the plots on which the Machine Learning model was trained and tested were completely different. The training data was collected from Plots 15, 17 and 18 where the soil series used was 'Zack' and the top soil cover on each of those plots were 26.3, 39 and 30.5 cm respectively. The model was tested on Plots 10, 12 and 13 where the soil series used was 'Booneville' and the top soil cover ranged between 30 to 45 cm. Therefore, it was difficult for the ML algorithm to perform accurately as the depth of the top soil and the climatic conditions varied substantially which in turn went on to affect the other irrigation parameters too. Another factor which must be considered is that the amount of runoff as well as the Soil Wetting Efficiency Index (SWEI) vary substantially depending on the geospatial conditions where the irrigation cycles are run. So, future work must be focused on constructing a more diverse dataset for training the Machine Learning model with precision.

With increase in the number of datapoints, the distribution of the data can be gauged and used with more certainty. The use of Bayesian techniques can be considered for making decisions as they are more robust and reliable compared to traditional Machine Learning approaches used in the study.

REFERENCES

[1] Xie, J., Chen, H., Liao, Z., Gu, X., Zhu, D., & Zhang, J. (2017). An integrated assessment of urban flooding mitigation strategies for robust decision making. *Environmental Modelling & Software*, *95*, 143–155. doi: 10.1016/j.envsoft.2017.06.027

[2] Agam, N., Kustas, W. P., Anderson, M. C., Li, F., & Colaizzi, P. D. (2007). Utility of thermal sharpening over Texas high plains irrigated agricultural fields. *Journal of Geophysical Research,112*(D19). doi:10.1029/2007jd008407

[3] Fereres, E., & Soriano, M. A. (2006). Deficit irrigation for reducing agricultural water use. *Journal of Experimental Botany,58*(2), 147-159. doi:10.1093/jxb/erl165

[4] Kothapalli, U. B. (December 2017). *Field Deployment and Integration of Wireless Communication & Operation Support System for the Landscape Irrigation Runoff Mitigation System*(Unpublished master's thesis). Texas A&M University.

[5] Capraro, F., Patino, D., Tosetti, S., & Schugurensky, C. (2008). Neural Network-Based Irrigation Control for Precision Agriculture. *2008 IEEE International Conference on Networking, Sensing and Control*. doi: 10.1109/icnsc.2008.4525240

[6] Karasekreter, N., Başçiftçi, F., & Fidan, U. (2013). A new suggestion for an irrigation schedule with an artificial neural network. *Journal of Experimental & Theoretical Artificial Intelligence,25*(1), 93-104. doi:10.1080/0952813x.2012.680071

[7] Mahmood A Khan, Md Zahidul Islam & Mohsin Hafeez (2013). Irrigation Water Requirement Prediction through Various Data Mining Techniques Applied on a Carefully Pre-processed Dataset. Journal of Research and Practice In Information Technology, pp. 1-13

[8] S Muhammad Umair & R. Usman (2010). Automation of Irrigation System Using ANN based Controller. International Journal of Electrical & Computer Sciences, Vol. 10, No. 2, pp. 41- 47

[9] Abdullah Gokhan Yilmaz & Nittin Muttil (2014). Runoff Estimation by Machine Learning Methods and Application to the Euphrates Basin in Turkey. Journal of Hydrologic Engineering, pp. 1015-1025

[10] Gude, V. G. (2017). Desalination and water reuse to address global water scarcity. *Reviews in Environmental Science and Bio/Technology,16*(4), 591-609. doi:10.1007/s11157-017-9449-7

[11] Bunn, S. E. (2016). Grand Challenge for the Future of Freshwater Ecosystems. *Frontiers in Environmental Science,4*. doi:10.3389/fenvs.2016.00021

[12] Willis, G. H., & Mcdowell, L. L. (1982). Review: Pesticides In Agricultural Runoff And Their Effects On Downstream Water Quality. *Environmental Toxicology and Chemistry,1*(4), 267. doi:10.1897/1552-8618

[13] Xie, J., Chen, H., Liao, Z., Gu, X., Zhu, D., & Zhang, J. (2017). An integrated assessment of urban flooding mitigation strategies for robust decision making. *Environmental Modelling & Software,95,* 143-155. doi:10.1016/j.envsoft.2017.06.027

[14] Gaborit, E., Anctil, F., Pelletier, G., & Vanrolleghem, P. (2015). Exploring forecast-based management strategies for stormwater detention ponds. *Urban Water Journal,13*(8), 841-851. doi:10.1080/1573062x.2015.1057172

[15] Tang, X., Zhu, B., & Katou, H. (2012). A review of rapid transport of pesticides from sloping farmland to surface waters: Processes and mitigation strategies. *Journal of Environmental Sciences,24*(3), 351-361. doi:10.1016/s1001-0742(11)60753-5

[16] Davis, A. P., Hunt, W. F., Traver, R. G., & Clar, M. (2009). Bioretention Technology: Overview of Current Practice and Future Needs. *Journal of Environmental Engineering,135*(3), 109-117. doi:10.1061/(asce)0733-9372(2009)135:3(109)

[17] Vymazal, J., & Březinová, T. (2015). The use of constructed wetlands for removal of pesticides from agricultural runoff and drainage: A review. *Environment International,75*, 11-20. doi:10.1016/j.envint.2014.10.026

[18] Troutman, B. M. (1985). Errors and Parameter Estimation in Precipitation-Runoff Modeling: 1. Theory. *Water Resources Research,21*(8), 1195-1213. doi:10.1029/wr021i008p01195

[19] Schafer, J. L., & Olsen, M. K. (1998). Multiple Imputation for Multivariate Missing-Data Problems: A Data Analysts Perspective. *Multivariate Behavioral Research,33*(4), 545-571. doi:10.1207/s15327906mbr3304_5

[20] Therese D Pigott (2001). A Review of Methods for Missing Data, Educational Research and Evaluation, 7:4, 353-383, DOI: 10.1076/edre.7.4.353.8937

[21] Tang, J., Liu, J., Zhang, M., & Mei, Q. (2016). Visualizing Large-scale and High-dimensional Data. *Proceedings of the 25th International Conference on World Wide Web - WWW 16*. doi:10.1145/2872427.2883041

[22] Pickett, R., & Grinstein, G. (n.d.). Iconographic Displays For Visualizing Multidimensional Data. *Proceedings of the 1988 IEEE International Conference on Systems, Man, and Cybernetics*. doi:10.1109/icsmc.1988.754351

[23] Keim, D. (2002). Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics,8*(1), 1-8. doi:10.1109/2945.981847

[24] Andreas Buja, Deborah F Swayne, Michael L Littman, Nathaniel Dean, Heike Hofmann & Lisha Chen (2008) Data Visualization With Multidimensional Scaling, *Journal of Computational and Graphical Statistics*, 17:2, 444-472, DOI: 10.1198/106186008X318440

[25] Chi, E. (n.d.). A taxonomy of visualization techniques using the data state reference model. *IEEE Symposium on Information Visualization 2000. INFOVIS 2000. Proceedings*. doi:10.1109/infvis.2000.885092

[26] Lei Yu and Huan Liu (2003). Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution. Proceedings of Twentieth International Conference on Machine Learning.

[27] Jaworska, N., & Chupetlovska-Anastasova, A. (2009). A Review of Multidimensional Scaling (MDS) and its Utility in Various Psychological Domains. *Tutorials in Quantitative Methods for Psychology,5*(1), 1-10. doi:10.20982/tqmp.05.1.p001

[28] Park, M. S., Na, J. H., & Choi, J. Y. (n.d.). PCA-based Feature Extraction using Class Information. *2005 IEEE International Conference on Systems, Man and Cybernetics*. doi:10.1109/icsmc.2005.1571169

[29] Dyk, D. A., & Meng, X. (2001). The Art of Data Augmentation. *Journal of Computational and Graphical Statistics,10*(1), 1-50. doi:10.1198/10618600152418584

[30] Wong, S. C., Gatt, A., Stamatescu, V., & Mcdonnell, M. D. (2016). Understanding Data Augmentation for Classification: When to Warp? *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*. doi:10.1109/dicta.2016.7797091

[31] Wei, G. C., & Tanner, M. A. (1991). Applications of Multiple Imputation to the Analysis of Censored Regression Data. *Biometrics,47*(4), 1297. doi:10.2307/2532387

[32] Efron, B. (1994). Missing Data, Imputation, and the Bootstrap. *Journal of the American Statistical Association,89*(426), 463. doi:10.2307/2290846

[33] Wherley, B. (2018). Method and System for Reduction of Irrigation Runoff. US9955636132

[34] Wherley, B. (2018). Effect of Sodic Irrigation Water on Organic Carbon and Nitrogen Concentrations, Fluxes and Exports from Newly Installed St. Augustine Grass Sod in South-Central Texas. *Journal of Horticulture Vol. 5 Issue 2* pp. 1-7

APPENDIX A

ML code when the target variable "runoff observed" is binned into two classes

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

data = pd.read_csv('C:/Users/samba/Desktop/ASIS_new.csv') #Importing the dataset

print('Training data shape:', data.shape) #Initial volume of the dataset (31 observations and
                                          32 observations)

#Categorizing the start time variable which we would be using in our analysis
#Any time between 9 pm and 4 am is categorized as Night time denoted by 1, any time
between 4 am and 12 pm is categorized as Morning time denoted by 2, any time between
12 pm and 4 pm is categorized as Afternoon time denoted by 3 and any time between 4 pm
and 9 pm is categorized as Evening time denoted by 4
        hour1= []
        rey = list(data['START_TIME_hr:min'].values)
        for i,j in enumerate(rey):
           hrs,mins = j.split(':')
           hrs = np.asarray(hrs)
           hrs = hrs.astype(int)
           for i in np.nditer(hrs):
              if i>21 and i<=23:
                 hour1.append(1)
              elif i>=0 and i<4:                    #Night
                 hour1.append(1)
              elif i>=4 and i<=12:                  #Morning
                 hour1.append(2)
              elif i>12 and i<=16:                  #Afternoon
                 hour1.append(3)
              elif i>16 and i<=21:                   #Evening
                 hour1.append(4)
        df1 = pd.DataFrame(hour1,columns=['START_TIME_CLASS'])
        result = pd.concat([data, df1], axis=1, join='inner')
        result = result.drop(['START_TIME_hr:min'], axis = 1)
        result

#Final list of predictors to be used in the analysis
        Xcols=['ETO_mm/day','SCHEDULED_IRRIGATION_TIME_mins','AVG
        WIND__SPEED_MPH','PRECIPITATION_inch','START_TIME_CLASS',
        'RUNOFF_VOLUME_OBSERVED_Gal']

        data3 = result[Xcols]
```

```
data3.to_csv('ASIS3.csv', index = False) #Saving the new generated encoded data
                                          to a new file to use in our algorithm

#There are 4 observations which have missing values of runoff. As the number of
observations in the dataset are too less to drop any of them from the dataset, it was decided
to impute the values of runoff by running a Linear Regression over rest of the predictors
    data = pd.read_csv('C:/Users/samba/Desktop/ASIS3.csv') #Importing the dataset
    df1 = data[data.isnull().any(axis=1)]
    x_test = df1.drop(['RUNOFF_VOLUME_OBSERVED_Gal'], axis = 1)
    x = data.dropna()
    x_train = x.drop(['RUNOFF_VOLUME_OBSERVED_Gal'], axis = 1)
    y_train = x['RUNOFF_VOLUME_OBSERVED_Gal']
    data = data.fillna(0) #Replacing the missing values of runoff observed with zero
                           before imputing them
    from sklearn.linear_model import LinearRegression
    regressor = LinearRegression()
    reg = regressor.fit(x_train, y_train) #Fitting a Linear Regression model over the
                                           training data
    y_test = reg.predict(x_test) #Predicting the missing values of runoff observed
    data4 = data['RUNOFF_VOLUME_OBSERVED_Gal']
    re = data4.as_matrix(columns=None)

    j=0
    for i, l in enumerate(re):
       if l ==0 :
          re[i] = y_test[j]
          j+=1

    df3 = pd.DataFrame(re,columns=['RUNOFF_VOLUME_OBSERVED_Gal'])
    data = data.drop(['RUNOFF_VOLUME_OBSERVED_Gal'], axis = 1)
    result = pd.concat([data, df3], axis=1, join='inner') #Storing the predicted values of
                                                           runoff in one dataframe

#Visualizing the runoff data
    from scipy.stats import norm
    x_d = np.linspace(-4, 8, 1000)
    re1 = result['RUNOFF_VOLUME_OBSERVED_Gal'].as_matrix(columns=None)
    from sklearn.neighbors import KernelDensity
    kde = KernelDensity(bandwidth=1.0, kernel='gaussian') #Instantiating and fitting
                                                           the KDE model
    kde.fit(re1[:, None])
    logprob = kde.score_samples(x_d[:, None]) #score_samples returns the log of the
                                               probability density

    plt.fill_between(x_d, np.exp(logprob), alpha=0.5)
```

```
frame1 = plt.gca()
frame1.axes.get_yaxis().set_visible(False)
plt.plot(re1, np.full_like(re1, -0.01), '|k', markeredgewidth=1)
plt.ylim(-0.02, 0.22)
plt.xlabel('Values of runoff data in Gallons')
```

#Categorizing the value of runoff based on its distribution. From the data visualization, the value of runoff volume below 105 Gallons was classified as Class 0 and any value over it was classified as Class 1 for analysis

```
re2 = []
for i,l in enumerate(re):
   if l < 105 :
      re2.append(0)
   else :
      re2.append(1)
df4 =
pd.DataFrame(re2,columns=['RUNOFF_VOLUME_OBSERVED_Gal_CLASS'])
result1 = pd.concat([result, df4], axis=1, join='inner')
result1 = result1.drop(['RUNOFF_VOLUME_OBSERVED_Gal'], axis = 1)
```

#Normalizing the features and constructing a new Dataframe by appending the target variable to be used for further analysis

```
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
re2 = result.as_matrix(columns = None)
X_scaled = min_max_scaler.fit_transform(re2[:,0:5])
X_cols3 = ['START_TIME_CLASS']
result2 = result1[X_cols3]
X_cols4 = ['RUNOFF_VOLUME_OBSERVED_Gal_CLASS']
result3 = result1[X_cols4]
#Constructing a new dataframe for the normalized features
dat1 = pd.DataFrame({'ETO_mm/day':X_scaled[:,0]})
dat2 = pd.DataFrame({'AVG_WIND_SPEED_MPH':X_scaled[:,2]})
dat4 = pd.DataFrame({'PRECIPITATION_inch':X_scaled[:,3]})
dat7 = pd.DataFrame({'EFFECTIVE_IRRIGATION_TIME_min':X_scaled[:,1]})
dataset = pd.concat([dat1, dat2, dat4, dat7, result2, result3], axis=1)
```

#Removing the target variable before running Principal Component Analysis on the dataset

```
y_frame = dataset.iloc[:,-1]
y_data = y_frame.as_matrix(columns = None)
dataset = dataset.drop(['RUNOFF_VOLUME_OBSERVED_Gal_CLASS'], axis =
1)
```

#Principal Component Analysis on the dataset to get a clear idea about the variance explained

```
from sklearn.decomposition import PCA
X = dataset.as_matrix(columns = None)
pca = PCA(n_components=3)
pca.fit(X)
x_data_pca = pca.fit_transform(X)
var1 = np.cumsum(np.round(pca.explained_variance_ratio_, decimals=2)*100)
print(pca.explained_variance_ratio_)
plt.xlabel("Number of components")
plt.ylabel("Cumulative Proportion of the variance explained")
plt.plot(var1)
print(var1)
data8 = pd.DataFrame(pca.components_,columns=dataset.columns,index = ['PC-1','PC-2','PC-3'])
```

#Synthetic Data generation for observations with runoff volume less than 105 Gallons and irrigation during night hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={"RUNOFF_VOLUME_OBSERVED_Gal_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 0, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 1, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=500, n_features=4, n_classes=1, random_state=1)
Synthetic_data_0=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SPEED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_0=Synthetic_data_0.assign(**{'START_TIME_CLASS': 1,'RUNOFF_VOLUME_OBSERVED_Gal_CLASS': 0})
```

#Synthetic Data generation for observations with runoff volume less than 105 Gallons and irrigation during morning hours (Distinct mean and covariance matrix between the classes)

```
dataset1=dataset.rename(index=str,columns={"RUNOFF_VOLUME_OBSERVED_Gal_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 0, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
```

```
dataset4 = dataset3.loc[lambda df: df.C == 2, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Construct dataset
X1, y1 = make_gaussian_quantiles(cov=cov_xe, mean=mean_xe,
                    n_samples=500, n_features=4,
                    n_classes=1, random_state=1)
Synthetic_data_1=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_1=Synthetic_data_1.assign(**{'START_TIME_CLASS':2,'RUN
OFF_VOLUME_OBSERVED_Gal_CLASS': 0})
```

#Synthetic Data generation for observations with runoff volume less than 105 Gallons and irrigation during afternoon hours (Distinct mean and covariance matrix between the classes)

```
dataset1=dataset.rename(index=str,columns={"RUNOFF_VOLUME_OBSERVE
D_Gal_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 0, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 3, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Construct dataset
X1, y1 = make_gaussian_quantiles(cov=cov_xe, mean=mean_xe,
                    n_samples=500, n_features=4,
                    n_classes=1, random_state=1)
Synthetic_data_2=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch',‘EFFECTIVE_IRRIGATION_TIME_min']
)
Synthetic_data_2=Synthetic_data_2.assign(**{'START_TIME_CLASS':3,'RUN
OFF_VOLUME_OBSERVED_Gal_CLASS': 0})
```

#Synthetic Data generation for observations with runoff volume less than 105 Gallons and irrigation during evening hours (Distinct mean and covariance matrix between the classes)

```
dataset1=dataset.rename(index=str,columns={"RUNOFF_VOLUME_OBSERVE
D_Gal_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 0, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 4, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
```

```
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
 #Construct dataset
X1, y1 = make_gaussian_quantiles(cov=cov_xe, mean=mean_xe,
                                 n_samples=500, n_features=4,
                                 n_classes=1, random_state=1)
Synthetic_data_3=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_3=Synthetic_data_3.assign(**{'START_TIME_CLASS':4,'RUN
OFF_VOLUME_OBSERVED_Gal_CLASS': 0})
```

#Synthetic Data generation for observations with runoff volume more than 105 Gallons and irrigation during night hours (Distinct mean and covariance matrix between the classes)

```
dataset1=dataset.rename(index=str,columns={"RUNOFF_VOLUME_OBSERVE
D_Gal_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 1, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 1, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
 #Construct dataset
X1, y1 = make_gaussian_quantiles(cov=cov_xe, mean=mean_xe,
                                 n_samples=500, n_features=4,
                                 n_classes=1, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4=Synthetic_data_4.assign(**{'START_TIME_CLASS':1,'RUN
OFF_VOLUME_OBSERVED_Gal_CLASS': 1})
```

#Synthetic Data generation for observations with runoff volume more than 105 Gallons and irrigation during morning hours (Distinct mean and covariance matrix between the classes)

```
dataset1=dataset.rename(index=str,columns={"RUNOFF_VOLUME_OBSERVE
D_Gal_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 1, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 2, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
```

```
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Construct dataset
X1, y1 = make_gaussian_quantiles(cov=cov_xe, mean=mean_xe,
                        n_samples=500, n_features=4,
                        n_classes=1, random_state=1)
Synthetic_data_5=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_5=Synthetic_data_5.assign(**{'START_TIME_CLASS':2,'RUN
OFF_VOLUME_OBSERVED_Gal_CLASS': 1})
```

#Synthetic Data generation for observations with runoff volume more than 105 Gallons and irrigation during evening hours (Distinct mean and covariance matrix between the classes)

```
dataset1=dataset.rename(index=str,columns={"RUNOFF_VOLUME_OBSERVE
D_Gal_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 1, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 4, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
 #Construct dataset
X1, y1 = make_gaussian_quantiles(cov=cov_xe, mean=mean_xe,
                        n_samples=500, n_features=4,
                        n_classes=1, random_state=1)
Synthetic_data_7=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_7=Synthetic_data_7.assign(**{'START_TIME_CLASS':4,'RUN
OFF_VOLUME_OBSERVED_Gal_CLASS': 1})
```

#Combining all the generated data for distinct mean and covariance matrix into a single Data frame

```
result1 = Synthetic_data_1.append(Synthetic_data_0, ignore_index=True)
result2 = result1.append(Synthetic_data_2, ignore_index=True)
result3 = result2.append(Synthetic_data_3, ignore_index=True)
result5 = result3.append(Synthetic_data_5, ignore_index=True)
result8 = result5.append(Synthetic_data_7, ignore_index=True)
```

#Synthetic Data generation for all irrigation events during the night (Shared mean and covariance matrix between classes)

```
dataset1=dataset.rename(index=str,columns={"RUNOFF_VOLUME_OBSERVE
D_Gal_CLASS": "B"})
dataset2 = dataset1.rename(index=str, columns={"START_TIME_CLASS": "C"})
```

```
dataset3 = dataset2.loc[lambda df: df.C == 1, :]
dataset4 = dataset3.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Constructing the dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,mean=mean_xe,n_samples=1000,
n_features=4,n_classes=2, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4 = Synthetic_data_4.assign(**{'START_TIME_CLASS': 1})
Synthetic_data_5=pd.DataFrame(y1,columns=['RUNOFF_VOLUME_OBSERV
ED_Gal_CLASS'])
result_C_12=pd.concat([Synthetic_data_4,Synthetic_data_5],axis=1,join_axes=[S
ynthetic_data_4.index])
```

#Synthetic Data generation for all irrigation events during the morning (Shared mean and covariance matrix between classes)

```
dataset1=dataset.rename(index=str,columns={"RUNOFF_VOLUME_OBSERVE
D_Gal_CLASS": "B"})
dataset2 = dataset1.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset3 = dataset2.loc[lambda df: df.C == 2, :]
dataset4 = dataset3.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Constructing the dataset
X1,y1= make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=1000,
n_features=4, n_classes=2, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch',
'EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4 = Synthetic_data_4.assign(**{'START_TIME_CLASS': 2})
Synthetic_data_5=pd.DataFrame(y1,columns=['RUNOFF_VOLUME_OBSERV
ED_Gal_CLASS'])
result_C_13=pd.concat([Synthetic_data_4,Synthetic_data_5],axis=1,join_axes=[S
ynthetic_data_4.index])
```

#Synthetic Data generation for all irrigation events during the afternoon (Shared mean and covariance matrix between classes)

```
dataset1=dataset.rename(index=str,columns={"RUNOFF_VOLUME_OBSERVE
D_Gal_CLASS": "B"})
dataset2 = dataset1.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset3 = dataset2.loc[lambda df: df.C == 3, :]
```

```
dataset4 = dataset3.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Constructing the dataset
X1,y1= make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=1000,
n_features=4, n_classes=2, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch',
'EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4 = Synthetic_data_4.assign(**{'START_TIME_CLASS': 3})
Synthetic_data_5=pd.DataFrame(y1,columns=['RUNOFF_VOLUME_OBSERV
ED_Gal_CLASS'])
result_C_14=pd.concat([Synthetic_data_4,Synthetic_data_5],axis=1,join_axes=[S
ynthetic_data_4.index])

#Synthetic Data generation for all irrigation events during the evening (Shared mean and
covariance matrix between classes)
dataset1=dataset.rename(index=str,columns={"RUNOFF_VOLUME_OBSERVE
D_Gal_CLASS": "B"})
dataset2 = dataset1.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset3 = dataset2.loc[lambda df: df.C == 4, :]
dataset4 = dataset3.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Constructing the dataset
X1,y1= make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=1000,
n_features=4, n_classes=2, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch',
'EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4 = Synthetic_data_4.assign(**{'START_TIME_CLASS': 4})
Synthetic_data_5=pd.DataFrame(y1,columns=['RUNOFF_VOLUME_OBSERV
ED_Gal_CLASS'])
result_C_15=pd.concat([Synthetic_data_4,Synthetic_data_5],axis=1,join_axes=[S
ynthetic_data_4.index])

#Combining all the generated data into a single Data frame
result5 = result_C_12.append(result_C_13, ignore_index=True)
result6 = result5.append(result_C_14, ignore_index=True)
result7 = result6.append(result_C_15, ignore_index=True)
final_result = result8.append(result7, ignore_index=True)
```

```
#Training RBF SVM for different values of train:test split on the synthetic data
    from sklearn.model_selection import train_test_split
    from sklearn.model_selection import GridSearchCV
    from sklearn.metrics import accuracy_score
    X = final_result.drop('RUNOFF_VOLUME_OBSERVED_Gal_CLASS', axis = 1)
    y = final_result['RUNOFF_VOLUME_OBSERVED_Gal_CLASS']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state
    = 42)
    import numpy as np
    import matplotlib.pyplot as plt
    from matplotlib import style
    style.use("ggplot")
    from sklearn import svm
    b = X_train.as_matrix(columns = None)
    c = y_train.as_matrix(columns = None)
    from sklearn import svm
    # Grid Search
    # Parameter Grid
    gammas = np.linspace(0.0, 1.0, num=100)
    param_grid = {'C': [100], 'gamma': gammas} #The value of penalty parameter is
                                                varied from 1 to 100 in interval of
                                                10 units. Best gamma is chosen for
                                                each case.
    # Make grid search classifier
    clf_grid = GridSearchCV(svm.SVC(), param_grid, verbose=1)
    # Training the classifier
    clf_grid.fit(b,c)
    print("Best Parameters:\n", clf_grid.best_params_)
    print("Best Estimators:\n", clf_grid.best_estimator_)
    b1 = X_test.as_matrix(columns = None)
    c1 = y_test.as_matrix(columns = None)
    result = clf_grid.predict(b1)
    accuracy_score(c1, result) #Gives the accuracy score on testing data

#Training RBF SVM on the entire synthetic data generated and training it on the real
dataset
    from sklearn.model_selection import train_test_split
    from sklearn.model_selection import GridSearchCV
    from sklearn.metrics import accuracy_score
    X_train = final_result.drop('RUNOFF_VOLUME_OBSERVED_Gal_CLASS',
    axis = 1)
    y_train = final_result['RUNOFF_VOLUME_OBSERVED_Gal_CLASS']
    style.use("ggplot")
    from sklearn import svm
    b = X_train.as_matrix(columns = None)
    c = y_train.as_matrix(columns = None)
```

```
X_test = dataset.drop('RUNOFF_VOLUME_OBSERVED_Gal_CLASS', axis = 1)
y_test = dataset['RUNOFF_VOLUME_OBSERVED_Gal_CLASS']
b1 = X_test.as_matrix(columns = None)
c1 = y_test.as_matrix(columns = None)
from sklearn import svm
#Grid Search
#Parameter Grid
gammas = np.linspace(0.0, 1.0, num=100)
param_grid1 = {'C': [100], 'gamma': gammas}
#Make grid search classifier
clf_grid1 = GridSearchCV(svm.SVC(), param_grid1, verbose=1)

#Train the classifier
clf_grid1.fit(b,c)
#clf = grid.best_estimator_()
print("Best Parameters:\n", clf_grid1.best_params_)
print("Best Estimators:\n", clf_grid1.best_estimator_)
result1 = clf_grid1.predict(b1)
accuracy_score(c1, result1)
```

APPENDIX B

ML code when the target variable "Time when the first instance of runoff is observed" is binned into three classes

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

data = pd.read_csv('C:/Users/samba/Desktop/ASIS_new.csv') #Importing the dataset

print('Training data shape:', data.shape) #Initial volume of the dataset (31 observations and
                                          32 predictors)

#Categorizing the start time variable which we would be using in our analysis
#Any time between 9 pm and 4 am is categorized as Night time denoted by 1, any time
between 4 am and 12 pm is categorized as Morning time denoted by 2, any time between
12 pm and 4 pm is categorized as Afternoon time denoted by 3 and any time between 4 pm
and 9 pm is categorized as Evening time denoted by 4
        hour1= []
        rey = list(data['START_TIME_hr:min'].values)
        for i,j in enumerate(rey):
           hrs,mins = j.split(':')
           hrs = np.asarray(hrs)
           hrs = hrs.astype(int)
           for i in np.nditer(hrs):
              if i>21 and i<=23:
                 hour1.append(1)
              elif i>=0 and i<4:                #Night
                 hour1.append(1)
              elif i>=4 and i<=12:              #Morning
                 hour1.append(2)
              elif i>12 and i<=16:              #Afternoon
                 hour1.append(3)
              elif i>16 and i<=21:               #Evening
                 hour1.append(4)
        df1 = pd.DataFrame(hour1,columns=['START_TIME_CLASS'])
        result = pd.concat([data, df1], axis=1, join='inner')
        result = result.drop(['START_TIME_hr:min'], axis = 1)
        result

#Final list of predictors to be used in the analysis
        Xcols=['ETO_mm/day','SCHEDULED_IRRIGATION_TIME_mins','AVG_WIN
        D_SPEED_MPH','PRECIPITATION_inch','START_TIME_CLASS',
        'TIME_UNTIL_FIRST_RUNOFF_sec']
```

```
data3 = result[Xcols]

data3.to_csv('ASIS3.csv', index = False) #Saving the new generated encoded data
                                          to a new file to use in our algorithm

data = pd.read_csv('C:/Users/samba/Desktop/ASIS3.csv') #Importing the dataset
result = data

#Visualizing the "time until the first instance of runoff is detected" in seconds
from scipy.stats import norm
x_d = np.linspace(-4, 8, 1000)
re1 = result['TIME_UNTIL_FIRST_RUNOFF_sec'].as_matrix(columns=None)
from sklearn.neighbors import KernelDensity
kde = KernelDensity(bandwidth=1.0, kernel='gaussian') #Instantiating and fitting
                                                       the KDE model
kde.fit(re1[:, None])
logprob = kde.score_samples(x_d[:, None]) #score_samples returns the log of the
                                           probability density

plt.fill_between(x_d, np.exp(logprob), alpha=0.5)
frame1 = plt.gca()
frame1.axes.get_yaxis().set_visible(False)
plt.plot(re1, np.full_like(re1, -0.01), '|k', markeredgewidth=1)
plt.ylim(-0.02, 0.22)
plt.xlabel('Values of time until the first instance of runoff is detected in seconds')

#Categorizing the abovementioned data based on its distribution. From the data
visualization, the value of time when the first instance of runoff was detected was 0 seconds
was classified as Class 0, any value between 0 and 600 seconds was classified as Class 1
and any value beyond was classified as Class 2 respectively
re2 = []
for i,l in enumerate(re):
   if l = = 0 :
      re2.append(0)
   elif l > 0 and l <= 600 :
      re2.append(1)
   else :
      re2.append(2)
df4=pd.DataFrame(re2,columns=['TIME_UNTIL_FIRST_RUNOFF_sec_CLASS
'])
result1 = pd.concat([result, df4], axis=1, join='inner')
result1 = result1.drop(['TIME_UNTIL_FIRST_RUNOFF_sec'], axis = 1)

#Normalizing the features and constructing a new Dataframe by appending the target
variable to be used for further analysis
from sklearn import preprocessing
```

```
min_max_scaler = preprocessing.MinMaxScaler()
re2 = result.as_matrix(columns = None)
X_scaled = min_max_scaler.fit_transform(re2[:,0:5])
X_cols3 = ['START_TIME_CLASS']
result2 = result1[X_cols3]
X_cols4 = ['TIME_UNTIL_FIRST_RUNOFF_sec _CLASS']
result3 = result1[X_cols4]
#Constructing a new dataframe for the normalized features
dat1 = pd.DataFrame({'ETO_mm/day':X_scaled[:,0]})
dat2 = pd.DataFrame({'AVG_WIND_SPEED_MPH':X_scaled[:,2]})
dat4 = pd.DataFrame({'PRECIPITATION_inch':X_scaled[:,3]})
dat7 = pd.DataFrame({'EFFECTIVE_IRRIGATION_TIME_min':X_scaled[:,1]})
dataset = pd.concat([dat1, dat2, dat4, dat7, result2, result3], axis=1)
```

#Removing the target variable before running Principal Component Analysis on the dataset

```
y_frame = dataset.iloc[:,-1]
y_data = y_frame.as_matrix(columns = None)
dataset = dataset.drop(['TIME_UNTIL_FIRST_RUNOFF_sec_CLASS'], axis = 1)
```

#Principal Component Analysis on the dataset to get a clear idea about the variance explained

```
from sklearn.decomposition import PCA
X = dataset.as_matrix(columns = None)
pca = PCA(n_components=3)
pca.fit(X)
x_data_pca = pca.fit_transform(X)
var1 = np.cumsum(np.round(pca.explained_variance_ratio_, decimals=2)*100)
print(pca.explained_variance_ratio_)
plt.xlabel("Number of components")
plt.ylabel("Cumulative Proportion of the variance explained")
plt.plot(var1)
print(var1)
data8 = pd.DataFrame(pca.components_,columns=dataset.columns,index = ['PC-1','PC-2','PC-3'])
```

#Synthetic Data generation for observations when the first instance of runoff detected was at 0 seconds and irrigation was conducted during night hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“TIME_UNTIL_FIRST_RUNOFF
_sec_CLASS": "B"})
```

```
dataset2 = dataset1.loc[lambda df: df.B == 0, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 1, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_0=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_0=Synthetic_data_0.assign(**{'START_TIME_CLASS':
1,'TIME_UNTIL_FIRST_RUNOFF_sec_CLASS': 0})
```

#Synthetic Data generation for observations when the first instance of runoff detected was at 0 seconds and irrigation was conducted during morning hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={"TIME_UNTIL_FIRST_RUNOFF
_sec_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 0, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 2, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_1=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_1=Synthetic_data_1.assign(**{'START_TIME_CLASS':2,
'TIME_UNTIL_FIRST_RUNOFF_sec_CLASS': 0})
```

#Synthetic Data generation for observations when the first instance of runoff detected was at 0 seconds and irrigation was conducted during afternoon hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
```

```
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“TIME_UNTIL_FIRST_RUNOFF
_sec_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 0, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 3, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_2=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_2=Synthetic_data_2.assign(**{'START_TIME_CLASS':3,
'TIME_UNTIL_FIRST_RUNOFF_sec_CLASS': 0})
```

#Synthetic Data generation for observations when the first instance of runoff detected was at 0 seconds and irrigation was conducted during evening hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“TIME_UNTIL_FIRST_RUNOFF
_sec_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 0, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 4, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_3=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
```

```
Synthetic_data_3=Synthetic_data_3.assign(**{'START_TIME_CLASS':3,
'TIME_UNTIL_FIRST_RUNOFF_sec_CLASS': 0})
```

#Synthetic Data generation for observations when the first instance of runoff detected was between 0 and 600 seconds and irrigation was conducted during night hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“TIME_UNTIL_FIRST_RUNOF_s
ec_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 1, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 1, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4=Synthetic_data_4.assign(**{'START_TIME_CLASS':1,
'TIME_UNTIL_FIRST_RUNOFF_sec_CLASS': 1})
```

#Synthetic Data generation for observations when the first instance of runoff detected was between 0 and 600 seconds and irrigation was conducted during morning hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“TIME_UNTIL_FIRST_RUNOFF
_sec_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 1, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 2, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
```

```
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_5=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_5=Synthetic_data_5.assign(**{'START_TIME_CLASS':2,
'TIME_UNTIL_FIRST_RUNOFF_sec_CLASS': 1})
```

#Synthetic Data generation for observations when the first instance of runoff detected was between 0 and 600 seconds and irrigation was conducted during afternoon hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“TIME_UNTIL_FIRST_RUNOFF
_sec_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 1, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 3, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_6=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_6=Synthetic_data_6.assign(**{'START_TIME_CLASS':3,
'TIME_UNTIL_FIRST_RUNOFF_sec_CLASS': 1})
```

#Synthetic Data generation for observations when the first instance of runoff detected was between 0 and 600 seconds and irrigation was conducted during evening hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“TIME_UNTIL_FIRST_RUNOFF
_sec_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 1, :]
```

```
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 4, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,  mean=mean_xe,  n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_7=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_7=Synthetic_data_7.assign(**{'START_TIME_CLASS':4,
'TIME_UNTIL_FIRST_RUNOFF_sec_CLASS': 1})
```

#Synthetic Data generation for observations when the first instance of runoff detected was more than 600 seconds and irrigation was conducted during night hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“TIME_UNTIL_FIRST_RUNOFF
_sec_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 2, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 1, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,  mean=mean_xe,  n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_8=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_8=Synthetic_data_8.assign(**{'START_TIME_CLASS':1,
'TIME_UNTIL_FIRST_RUNOFF_sec_CLASS':2})
```

#Synthetic Data generation for observations when the first instance of runoff detected was more than 600 seconds and irrigation was conducted during morning hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
```

```
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“TIME_UNTIL_FIRST_RUNOFF
_sec_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 2, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 2, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,  mean=mean_xe,  n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_9=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_9=Synthetic_data_9.assign(**{'START_TIME_CLASS':2,
'TIME_UNTIL_FIRST_RUNOFF_sec_CLASS':2})
```

#Synthetic Data generation for observations when the first instance of runoff detected was more than 600 seconds and irrigation was conducted during afternoon hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“TIME_UNTIL_FIRST_RUNOFF
_sec_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 2, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 3, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,  mean=mean_xe,  n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_10=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_S
PEED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'
])
```

```
Synthetic_data_10=Synthetic_data_10.assign(**{'START_TIME_CLASS':3,
'TIME_UNTIL_FIRST_RUNOFF_sec_CLASS':2})
```

#Synthetic Data generation for observations when the first instance of runoff detected was more than 600 seconds and irrigation was conducted during evening hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“TIME_UNTIL_FIRST_RUNOFF
_sec_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 2, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 4, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,  mean=mean_xe,  n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_11=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_S
PEED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min’
])
Synthetic_data_11=Synthetic_data_11.assign(**{'START_TIME_CLASS':4,
'TIME_UNTIL_FIRST_RUNOFF_sec_CLASS':2})
```

#Combining all the generated data for distinct mean and covariance matrix into a single Data frame

```
result1 = Synthetic_data_1.append(Synthetic_data_0, ignore_index=True)
result2 = result1.append(Synthetic_data_2, ignore_index=True)
result3 = result2.append(Synthetic_data_3, ignore_index=True)
result4 = result3.append(Synthetic_data_4, ignore_index=True)
result5 = result4.append(Synthetic_data_5, ignore_index=True)
result6 = result5.append(Synthetic_data_6, ignore_index=True)
result7 = result6.append(Synthetic_data_7, ignore_index=True)
result8 = result7.append(Synthetic_data_8, ignore_index=True)
result9 = result8.append(Synthetic_data_9, ignore_index=True)
result10 = result9.append(Synthetic_data_10, ignore_index=True)
result11 = result10.append(Synthetic_data_11, ignore_index=True)
```

```
#Synthetic Data generation for all irrigation events during the night (Shared mean and covariance matrix between classes)
    dataset1=dataset.rename(index=str,columns={"TIME_UNTIL_FIRST_RUNOFF
    _sec_CLASS": "B"})
    dataset2 = dataset1.rename(index=str, columns={"START_TIME_CLASS": "C"})
    dataset3 = dataset2.loc[lambda df: df.C == 1, :]
    dataset4 = dataset3.drop(['C','B'], axis = 1)
    xe = dataset4.as_matrix(columns=None)
    mean_xe = np.mean(xe, axis=0)
    cov_xe = np.cov(xe, rowvar=0)
    from sklearn.datasets import make_gaussian_quantiles
    #Constructing the dataset
    X1,y1=make_gaussian_quantiles(cov=cov_xe,mean=mean_xe,n_samples=1000,
    n_features=4,n_classes=3, random_state=1)
    Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
    EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
    Synthetic_data_4 = Synthetic_data_4.assign(**{'START_TIME_CLASS': 1})
    Synthetic_data_5=pd.DataFrame(y1,columns=["TIME_UNTIL_FIRST_RUNOF
    F_sec_CLASS"])
    result_C_12=pd.concat([Synthetic_data_4,Synthetic_data_5],axis=1,join_axes=[S
    ynthetic_data_4.index])

#Synthetic Data generation for all irrigation events during the morning (Shared mean and covariance matrix between classes)
    dataset1=dataset.rename(index=str,columns={"RUNOFF_VOLUME_OBSERVE
    D_Gal_CLASS": "B"})
    dataset2 = dataset1.rename(index=str, columns={"START_TIME_CLASS": "C"})
    dataset3 = dataset2.loc[lambda df: df.C == 2, :]
    dataset4 = dataset3.drop(['C','B'], axis = 1)
    xe = dataset4.as_matrix(columns=None)
    mean_xe = np.mean(xe, axis=0)
    cov_xe = np.cov(xe, rowvar=0)
    from sklearn.datasets import make_gaussian_quantiles
     #Construct dataset
    X1,y1= make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=1000,
    n_features=4, n_classes=3, random_state=1)
    Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
    EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
    Synthetic_data_4 = Synthetic_data_4.assign(**{'START_TIME_CLASS': 2})
    Synthetic_data_5=pd.DataFrame(y1,columns=["TIME_UNTIL_FIRST_RUNOF
    F_sec_CLASS"])
    result_C_13=pd.concat([Synthetic_data_4,Synthetic_data_5],axis=1,join_axes=[S
    ynthetic_data_4.index])

#Synthetic Data generation for all irrigation events during the afternoon (Shared mean and covariance matrix between classes)
```

```
dataset1=dataset.rename(index=str,columns={"TIME_UNTIL_FIRST_RUNOFF
_sec_CLASS": "B"})
dataset2 = dataset1.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset3 = dataset2.loc[lambda df: df.C == 3, :]
dataset4 = dataset3.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Construct dataset
X1,y1= make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=1000,
n_features=4, n_classes=3, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch',
'EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4 = Synthetic_data_4.assign(**{'START_TIME_CLASS': 3})
Synthetic_data_5=pd.DataFrame(y1,columns=['TIME_UNTIL_FIRST_RUNOFF
_sec_CLASS'])
result_C_14=pd.concat([Synthetic_data_4,Synthetic_data_5],axis=1,join_axes=[S
ynthetic_data_4.index])
```

#Synthetic Data generation for all irrigation events during the evening (Shared mean and covariance matrix between classes)

```
dataset1=dataset.rename(index=str,columns={"TIME_UNTIL_FIRST_RUNOFF
_sec_CLASS": "B"})
dataset2 = dataset1.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset3 = dataset2.loc[lambda df: df.C == 4, :]
dataset4 = dataset3.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Construct dataset
X1, y1=make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=1000,
n_features=4, n_classes=3, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4 = Synthetic_data_4.assign(**{'START_TIME_CLASS': 4})
Synthetic_data_5=pd.DataFrame(y1,columns=['TIME_UNTIL_FIRST_RUNOFF
_sec_CLASS'])
result_C_15=pd.concat([Synthetic_data_4,Synthetic_data_5],axis=1,join_axes=[S
ynthetic_data_4.index])
```

#Combining all the generated data into a single Data frame

```
result12 = result_C_12.append(result_C_13, ignore_index=True)
result13 = result12.append(result_C_14, ignore_index=True)
```

```
result14 = result13.append(result_C_15, ignore_index=True)
final_result = result11.append(result14, ignore_index=True)

#Training RBF SVM for different values of train:test split on the synthetic data
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
X = final_result.drop("TIME_UNTIL_FIRST_RUNOFF_sec_CLASS", axis = 1)
y = final_result['TIME_UNTIL_FIRST_RUNOFF_sec_CLASS']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state
= 42)
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
style.use("ggplot")
from sklearn import svm
b = X_train.as_matrix(columns = None)
c = y_train.as_matrix(columns = None)
from sklearn import svm
# Grid Search
# Parameter Grid
gammas = np.linspace(0.0, 1.0, num=100)
param_grid = {'C': [100], 'gamma': gammas} #The value of penalty parameter is
                                            varied between 1 to 100 I terms of
                                            10 units. Best Gamma is chosen for
                                            each case.
# Make grid search classifier
clf_grid = GridSearchCV(svm.SVC(), param_grid, verbose=1)
# Training the classifier
clf_grid.fit(b,c)
print("Best Parameters:\n", clf_grid.best_params_)
print("Best Estimators:\n", clf_grid.best_estimator_)
b1 = X_test.as_matrix(columns = None)
c1 = y_test.as_matrix(columns = None)
result = clf_grid.predict(b1)
accuracy_score(c1, result) #Gives the accuracy score on testing data

#Training RBF SVM on the entire synthetic data generated and training it on the real
dataset
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
X_train = final_result.drop('TIME_UNTIL_FIRST_RUNOFF_sec_CLASS', axis
= 1)
y_train = final_result['TIME_UNTIL_FIRST_RUNOFF_sec_CLASS']
style.use("ggplot")
```

```
from sklearn import svm
b = X_train.as_matrix(columns = None)
c = y_train.as_matrix(columns = None)
X_test = dataset.drop('TIME_UNTIL_FIRST_RUNOFF_sec_CLASS', axis = 1)
y_test = dataset['TIME_UNTIL_FIRST_RUNOFF_sec_CLASS']
b1 = X_test.as_matrix(columns = None)
c1 = y_test.as_matrix(columns = None)
from sklearn import svm
#Grid Search
#Parameter Grid
gammas = np.linspace(0.0, 1.0, num=100)
param_grid1 = {'C': [100], 'gamma': gammas}
#Make grid search classifier
clf_grid1 = GridSearchCV(svm.SVC(), param_grid1, verbose=1)

#Train the classifier
clf_grid1.fit(b,c)
#clf = grid.best_estimator_()
print("Best Parameters:\n", clf_grid1.best_params_)
print("Best Estimators:\n", clf_grid1.best_estimator_)
result1 = clf_grid1.predict(b1)
accuracy_score(c1, result1)
```

APPENDIX C

(a) ML code when the target variable "SOIL WETTING EFFICIENCY INDEX" is binned into two classes

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

data = pd.read_csv('C:/Users/samba/Desktop/ASIS_new.csv') #Importing the dataset

print('Training data shape:', data.shape) #Initial volume of the dataset (31 observations and
                                          32 predictors)

#Categorizing the start time variable which we would be using in our analysis
#Any time between 9 pm and 4 am is categorized as Night time denoted by 1, any time
between 4 am and 12 pm is categorized as Morning time denoted by 2, any time between
12 pm and 4 pm is categorized as Afternoon time denoted by 3 and any time between 4 pm
and 9 pm is categorized as Evening time denoted by 4
        hour1= []
        rey = list(data['START_TIME_hr:min'].values)
        for i,j in enumerate(rey):
           hrs,mins = j.split(':')
           hrs = np.asarray(hrs)
           hrs = hrs.astype(int)
           for i in np.nditer(hrs):
              if i>21 and i<=23:
                 hour1.append(1)
              elif i>=0 and i<4:                #Night
                 hour1.append(1)
              elif i>=4 and i<=12:              #Morning
                 hour1.append(2)
              elif i>12 and i<=16:              #Afternoon
                 hour1.append(3)
              elif i>16 and i<=21:               #Evening
                 hour1.append(4)
        df1 = pd.DataFrame(hour1,columns=['START_TIME_CLASS'])
        result = pd.concat([data, df1], axis=1, join='inner')
        result = result.drop(['START_TIME_hr:min'], axis = 1)
        result

#Final list of predictors to be used in the analysis
        Xcols=['ETO_mm/day','SCHEDULED_IRRIGATION_TIME_mins','AVG
        WIND_SPEED_MPH','PRECIPITATION_inch','START_TIME_CLASS','SOIL_
        WETTING_EFFICIENCY_INDEX']
```

```
data3 = result[Xcols]

data3.to_csv('ASIS3.csv', index = False) #Saving the new generated encoded data
                                          to a new file to use in our algorithm
data = pd.read_csv('C:/Users/samba/Desktop/ASIS3.csv') #Importing the dataset
result = data

#Visualzing the Soil Wetting Efficiency Index data
from scipy.stats import norm
x_d = np.linspace(-4, 8, 1000)
re1=result['SOIL_WETTING_EFFICIENCY_INDEX'].as_matrix(columns=None
)
from sklearn.neighbors import KernelDensity
kde = KernelDensity(bandwidth=1.0, kernel='gaussian') #Instantiating and fitting
                                                       the KDE model
kde.fit(re1[:, None])
logprob = kde.score_samples(x_d[:, None]) #score_samples returns the log of the
                                           probability density

plt.fill_between(x_d, np.exp(logprob), alpha=0.5)
frame1 = plt.gca()
frame1.axes.get_yaxis().set_visible(False)
plt.plot(re1, np.full_like(re1, -0.01), '|k', markeredgewidth=1)
plt.ylim(-0.02, 0.22)
plt.xlabel('Values of SOIL WETTING EFFICIENCY INDEX')

#Categorizing the value of runoff based on its distribution. From the data visualization,
the value of SWEI below 1.7 was classified as Class 0 and any value over it was classified
as Class 1 for analysis
re2 = []
for i,l in enumerate(re):
   if l <= 1.7 :
      re2.append(0)
   else :
      re2.append(1)
df4=pd.DataFrame(re2,columns=['SOIL_WETTING_EFFICIENCY_INDEX_CL
ASS'])
result1 = pd.concat([result, df4], axis=1, join='inner')
result1 = result1.drop(['SOIL_WETTING_EFFICIENCY_INDEX'], axis = 1)

#Normalizing the features and constructing a new Dataframe by appending the target
variable to be used for further analysis
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
re2 = result.as_matrix(columns = None)
X_scaled = min_max_scaler.fit_transform(re2[:,0:5])
```

```
X_cols3 = ['START_TIME_CLASS']
result2 = result1[X_cols3]
X_cols4 = ['SOIL_WETTING_EFFICIENCY_INDEX_CLASS']
result3 = result1[X_cols4]
#Constructing a new dataframe for the normalized features
dat1 = pd.DataFrame({'ETO_mm/day':X_scaled[:,0]})
dat2 = pd.DataFrame({'AVG_WIND_SPEED_MPH':X_scaled[:,2]})
dat4 = pd.DataFrame({'PRECIPITATION_inch':X_scaled[:,3]})
dat7 = pd.DataFrame({'EFFECTIVE_IRRIGATION_TIME_min':X_scaled[:,1]})
dataset = pd.concat([dat1, dat2, dat4, dat7, result2, result3], axis=1)
```

#Removing the target variable before running Principal Component Analysis on the dataset

```
y_frame = dataset.iloc[:,-1]
y_data = y_frame.as_matrix(columns = None)
dataset = dataset.drop(['SOIL_WETTING_EFFICIENCY_INDEX_CLASS'], axis = 1)
```

#Principal Component Analysis on the dataset to get a clear idea about the variance explained

```
from sklearn.decomposition import PCA
X = dataset.as_matrix(columns = None)
pca = PCA(n_components=3)
pca.fit(X)
x_data_pca = pca.fit_transform(X)
var1 = np.cumsum(np.round(pca.explained_variance_ratio_, decimals=2)*100)
print(pca.explained_variance_ratio_)
plt.xlabel("Number of components")
plt.ylabel("Cumulative Proportion of the variance explained")
plt.plot(var1)
print(var1)
data8 = pd.DataFrame(pca.components_,columns=dataset.columns,index = ['PC-1','PC-2','PC-3'])
```

#Synthetic Data generation for observations with SWEI less than 1.7 and irrigation during night hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={"SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 0, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 1, :]
```

```
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,  mean=mean_xe,  n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_0=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_0=Synthetic_data_0.assign(**{'START_TIME_CLASS':1,‘SOIL
_WETTING_EFFICIENCY_INDEX_CLASS’: 0})
```

 # Synthetic Data generation for observations with SWEI less than 1.7 and irrigation during morning hours (Distinct mean and covariance matrix between the classes)

```
dataset1=dataset.rename(index=str,columns={“SOIL_WETTING_EFFICIENCY
_INDEX_CLASS”: "B"})
dataset2 = dataset1.loc[lambda df: df.B == 0, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 2, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1, y1 = make_gaussian_quantiles(cov=cov_xe, mean=mean_xe,
                    n_samples=500, n_features=4,
                    n_classes=1, random_state=1)
Synthetic_data_1=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_1=Synthetic_data_1.assign(**{'START_TIME_CLASS':2,
‘SOIL_WETTING_EFFICIENCY_INDEX_CLASS’: 0})
```

 # Synthetic Data generation for observations with SWEI less than 1.7 and irrigation during afternoon hours (Distinct mean and covariance matrix between the classes)

```
dataset1=dataset.rename(index=str,columns={"SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 0, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 3, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
```

```
#Construct dataset
X1, y1 = make_gaussian_quantiles(cov=cov_xe, mean=mean_xe,
                        n_samples=500, n_features=4,
                        n_classes=1, random_state=1)
Synthetic_data_2=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch',‘EFFECTIVE_IRRIGATION_TIME_min']
)
Synthetic_data_2=Synthetic_data_2.assign(**{'START_TIME_CLASS':3,'SOIL
_WETTING_EFFICIENCY_INDEX_CLASS': 0})
```

 # Synthetic Data generation for observations with SWEI less than 1.7 and irrigation during evening hours (Distinct mean and covariance matrix between the classes)

```
dataset1=dataset.rename(index=str,columns={'SOIL_WETTING_EFFICIENCY_
INDEX_CLASS': "B"})
dataset2 = dataset1.loc[lambda df: df.B == 0, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 4, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
 #Construct dataset
X1, y1 = make_gaussian_quantiles(cov=cov_xe, mean=mean_xe,
                        n_samples=500, n_features=4,
                        n_classes=1, random_state=1)
Synthetic_data_3=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_3=Synthetic_data_3.assign(**{'START_TIME_CLASS':4,'SOIL
_WETTING_EFFICIENCY_INDEX_CLASS': 0})
```

 # Synthetic Data generation for observations with SWEI more than 1.7 and irrigation during night hours (Distinct mean and covariance matrix between the classes)

```
dataset1=dataset.rename(index=str,columns={"SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 1, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 1, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Construct dataset
X1, y1 = make_gaussian_quantiles(cov=cov_xe, mean=mean_xe,
                        n_samples=500, n_features=4,
```

```
                                n_classes=1, random_state=1)
        Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
        EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
        Synthetic_data_4=Synthetic_data_4.assign(**{'START_TIME_CLASS':1,'SOIL
        _WETTING_EFFICIENCY_INDEX_CLASS ': 1})

# Synthetic Data generation for observations with SWEI more than 1.7 and irrigation
during morning hours (Distinct mean and covariance matrix between the classes)
        dataset1=dataset.rename(index=str,columns={"SOIL_WETTING_EFFICIENCY
        _INDEX_CLASS": "B"})
        dataset2 = dataset1.loc[lambda df: df.B == 1, :]
        dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
        dataset4 = dataset3.loc[lambda df: df.C == 2, :]
        dataset4 = dataset4.drop(['C','B'], axis = 1)
        xe = dataset4.as_matrix(columns=None)
        mean_xe = np.mean(xe, axis=0)
        cov_xe = np.cov(xe, rowvar=0)
        from sklearn.datasets import make_gaussian_quantiles
         #Construct dataset
        X1, y1 = make_gaussian_quantiles(cov=cov_xe, mean=mean_xe,
                                n_samples=500, n_features=4,
                                n_classes=1, random_state=1)
        Synthetic_data_5=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
        EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
        Synthetic_data_5=Synthetic_data_5.assign(**{'START_TIME_CLASS':2,'SOIL
        _WETTING_EFFICIENCY_INDEX_CLASS ': 1})

# Synthetic Data generation for observations with SWEI more than 1.7 and irrigation
during evening hours (Distinct mean and covariance matrix between the classes)
        dataset1=dataset.rename(index=str,columns={"SOIL_WETTING_EFFICIENCY
        _INDEX_CLASS ": "B"})
        dataset2 = dataset1.loc[lambda df: df.B == 1, :]
        dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
        dataset4 = dataset3.loc[lambda df: df.C == 4, :]
        dataset4 = dataset4.drop(['C','B'], axis = 1)
        xe = dataset4.as_matrix(columns=None)
        mean_xe = np.mean(xe, axis=0)
        cov_xe = np.cov(xe, rowvar=0)
        from sklearn.datasets import make_gaussian_quantiles
        #Construct dataset
        X1, y1 = make_gaussian_quantiles(cov=cov_xe, mean=mean_xe,
                                n_samples=500, n_features=4,
                                n_classes=1, random_state=1)
        Synthetic_data_7=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
        EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
```

```
Synthetic_data_7=Synthetic_data_7.assign(**{'START_TIME_CLASS':4,‘SOIL
_WETTING_EFFICIENCY_INDEX_CLASS’: 1})
```

#Combining all the generated data for distinct mean and covariance matrix into a single Data frame

```
result1 = Synthetic_data_1.append(Synthetic_data_0, ignore_index=True)
result2 = result1.append(Synthetic_data_2, ignore_index=True)
result3 = result2.append(Synthetic_data_3, ignore_index=True)
result5 = result3.append(Synthetic_data_5, ignore_index=True)
result8 = result5.append(Synthetic_data_7, ignore_index=True)
```

#Synthetic Data generation for all irrigation events during the night (Shared mean and covariance matrix between classes)

```
dataset1=dataset.rename(index=str,columns={"SOIL_WETTING_EFFICIENCY
_INDEX_CLASS ": "B"})
dataset2 = dataset1.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset3 = dataset2.loc[lambda df: df.C == 1, :]
dataset4 = dataset3.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Constructing the dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,mean=mean_xe,n_samples=1000,
n_features=4,n_classes=2, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4 = Synthetic_data_4.assign(**{'START_TIME_CLASS': 1})
Synthetic_data_5=pd.DataFrame(y1,columns=[‘SOIL_WETTING_EFFICIENCY
_INDEX_CLASS’])
result_C_12=pd.concat([Synthetic_data_4,Synthetic_data_5],axis=1,join_axes=[S
ynthetic_data_4.index])
```

#Synthetic Data generation for all irrigation events during the morning (Shared mean and covariance matrix between classes)

```
dataset1=dataset.rename(index=str,columns={"SOIL_WETTING_EFFICIENCY
_INDEX_CLASS ": "B"})
dataset2 = dataset1.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset3 = dataset2.loc[lambda df: df.C == 2, :]
dataset4 = dataset3.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Constructing the dataset
```

```
X1,y1= make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=1000,
n_features=4, n_classes=2, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4 = Synthetic_data_4.assign(**{'START_TIME_CLASS': 2})
Synthetic_data_5=pd.DataFrame(y1,columns=['SOIL_WETTING_EFFICIENCY
_INDEX_CLASS '])
result_C_13=pd.concat([Synthetic_data_4,Synthetic_data_5],axis=1,join_axes=[S
ynthetic_data_4.index])
```

#Synthetic Data generation for all irrigation events during the afternoon (Shared mean and covariance matrix between classes)

```
dataset1=dataset.rename(index=str,columns={"SOIL_WETTING_EFFICIENCY
_INDEX_CLASS ": "B"})
dataset2 = dataset1.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset3 = dataset2.loc[lambda df: df.C == 3, :]
dataset4 = dataset3.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Construct dataset
X1,y1= make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=1000,
n_features=4, n_classes=2, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch',
'EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4 = Synthetic_data_4.assign(**{'START_TIME_CLASS': 3})
Synthetic_data_5=pd.DataFrame(y1,columns=['SOIL_WETTING_EFFICIENCY
_INDEX_CLASS '])
result_C_14=pd.concat([Synthetic_data_4,Synthetic_data_5],axis=1,join_axes=[S
ynthetic_data_4.index])
```

#Synthetic Data generation for all irrigation events during the evening (Shared mean and covariance matrix between classes)

```
dataset1=dataset.rename(index=str,columns={"SOIL_WETTING_EFFICIENCY
_INDEX_CLASS ": "B"})
dataset2 = dataset1.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset3 = dataset2.loc[lambda df: df.C == 4, :]
dataset4 = dataset3.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Construct dataset
```

```
X1,y1= make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=1000,
n_features=4, n_classes=2, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4 = Synthetic_data_4.assign(**{'START_TIME_CLASS': 4})
Synthetic_data_5=pd.DataFrame(y1,columns=[‘SOIL_WETTING_EFFICIENCY
_INDEX_CLASS’])
result_C_15=pd.concat([Synthetic_data_4,Synthetic_data_5],axis=1,join_axes=[S
ynthetic_data_4.index])

#Combining all the generated data into a single Data frame
result5 = result_C_12.append(result_C_13, ignore_index=True)
result6 = result5.append(result_C_14, ignore_index=True)
result7 = result6.append(result_C_15, ignore_index=True)
final_result = result8.append(result7, ignore_index=True)

#Training RBF SVM for different values of train:test split on the synthetic data
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
X = final_result.drop('SOIL_WETTING_EFFICIENCY_INDEX_CLASS’, axis =
1)
y = final_result['SOIL_WETTING_EFFICIENCY_INDEX_CLASS']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state
= 42)
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
style.use("ggplot")
from sklearn import svm
b = X_train.as_matrix(columns = None)
c = y_train.as_matrix(columns = None)
from sklearn import svm
# Grid Search
# Parameter Grid
gammas = np.linspace(0.0, 1.0, num=100)
param_grid = {'C': [100], 'gamma': gammas} #The value of penalty parameter is
                                            varied between 1 to 100 in interval
                                            of 10 units. Best Gamma is chosen
                                            for each case.
# Make grid search classifier
clf_grid = GridSearchCV(svm.SVC(), param_grid, verbose=1)
# Training the classifier
clf_grid.fit(b,c)
print("Best Parameters:\n", clf_grid.best_params_)
print("Best Estimators:\n", clf_grid.best_estimator_)
```

```
b1 = X_test.as_matrix(columns = None)
c1 = y_test.as_matrix(columns = None)
result = clf_grid.predict(b1)
accuracy_score(c1, result) #Gives the accuracy score on testing data

#Training RBF SVM on the entire synthetic data generated and training it on the real dataset
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
X_train = final_result.drop('SOIL_WETTING_EFFICIENCY_INDEX_CLASS', axis =1)
y_train = final_result['SOIL_WETTING_EFFICIENCY_INDEX_CLASS']
style.use("ggplot")
from sklearn import svm
b = X_train.as_matrix(columns = None)
c = y_train.as_matrix(columns = None)
X_test = dataset.drop('SOIL_WETTING_EFFICIENCY_INDEX_CLASS', axis = 1)
y_test = dataset['SOIL_WETTING_EFFICIENCY_INDEX_CLASS']
b1 = X_test.as_matrix(columns = None)
c1 = y_test.as_matrix(columns = None)
from sklearn import svm
#Grid Search
#Parameter Grid
gammas = np.linspace(0.0, 1.0, num=100)
param_grid1 = {'C': [100], 'gamma': gammas}
#Make grid search classifier
clf_grid1 = GridSearchCV(svm.SVC(), param_grid1, verbose=1)

#Train the classifier
clf_grid1.fit(b,c)
#clf = grid.best_estimator_()
print("Best Parameters:\n", clf_grid1.best_params_)
print("Best Estimators:\n", clf_grid1.best_estimator_)
result1 = clf_grid1.predict(b1)
accuracy_score(c1, result1)
```

(b) ML code when the target variable "SOIL WETTING EFFICIENCY INDEX" is binned into three classes

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

data = pd.read_csv('C:/Users/samba/Desktop/ASIS_new.csv') #Importing the dataset

print('Training data shape:', data.shape) #Initial volume of the dataset (31 observations and
                                          32 predictors)

#Categorizing the start time variable which we would be using in our analysis
#Any time between 9 pm and 4 am is categorized as Night time denoted by 1, any time
between 4 am and 12 pm is categorized as Morning time denoted by 2, any time between
12 pm and 4 pm is categorized as Afternoon time denoted by 3 and any time between 4 pm
and 9 pm is categorized as Evening time denoted by 4
        hour1= []
        rey = list(data['START_TIME_hr:min'].values)
        for i,j in enumerate(rey):
           hrs,mins = j.split(':')
           hrs = np.asarray(hrs)
           hrs = hrs.astype(int)
           for i in np.nditer(hrs):
              if i>21 and i<=23:
                 hour1.append(1)
              elif i>=0 and i<4:                #Night
                 hour1.append(1)
              elif i>=4 and i<=12:              #Morning
                 hour1.append(2)
              elif i>12 and i<=16:              #Afternoon
                 hour1.append(3)
              elif i>16 and i<=21:              #Evening
                 hour1.append(4)
        df1 = pd.DataFrame(hour1,columns=['START_TIME_CLASS'])
        result = pd.concat([data, df1], axis=1, join='inner')
        result = result.drop(['START_TIME_hr:min'], axis = 1)

#Final list of predictors to be used in the analysis
        Xcols=['ETO_mm/day','SCHEDULED_IRRIGATION_TIME_mins','AVG_WIN
        D_SPEED_MPH','PRECIPITATION_inch','START_TIME_CLASS',
        'SOIL_WETTING_EFFICIENCY_INDEX']
        data3 = result[Xcols]
        data3.to_csv('ASIS3.csv', index = False) #Saving the new generated encoded data
                                                 to a new file to use in our algorithm
```

```
data = pd.read_csv('C:/Users/samba/Desktop/ASIS3.csv') #Importing the dataset
result = data

#Visualizing the "SOIL WETTING EFFICIENCY INDEX" variable
from scipy.stats import norm
x_d = np.linspace(-4, 8, 1000)
re1=result['SOIL_WETTING_EFFICIENCY_INDEX'].as_matrix(columns=None
)
from sklearn.neighbors import KernelDensity
kde = KernelDensity(bandwidth=1.0, kernel='gaussian') #Instantiating and fitting
                                                        the KDE model
kde.fit(re1[:, None])
logprob = kde.score_samples(x_d[:, None]) #score_samples returns the log of the
                                                    probability density

plt.fill_between(x_d, np.exp(logprob), alpha=0.5)
frame1 = plt.gca()
frame1.axes.get_yaxis().set_visible(False)
plt.plot(re1, np.full_like(re1, -0.01), '|k', markeredgewidth=1)
plt.ylim(-0.02, 0.22)
plt.xlabel('Values of SOIL WETTING EFFICIENCY INDEX')

#Categorizing the abovementioned data based on its distribution. From the data
visualization, the value of time when the SWEI calculated was less than 1.5 was classified
as Class 0, any value between 1.5 and 2.2 was classified as Class 1 and any value beyond
was classified as Class 2 respectively
re2 = []
for i,l in enumerate(re):
   if l <= 1.5 :
      re2.append(0)
   elif l > 1.5 and l <= 2.2 :
      re2.append(1)
   else :
      re2.append(2)
df4=pd.DataFrame(re2,columns=['SOIL_WETTING_EFFICIENCY_INDEX_CL
ASS'])
result1 = pd.concat([result, df4], axis=1, join='inner')
result1 = result1.drop(['SOIL_WETTING_EFFICIENCY_INDEX'], axis = 1)

#Normalizing the features and constructing a new Dataframe by appending the target
variable to be used for further analysis
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
re2 = result.as_matrix(columns = None)
X_scaled = min_max_scaler.fit_transform(re2[:,0:5])
X_cols3 = ['START_TIME_CLASS']
```

```
result2 = result1[X_cols3]
X_cols4 = ['SOIL_WETTING_EFFICIENCY_INDEX_CLASS']
result3 = result1[X_cols4]
#Constructing a new dataframe for the normalized features
dat1 = pd.DataFrame({'ETO_mm/day':X_scaled[:,0]})
dat2 = pd.DataFrame({'AVG_WIND_SPEED_MPH':X_scaled[:,2]})
dat4 = pd.DataFrame({'PRECIPITATION_inch':X_scaled[:,3]})
dat7 = pd.DataFrame({'EFFECTIVE_IRRIGATION_TIME_min':X_scaled[:,1]})
dataset = pd.concat([dat1, dat2, dat4, dat7, result2, result3], axis=1)
```

#Removing the target variable before running Principal Component Analysis on the dataset

```
y_frame = dataset.iloc[:,-1]
y_data = y_frame.as_matrix(columns = None)
dataset = dataset.drop(['SOIL_WETTING_EFFICIENCY_INDEX_CLASS'], axis = 1)
```

#Principal Component Analysis on the dataset to get a clear idea about the variance explained

```
from sklearn.decomposition import PCA
X = dataset.as_matrix(columns = None)
pca = PCA(n_components=3)
pca.fit(X)
x_data_pca = pca.fit_transform(X)
var1 = np.cumsum(np.round(pca.explained_variance_ratio_, decimals=2)*100)
print(pca.explained_variance_ratio_)
plt.xlabel("Number of components")
plt.ylabel("Cumulative Proportion of the variance explained")
plt.plot(var1)
print(var1)
data8 = pd.DataFrame(pca.components_,columns=dataset.columns,index = ['PC-1','PC-2','PC-3'])
```

#Synthetic Data generation for observations when SWEI was calculated to be less than 1.5 and irrigation was conducted during night hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“SOIL_WETTING_EFFICIENCY_INDEX_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 0, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 1, :]
```

```
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,  mean=mean_xe,  n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_0=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_0=Synthetic_data_0.assign(**{'START_TIME_CLASS':1,'SOIL
_WETTING_EFFICIENCY_INDEX_CLASS': 0})
```

#Synthetic Data generation for observations when the SWEI was calculated to be less than 1.5 and irrigation was conducted during morning hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 0, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 2, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,  mean=mean_xe,  n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_1=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_1=Synthetic_data_1.assign(**{'START_TIME_CLASS':2,'SOIL
_WETTING_EFFICIENCY_INDEX_CLASS': 0})
```

#Synthetic Data generation for observations when the SWEI was calculated to be less than 1.5 and irrigation was conducted during afternoon hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
```

```
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 0, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 3, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,  mean=mean_xe,  n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_2=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_2=Synthetic_data_2.assign(**{'START_TIME_CLASS':3,
‘SOIL_WETTING_EFFICIENCY_INDEX_CLASS': 0})
```

#Synthetic Data generation for observations when the SWEI was calculated to be less than 1.5 and irrigation was conducted during evening hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 0, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 4, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,  mean=mean_xe,  n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_3=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_3=Synthetic_data_3.assign(**{'START_TIME_CLASS':3,
‘SOIL_WETTING_EFFICIENCY_INDEX_CLASS': 0})
```

#Synthetic Data generation for observations when the SWEI was calculated to be less than 1.5 and irrigation was conducted during night hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 1, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 1, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,  mean=mean_xe,  n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4=Synthetic_data_4.assign(**{'START_TIME_CLASS':1,
'SOIL_WETTING_EFFICIENCY_INDEX_CLASS': 1})
```

#Synthetic Data generation for observations when the SWEI was calculated to be between 1.5 and 2.2 and irrigation was conducted during morning hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 1, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 2, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
```

```
X1,y1=make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_5=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_5=Synthetic_data_5.assign(**{'START_TIME_CLASS':2,
'SOIL_WETTING_EFFICIENCY_INDEX_CLASS': 1})
```

#Synthetic Data generation for observations when the SWEI was calculated to be between 1.5 and 2.2 and irrigation was conducted during afternoon hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 1, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 3, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_6=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_6=Synthetic_data_6.assign(**{'START_TIME_CLASS':3,
'SOIL_WETTING_EFFICIENCY_INDEX_CLASS': 1})
```

#Synthetic Data generation for observations when the SWEI was calculated to be between 1.5 and 2.2 and irrigation was conducted during evening hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 1, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 4, :]
```

```
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_7=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_7=Synthetic_data_7.assign(**{'START_TIME_CLASS':4,
'SOIL_WETTING_EFFICIENCY_INDEX_CLASS': 1})
```

#Synthetic Data generation for observations when the SWEI was calculated to be more than 2.2 and irrigation was conducted during night hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 2, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 1, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_8=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_8=Synthetic_data_8.assign(**{'START_TIME_CLASS':1,
'SOIL_WETTING_EFFICIENCY_INDEX_CLASS':2})
```

#Synthetic Data generation for observations when the SWEI was calculated to be more than 2.2 and irrigation was conducted during morning hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
```

```
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 2, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 2, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,  mean=mean_xe,  n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_9=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_9=Synthetic_data_9.assign(**{'START_TIME_CLASS':2,
'SOIL_WETTING_EFFICIENCY_INDEX_CLASS':2})
```

#Synthetic Data generation for observations when the SWEI calculated was more than 2.2 and irrigation was conducted during afternoon hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 2, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 3, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,  mean=mean_xe,  n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_10=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_S
PEED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'
])
Synthetic_data_10=Synthetic_data_10.assign(**{'START_TIME_CLASS':3,
'SOIL_WETTING_EFFICIENCY_INDEX_CLASS':2})
```

#Synthetic Data generation for observations when the SWEI calculated was more than 2.2 and irrigation was conducted during evening hours (Distinct mean and covariance matrix between the classes)

```
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
from scipy.stats import multivariate_normal
dataset1=dataset.rename(index=str,columns={“SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.loc[lambda df: df.B == 2, :]
dataset3 = dataset2.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset4 = dataset3.loc[lambda df: df.C == 4, :]
dataset4 = dataset4.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
# Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,  mean=mean_xe,  n_samples=500,
n_features=4, n_classes=1, random_state=1)
Synthetic_data_11=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_S
PEED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'
])
Synthetic_data_11=Synthetic_data_11.assign(**{'START_TIME_CLASS':4,
'SOIL_WETTING_EFFICIENCY_INDEX_CLASS':2})
```

#Combining all the generated data for distinct mean and covariance matrix into a single Data frame

```
result1 = Synthetic_data_1.append(Synthetic_data_0, ignore_index=True)
result2 = result1.append(Synthetic_data_2, ignore_index=True)
result3 = result2.append(Synthetic_data_3, ignore_index=True)
result4 = result3.append(Synthetic_data_4, ignore_index=True)
result5 = result4.append(Synthetic_data_5, ignore_index=True)
result6 = result5.append(Synthetic_data_6, ignore_index=True)
result7 = result6.append(Synthetic_data_7, ignore_index=True)
result8 = result7.append(Synthetic_data_8, ignore_index=True)
result9 = result8.append(Synthetic_data_9, ignore_index=True)
result10 = result9.append(Synthetic_data_10, ignore_index=True)
result11 = result10.append(Synthetic_data_11, ignore_index=True)
```

#Synthetic Data generation for all irrigation events during the night (Shared mean and covariance matrix between classes)

```
dataset1=dataset.rename(index=str,columns={"SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
```

```
dataset2 = dataset1.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset3 = dataset2.loc[lambda df: df.C == 1, :]
dataset4 = dataset3.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Constructing the dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe,mean=mean_xe,n_samples=1000,
n_features=4,n_classes=3, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch','EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4 = Synthetic_data_4.assign(**{'START_TIME_CLASS': 1})
Synthetic_data_5=pd.DataFrame(y1,columns=["SOIL_WETTING_EFFICIENCY
_INDEX_CLASS"])
result_C_12=pd.concat([Synthetic_data_4,Synthetic_data_5],axis=1,join_axes=[S
ynthetic_data_4.index])
```

#Synthetic Data generation for all irrigation events during the morning (Shared mean and covariance matrix between classes)

```
dataset1=dataset.rename(index=str,columns={"SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset3 = dataset2.loc[lambda df: df.C == 2, :]
dataset4 = dataset3.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Construct dataset
X1, y1 = make_gaussian_quantiles(cov=cov_xe, mean=mean_xe,
n_samples=1000, n_features=4, n_classes=3, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch',
'EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4 = Synthetic_data_4.assign(**{'START_TIME_CLASS': 2})
Synthetic_data_5=pd.DataFrame(y1,columns=["SOIL_WETTING_EFFICIENCY
_INDEX_CLASS"])
result_C_13=pd.concat([Synthetic_data_4,Synthetic_data_5],axis=1,join_axes=[S
ynthetic_data_4.index])
```

#Synthetic Data generation for all irrigation events during the afternoon (Shared mean and covariance matrix between classes)

```
dataset1=dataset.rename(index=str,columns={"SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.rename(index=str, columns={"START_TIME_CLASS": "C"})
```

```
dataset3 = dataset2.loc[lambda df: df.C == 3, :]
dataset4 = dataset3.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Construct dataset
X1,y1= make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=1000,
n_features=4, n_classes=3, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch',
'EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4 = Synthetic_data_4.assign(**{'START_TIME_CLASS': 3})
Synthetic_data_5=pd.DataFrame(y1,columns=['SOIL_WETTING_EFFICIENCY
_INDEX_CLASS'])
result_C_14=pd.concat([Synthetic_data_4,Synthetic_data_5],axis=1,join_axes=[S
ynthetic_data_4.index])
```

#Synthetic Data generation for all irrigation events during the evening (Shared mean and covariance matrix between classes)

```
dataset1=dataset.rename(index=str,columns={"SOIL_WETTING_EFFICIENCY
_INDEX_CLASS": "B"})
dataset2 = dataset1.rename(index=str, columns={"START_TIME_CLASS": "C"})
dataset3 = dataset2.loc[lambda df: df.C == 4, :]
dataset4 = dataset3.drop(['C','B'], axis = 1)
xe = dataset4.as_matrix(columns=None)
mean_xe = np.mean(xe, axis=0)
cov_xe = np.cov(xe, rowvar=0)
from sklearn.datasets import make_gaussian_quantiles
#Construct dataset
X1,y1=make_gaussian_quantiles(cov=cov_xe, mean=mean_xe, n_samples=1000,
n_features=4, n_classes=3, random_state=1)
Synthetic_data_4=pd.DataFrame(X1,columns=['ETO_mm/day','AVG_WIND_SP
EED_MPH','PRECIPITATION_inch',
'EFFECTIVE_IRRIGATION_TIME_min'])
Synthetic_data_4 = Synthetic_data_4.assign(**{'START_TIME_CLASS': 4})
Synthetic_data_5=pd.DataFrame(y1,columns=['SOIL_WETTING_EFFICIENCY
_INDEX_CLASS'])
result_C_15=pd.concat([Synthetic_data_4,Synthetic_data_5],axis=1,join_axes=[S
ynthetic_data_4.index])
```

#Combining all the generated data into a single Data frame

```
result12 = result_C_12.append(result_C_13, ignore_index=True)
result13 = result12.append(result_C_14, ignore_index=True)
result14 = result13.append(result_C_15, ignore_index=True)
final_result = result11.append(result14, ignore_index=True)
```

```
#Training RBF SVM for different values of train:test split on the synthetic data
    from sklearn.model_selection import train_test_split
    from sklearn.model_selection import GridSearchCV
    from sklearn.metrics import accuracy_score
    X = final_result.drop("SOIL_WETTING_EFFICIENCY_INDEX_CLASS", axis
    =1)
    y = final_result['SOIL_WETTING_EFFICIENCY_INDEX_CLASS']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state
    = 42)
    import numpy as np
    import matplotlib.pyplot as plt
    from matplotlib import style
    style.use("ggplot")
    from sklearn import svm
    b = X_train.as_matrix(columns = None)
    c = y_train.as_matrix(columns = None)
    from sklearn import svm
    # Grid Search
    # Parameter Grid
    gammas = np.linspace(0.0, 1.0, num=100)
    param_grid = {'C': [100], 'gamma': gammas} #The value of penalty parameter is
                                                varied from 1 to 100 in terms of 10
                                                units. Best Gamma is chosen for each
                                                case.
    # Make grid search classifier
    clf_grid = GridSearchCV(svm.SVC(), param_grid, verbose=1)
    # Training the classifier
    clf_grid.fit(b,c)
    print("Best Parameters:\n", clf_grid.best_params_)
    print("Best Estimators:\n", clf_grid.best_estimator_)
    b1 = X_test.as_matrix(columns = None)
    c1 = y_test.as_matrix(columns = None)
    result = clf_grid.predict(b1)
    accuracy_score(c1, result) #Gives the accuracy score on testing data

#Training RBF SVM on the entire synthetic data generated and training it on the real
dataset
    from sklearn.model_selection import train_test_split
    from sklearn.model_selection import GridSearchCV
    from sklearn.metrics import accuracy_score
    X_train = final_result.drop('SOIL_WETTING_EFFICIENCY_INDEX_CLASS',
    axis = 1)
    y_train = final_result['SOIL_WETTING_EFFICIENCY_INDEX_CLASS']
    style.use("ggplot")
    from sklearn import svm
```

```
b = X_train.as_matrix(columns = None)
c = y_train.as_matrix(columns = None)
X_test = dataset.drop('SOIL_WETTING_EFFICIENCY_INDEX_CLASS', axis =
1)
y_test = dataset['SOIL_WETTING_EFFICIENCY_INDEX_CLASS']
b1 = X_test.as_matrix(columns = None)
c1 = y_test.as_matrix(columns = None)
from sklearn import svm
#Grid Search
#Parameter Grid
gammas = np.linspace(0.0, 1.0, num=100)
param_grid1 = {'C': [100], 'gamma': gammas}
#Make grid search classifier
clf_grid1 = GridSearchCV(svm.SVC(), param_grid1, verbose=1)

#Train the classifier
clf_grid1.fit(b,c)
#clf = grid.best_estimator_()
print("Best Parameters:\n", clf_grid1.best_params_)
print("Best Estimators:\n", clf_grid1.best_estimator_)
result1 = clf_grid1.predict(b1)
accuracy_score(c1, result1)
```