Performance Comparison of Machine Learning Models for Predicting a One-day Weather Forecast (2022)

Christopher M. Frutos*, Andrew B. Garcia*, Kayleigh E. Movalli*

*Virginia Polytechnic Institute and State University – Graduate School, cfrutos@vt.edu, agarcia1296@vt.edu, kayleighm@vt.edu

***Abstract*—A novel approach to predicting weather by experimenting with two machine learning models' accuracy as an appropriate replacement for the current process of prediction.**

The two models being decision trees and Linear Regressions are supervised learning models, so they will train on a labeled dataset. The dataset was provided by the U.S. National Oceanic and Atmospheric Administration, and it is meteorological datasets containing precipitation rates that go as far back as the year 1900. A dataset this large should provide enough information to accurately predict the upcoming weather, which makes for an optimal situation to analyze the performance of these two learning models.

***Index Terms*— Machine Learning, Decision Tree, Linear Regression, Learning Model**

I. INTRODUCTION

Being able to accurately predict weather can save lives of many, such as when and where a hurricane will land. The time and accuracy output of weather prediction allows for evacuation orders or families to act on their life-preparedness plan to minimize damage. With current methods of predicting weather being massive amounts of fluid dynamics and typically on supercomputers [1], we hope to implement a simple and straightforward model that can be comparable to the current method. We are working off the knowledge that the current accuracy rates of one-day forecasts are correct within 2-2.5 degrees- meaning that if the forecasted high is 80 Fahrenheit, it could be 78-83 degrees Fahrenheit.

Because there has been a rise in the use of Artificial Intelligence and Machine Learning (and the projected market increasing 800% from 2016 to 2022 [2]) it is undeniable that there are practical applications for this technology, and we believe that weather forecasting may be one of these applications. In our study we will be exploring the two most common algorithms, Decision Tree and Linear Regression, to test their predictive capabilities for weather forecasts. We think these models would be best suited for this scenario due to the nature of their predictive methods. Decision Trees are easy to manipulate and less complex than other models, while Linear Regression models have the added advantage of being fast. Using information provided from the weather station USW00023066, we will train both models and compare the results.

II. DATA PREPARATION & STATION INFORMATION

A. Data Origins

The weather station, named USW00023066, provided weather data for the dates between January 1st, 1900, and March 2nd, 2022. USW00023066 is a weather station located in Grand Junction, Colorado. It was established in 1895 and is still running through year 2022. As previously mentioned, this data was recorded by the U.S. National Oceanic and Atmospheric Administration.

B. Data Quality Reports

Using the raw data provided, the following quality report (Table 1) was generated pre-data preparation and will serve as a comparison for data modifications and validation for the classifiers. When looking at the dataset before modification, we can see that there is a large percentage of the data missing, namely in the MFLAG1, Q_FLAG1, SFLAG1, and VALUE2 fields. We opted to not use these fields for this reason.

stat	ID	DATE	ELEMENT	VALUE1	MFLAG1	Q_FLAG1	SFLAG1	VALUE2
cardinality	1	44595	60	2056	3	4	7	1
mean	N/A	19746016	N/A	154.1893	N/A	N/A	N/A	2400
median	N/A	19810909	N/A	41	N/A	N/A	N/A	2400
n_at_median	N/A	15	N/A	678	N/A	N/A	N/A	75845
mode	N/A	19930108	N/A	0	N/A	N/A	0	2400
n_at_mode	N/A	24	N/A	130615	N/A	N/A	0	75845
stddev	N/A	319463.4	N/A	368.3446	N/A	N/A	N/A	0
min	USW00023066	19000101	ACMH	-378	N/A	N/A	0	2400
max	USW00023066	20220302	WV07	9999	N/A	N/A	Z	2400
nzero	0	0	0	130615	0	0	0	0
nmissing	0	0	0	0	391672	418806	0	343077

Table 1. The data report on the original data set.

In order to make this data useful, some modifications had to be made to the original data set. Under the value "ELEMENT" all original features, sixty in total, for each day were stored in one column. This made the data difficult to read and impossible to train on as each date would be repeated sixty times. In order to make the data appropriate for our models, we used a python script to extract the features and separate them by date into their own columns so that the data set contained one row per day.

After the data was sorted, we ran a quality report in order to determine whether it was ready to train our models. The report returned flags as the new features also contained missing data. To make our data as concise and accurate as possible, we created another python script to traverse the data set and calculate how much data was missing from the features. If more than 10% was missing, the feature was dropped from the data set, and if less than 10% was missing then they were replaced with the average of the feature. The following elements were dropped from the data set:

DAPR, MDPR, WT01, WT16, GAHT, WT03, EVAP, WDMV, WT09, WT06, DAEV, DAWM, MDEV, MDWM, WT04, WT08, WESD, WT07, WSFG, FRGT, FRTH, THIC, ACMH, ACSH, PSUN, TSUN, WDFM, WT02, PGTM, WDFG, WDF1, WSF1, AWND, FMTM, WT17, WT10, WDF2, WDF5, WSF2, WSF5, WT19, WT11, WT13, WT22, TAVG, WV03, WV07, WV01, WT21

After the missing data was refined, four columns were added to the data set. The first two are PRECIPFLAG and PRECIPAMT. The first is used as a flag to signal if there was rain or snow that particular day and the second was calculated by converting the PRCP and SNOW data into inches and adding them together (assuming 8in of snow is 1in of rain). Then the columns were added to the previous days data as our target columns called NEXTDAYPRECIPFLAG and NEXTDAYPRECIPAMT.

With this in place, the data looks like this:

stat	DATE	TMAX	TMIN	PRCP	SNOW	SNWD	PRECIP FLAG	PRECIP AMT	NEXT DAY PRECIP FLAG	NEXT DAY PRECIP AMT
cardinality	44595	122	143	121	72	24	2	656	2	656
mean	19606803	186.3458	45.65809	6.073519	1.487834	5.580255	0.204664	0.03122	0.204664	0.03122
median	19610213	189	44	0	0	0	0	0	0	0
n at median	1	559	769	35674	42124	41309	35468	35468	35468	35468
mode	N/A	333	0	0	0	0	0	0	0	0
n at mode	N/A	858	1044	35674	42124	41309	35468	35468	35468	35468
stddev	352554.3	117.2802	98.9568	21.31176	9.545402	25.01126	0.403461	0.10929	0.403461	0.10929
min	19000101	-167	-306	0	0	0	0	0	0	0
max	20220302	417	256	475	356	457	1	2.381891	1	2.381891
nzero	0	305	1044	35674	42124	41309	35468	35468	35468	35468
nmissing	0	0	0	0	0	0	0	0	0	0

Table 2. The data set after data preparation.

With this data quality report, we were able to determine that the data is ready for training as there are no missing values and all features are represented in numeric values without unnatural outliers.

C. Analysis of Data Preparation

The prepped data was taken to Tableau for additional analysis. A histogram was made for both target values, the first of those being NEXTDAYPRECIPFLAG in Figure 1. The histogram for next day precipitation flag shows us that precipitation occurred on average every one in four days, being that it rained or snowed nine thousand out of thirty-five thousand days.

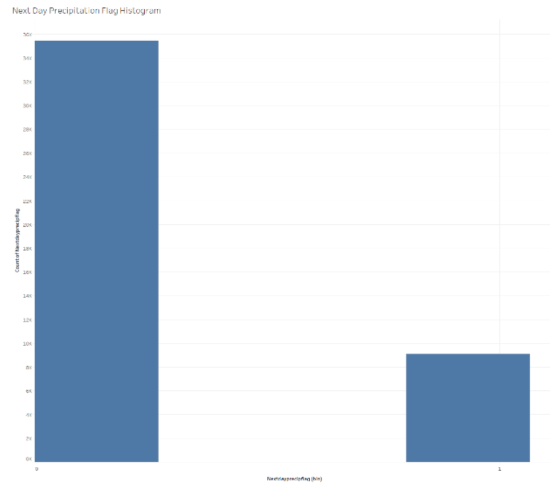

Fig 1. Histogram of Precipitation Amount

The histogram for next day precipitation amount (Figure 2) was binned using a range value range of 0.8 inches. It can be seen that a sharp exponential decay occurs where there are more counts in the 0 to 0.1 range and less counts in the larger precipitation bins.

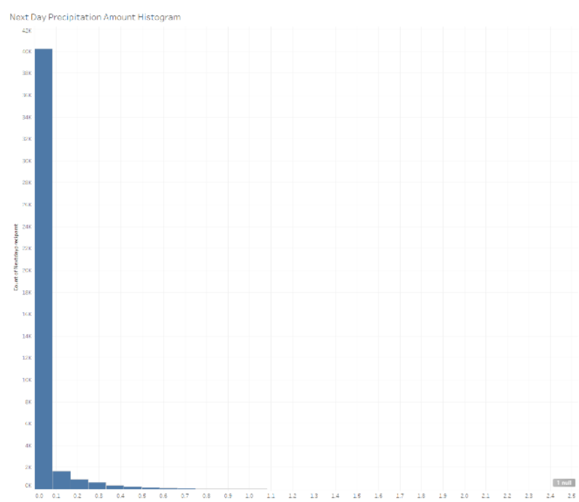

Fig 2. Histogram of Precipitation Flag

IV. Decision Tree Model Results

A. With Data from The Day Of

Per the setup of our experiment, we decided to test two different criteria for our decision tree classifiers, Gini and Entropy. We trained the decision tree models on the features for the current day's weather information to predict if there will be precipitation on the next day (NEXTDAYPRECIPFLAG). In order to being training, we split the data set into 70% testing and 30% training data, using a random seed value to make the splitting consistent. After splitting we used sci-kit learn to fit the model to our training data and tried various different tree depths to see what scored the best.

For Entropy and Gini, we found that a tree depth of 6 decisions was best for both models because it provided the highest true positive rate (TPR) on testing data without overfitting our model to the training data. Tree depths of 7 and 8 started showing an increase in training scores, but a decrease in testing scores.

Both decision trees happen to prioritize splitting the data the same way in the first few decision nodes. The starting node splits the data based on precipitation (PRCP) values that are less than or equal to 1.5 (0.15mm). Following the decision tree to the left where the initial node is true, a decision is made based on minimum temperature (TMIN) values less than or equal to 175 (17.5 degrees C). In 6300 cases where the initial node is false, meaning it had rained that day, the decision tree checks for precipitation (PRCP) values less than or equal to 19 (1.9mm). Looking further down the decision tree, values with data that are larger than 1.9 inches of rain are often flagged to set next day precipitation to be true. The decision trees for the same-day Entropy and Gini models are shown in Figures 3 and 4 respectively.

Fig 3. Entropy Classification

Fig 4. Gini Classification

B. With Data from One Day Prior

As a secondary test for accuracy, we decided to train another version of the models on two previous days' worth of data instead of just one. We followed the same process of collecting training scores then testing and calculating the TPR. It was found that a tree depth of 3 yielded the best TPR scores for testing data for our Entropy decision tree and a tree depth of 4 for our Gini decision tree.

Same as the previous experiment, both decision tree classifiers prioritize the same top node decisions, precipitation (PRCP) less than or equal to 1.5 (0.15mm). However, because of the introduction of new data, a new way of splitting the data can be made compared to the previous method. The next two decisions are made based on if the previous day precipitation amount (PREV_PRECIP_AMT) is less than or equal to 0.029 (0.029in). Similar to the previous model, these models also make a decision based on if the precipitation (PRCP) is less than or equal to 19 (1.9mm)

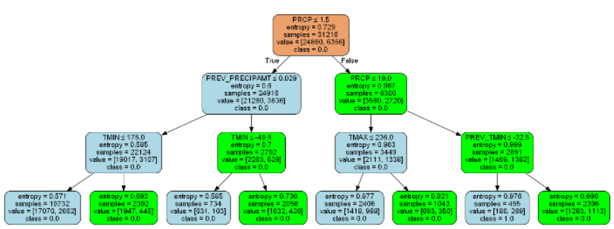

Fig 5. Entropy Classification – Using Prior Days Data.

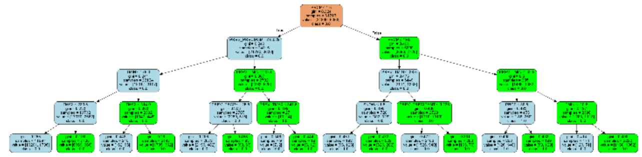

Fig 6. Gini Classification – Using Prior Days Data.

This concludes the results of our decision tree model tests.

V. Linear Regression Model results

A. With Data from The Day Of

The way we test for accuracy on Linear Regression models is different than through the decision tree model. Instead of using the TPR function, we must calculate something known as the Mean Square Error (MSE), which essentially shows us the rate of error in the model. If a Linear Regression model has a high accuracy, the MSE number will be very small. We show how we calculate the MSE in the "Equations" sections (section VI).

<u>Linear Regression Using Same Day Data:</u>
Coefficients: [4.37080542e-09 -2.68430075e-04 2.90667505e-04 -2.90985261e-04
-3.94199171e-04 8.71916380e-05 3.27823152e-02 1.46123133e-01]
Intercept: -0.0269983740417523 43
Mean squared error (MSE): 0.011074936654111519

<u>RidgeCV using Same Day Data:</u>
Coefficients: [-7.45058060e-09 -2.68440078e-04 2.90682639e-04 3.45973931e-05
1.22183387e-05 8.72534388e-05 3.27844465e-02 6.34360334e-02]
Intercept: 0.20479000178277015
Mean squared error (MSE): 0.011076925956234253

B. With Data from One Day Prior

Again, in accordance with the setup of the experiments, we also trained another Linear Regression model on the previous days' data as we did with the decision tree model.

<u>Linear Regression Using Prior Days Data:</u>
Coefficients: [4.18464464e-09 -2.05072963e-04 4.93072328e-04 -2.51318927e-04
-4.07520307e-04 2.66942511e-05 3.58396298e-02 1.50285101e-01
4.55928473e-10 -1.00799339e-04 -1.69316453e-04 -1.48120056e-03
-1.61224883e-03 4.65428710e-05 -8.55457825e-03 3.58542729e-01]
Intercept: -0.025714000023889003
Mean squared error (MSE): 0.01104011055741799

<u>RidgeCV Using Prior Days Data:</u>
Coefficients: [5.18048182e-09 -2.05421065e-04 4.93566685e-04 3.36869451e-04
3.26043195e-04 2.68187751e-05 3.57200452e-02 1.14825630e-03

1.45519152e-09 -1.00702288e-04 -1.69505302e-04 -7.01598129e-05
1.51894062e-04 4.65053749e-05 -8.50789023e-03 7.83732406e-05]

Intercept: -0.0647920633331147

Mean squared error (MSE): 0.011043062842543814

This concludes our testing and calculations for our Linear Regression model.

VI. Equations

A. True Positive Rate

The following equation was used to determine the performance of the models using Gini and Entropy classification. The result found is the (1) True Positive Rate (TPR). The higher the TPR, the better the criteria performed; it can be described as the Correctly Predicted Test Set divided by the Test Set.

$$TPR = \frac{|TestSetCorrectlyPredicted|}{|TestSet|} \quad (1)$$

B. Mean Square Error

The Mean Square Error (MSE) (2) equation is another equation used to measure performance in the regression predictive model used in predicting the next day's precipitation amount. Just like the TPR (1), the performance of the model is high if the value of the MSE (2) is high.

$$\text{MSE} = \frac{1}{|TestSet|} \sum_{1}^{|TestSet|} (NEXTDAYPRECIPAMT - MODELOUTPUT)^2 \quad (2)$$

VII. Analysis of Results

A. Decision Tree Analysis

When comparing the accuracy of the four decision tree models it is clear to us that the Gini classification model scored higher than the entropy classification model, but not by a wide margin. On our same-day data set, the Gini classification only scored 0.00008 higher, and on the one-day prior the Gini scored 0.002 higher than the entropy model. Overall, both models had a slightly better accuracy rate when incorporating one-day prior's data than just using same-day data. We also observed that the training set performed better than the testing set, which is to be expected because the training decision tree is using data it's trained on, while the latter is using unfamiliar data. The same trend between training vs. testing test scores is true for the same-day data entropy classification model using TPR (1); the training score is slightly higher than the testing score. Similarly, in the same-day data classification, this trend is also true when data was used from the day prior using both Gini and entropy classification: the training score was better than the testing score, but only by a small amount.

Using this information, we wanted to determine what the most optimal tree depth would be for peak performance. In order to this, we wrote a script to calculate the training and testing times of various tree depth lengths for both the same-day data frame, and the one-day prior data frame- and we did this for both classification types. This is shown in Table 4.

Table 3. Estimated training/testing scores for various tree depths.

Tree Depth	Entropy Training Score	Entropy Test Score	Gini Training Score	Gini Test Score	Entropy Training Score Prior Day Data	Entropy Test Score Prior Day Data	Gini Training Score Prior Day Data	Gini Test Score Prior Day Data
3	0.797155	0.793557	0.797155	0.793557	0.799045	0.793408	0.799077	0.793183
4	0.798469	0.793707	0.798469	0.793707	0.799077	0.793183	0.80116	0.794902
5	0.799942	0.794006	0.800006	0.79408	0.800647	0.792361	0.802537	0.794678
6	0.802954	0.7955	0.803338	0.796323	0.802986	0.793034	0.804523	0.792511
7	0.805709	0.793931	0.806157	0.794678	0.805292	0.791913	0.807535	0.789222
8	0.809104	0.794454	0.810418	0.794828	0.810386	0.788848	0.812948	0.791763

When we analyze this table and prioritize testing scores, we can determine that the highest performing decision tree model is the Gini classification using one-day prior's data, with a tree depth of 4. This is represented in [2, 8]. We can also acknowledge that this is only better by a very small margin, as overall the scores are quite similar.

B. Modeling Regression Analysis

When analyzing our Linear Regression models, using same day or prior days data, our MSE (2) score goes down as the addition of prior days data is included, which is good. The same can be said for the RidgeCV model. Comparing the same day data models versus the use of prior days data, it can be seen that the largest coefficients increase in the Linear Regression model and significantly decreases in the RidgeCV model. The Linear Regression Model Using Prior Days Data scored the lowest MSE (2) compared to other methods tried. This method beats RidgeCV Using Prior Days data by a value of 0.000002, but the RidgeCV has a much lower largest coefficient compared to the Linear Regression Model. Because of these reasons we can say that the Linear Regression Using Prior Days Data yeilded the best results.

Table 4. Modeling Regression Analysis Table

Measurement	Linear Regression Using Same Days Data	Linear Regression Using Prior Days Data	RidgeCV Using Same Day Data	RidgeCV Using Prior Days Data
MSE	0.011074937	0.011040111	0.0110769	0.0110431
Largest Coeff	1.46E-01	3.59E-01	6.34E-02	7.84E-05

VIII. Conclusion

In conclusion, when reviewing all of our outputs, we were able to select one learning model for each approach that performed higher than the others. The higher performing models did not significantly beat the others, but overall did score better.

Of our decision trees, the highest performing model was the Gini classification with a tree depth of 4. This was proven in Table 3, when we calculated the resulted training/testing scores from different tree depths. It is important to note that though the differences in performance are miniscule, when dealing with larger amounts of data, the difference in accuracy is greatly affected even with a small change in performance. If we proceeded with predicting weather and chose to go with decision trees, this would be the model we would select.

Of our Linear Regression models, we were able to deduce the highest performance model based on the lowest MSE (2). The Linear Regression model, when using one-day prior's data, scored the lowest MSE (2) out of the four different model and method combinations that we tried. This made it clear to us that going forward, this is the model we would use if we were going the Linear Regression method.

References

[1] S. Lazzaro, "Machine Learning's rise, applications, and challenges," *VentureBeat*, 21-Jun-2021. [Online]. Available: https://venturebeat.com/2021/06/21/machine-learnings-rise-applications-and-challenges/#:~:text=In%20the%20first%20quarter%20of,to%20%248.81%20billion%20by%202022. [Accessed: 21-Mar-2022].

[2] J. Bowler, "Is weather forecasting getting less accurate?," *Cosmos*, 04-Feb-2022. [Online]. Available: https://cosmosmagazine.com/earth/climate/weather-forecasting-less-accurate/. [Accessed: 21-Mar-2022].

[3] NCEI.Monitoring.info@noaa.gov, "Climate at a glance," *National Climatic Data Center*. [Online]. Available: https://www.ncdc.noaa.gov/cag/city/time-series/USW00023066. [Accessed: 21-Mar-2022].

[4] *Index of /*. [Online]. Available: https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/readme.txt. [Accessed: 22-Mar-2022].

IX. Appendix

A. *proj1.py*

```
import pandas as pd
import os
from tqdm import tqdm
#%% Setup
# Create Full Path - This is the OS agnostic way of doing so
dir_name = os.getcwd()
filename = 'USW00023066.csv'
full_path = os.path.join(dir_name, filename)

#
# Create the Main Data Frame
#
data_headers = ['ID', 'DATE', 'ELEMENT', 'VALUE1', 'MFLAG1', 'Q_FLAG1', 'SFLAG1', 'VALUE2']
df_main = pd.read_csv(full_path, names = data_headers) # read Excel spreadsheet
print('File {0} is of size {1}'.format(full_path, df_main.shape))


#%% Generating a Report for RAW
from utils_project1 import StatsReport

labels = df_main.columns
report = StatsReport()

# Create a simple data set summary for the console
for thisLabel in tqdm(labels): # for each column, report stats
    thisCol = df_main[thisLabel]
    report.addCol(thisLabel, thisCol)

print(report.to_string())
report.statsdf.to_excel("Quality_Report_Before_Prep.xlsx")

#%%
def get_unique_column_values(df):
    """
    Identifying Unique Values of each Column in DF
    Output is a Dictionary of each Column
    """
    headers_unique = {}
    for label in tqdm(df.columns):
        headers_unique[label] = df[label].unique()
    #pbar.close()
    return headers_unique

headers_unique = get_unique_column_values(df_main)
print(f"List of Dates: {headers_unique['DATE']}")

#%% Data Preperation - THIS TAKES SEVERAL MINUTES

def prep_data(df, df_out, headers_unique):
    """
    Extract Values for Elements and insert into df_prep
    """
    index_ = 0
    for date in tqdm(headers_unique['DATE']):
        date_idx = df['DATE'] == date
        df_by_date = df[date_idx]
        df_out.loc[index_, 'DATE'] = date
```

```
        for idx in
df_by_date['ELEMENT'].index:
            df_out.loc[index_,
df_by_date['ELEMENT'][idx]] =
df_by_date['VALUE1'][idx]
        index_ = index_+1


df_prep = pd.DataFrame(columns = ['DATE',
*headers_unique['ELEMENT']])
prep_data(df_main, df_prep, headers_unique)

#%% Create Target Columns
#
# Create Columns - PRECIPFLAG and PRECIPAMT
# Create Target Columns - NEXTDAYPRECIPFLAG
and NEXTDAYPRECIPAMT
#
for idx in tqdm(df_prep.index):
    rain = df_prep['PRCP'][idx] # in tenths
of mm
    snow = df_prep['SNOW'][idx]
    if (rain or snow) > 0:
        df_prep.loc[idx, 'PRECIPFLAG'] = 1 #
It rained/snowed
        df_prep.loc[idx, 'PRECIPAMT'] =
0.0393701*(rain/10) + (0.0393701*snow)/8 #
result is in inches
    else:
        df_prep.loc[idx, 'PRECIPFLAG'] = 0 #
It did not rain/snow
        df_prep.loc[idx, 'PRECIPAMT'] = 0
    if idx > 0:
        df_prep.loc[idx-1,
'NEXTDAYPRECIPFLAG'] = df_prep.loc[idx,
'PRECIPFLAG']
        df_prep.loc[idx-1,
'NEXTDAYPRECIPAMT'] = df_prep.loc[idx,
'PRECIPAMT']

#%% Generating a Report
labels_post = df_prep.columns
report_post = StatsReport()

# Create a simple data set summary for the
console
for thisLabel in tqdm(labels_post): # for
each column, report stats
    thisCol = df_prep[thisLabel]
    report_post.addCol(thisLabel, thisCol)

#print(report.to_string())
report_post.statsdf.to_excel("Quality_Repor
t_Post_Prep.xlsx")

#%% Sus out Bad Elements
from utils_project1 import
replace_missing_values_avg

df_final = df_prep.copy()

temp_report_df = report_post.statsdf

for element in tqdm(labels_post):
    if temp_report_df[element][10] >
len(df_prep)*0.1: # Weeding out Elements that
have more than 10% of missing values
        df_final = df_final.drop(columns =
[element])
        print('ELEMENT Dropped:', element)
    elif temp_report_df[element][10] <
len(df_prep)*0.1:
        if element == 'NEXTDAYPRECIPAMT' or
element == 'NEXTDAYPRECIPFLAG':
            avg_value = 0
        else:
            avg_value =
temp_report_df[element][1]

replace_missing_values_avg(df_final, element,
avg_value)

#%% Run Quality Report and Output Data to
Excel
df_final.to_excel('Weather_Data_Final.xlsx'
)

labels_final = df_final.columns
report_final = StatsReport()

# Create a simple data set summary for the
console
for thisLabel in tqdm(labels_final): # for
each column, report stats
    thisCol = df_final[thisLabel]
    report_final.addCol(thisLabel, thisCol)

#print(report.to_string())
#report_final.statsdf.to_excel("Quality_Rep
ort_Final.xlsx")

#%% Setting up Training Data
# Data
feature_names =
df_final.columns.drop(['NEXTDAYPRECIPFLAG','N
EXTDAYPRECIPAMT'])
X = df_final[feature_names]

# Target
y_precip_flag = df_final.loc[:,
['NEXTDAYPRECIPFLAG']]
```

```
labels = y_precip_flag['NEXTDAYPRECIPFLAG'].unique()

#%% Create Testing and Training data for Precip Flag
from sklearn.model_selection import train_test_split
import numpy as np
from utils_project1 import writegraphtofile, get_true_positive
from sklearn import tree

X_train_flag, X_test_flag, y_train_flag, y_test_flag = train_test_split(X, y_precip_flag, test_size=0.3,

train_size=0.7, random_state=1996,

shuffle=True, stratify=None)
#%% Create Decision Tree - Entropy
score_df = pd.DataFrame()
idx = 0
for i in range(3,9):
    clf_entropy = tree.DecisionTreeClassifier(criterion = "entropy", max_depth = i)
    clf_entropy = clf_entropy.fit(X_train_flag, np.array(y_train_flag['NEXTDAYPRECIPFLAG']))

    training_score = clf_entropy.score(X_train_flag, y_train_flag['NEXTDAYPRECIPFLAG'])
    true_positive_entropy, matrix_df_entropy = get_true_positive(clf_entropy, X_test_flag, y_test_flag)

    score_df.loc[idx,'Tree Depth'] = i
    score_df.loc[idx,'Entropy Training Score'] = training_score
    score_df.loc[idx,'Entropy Test Score'] = true_positive_entropy

    idx = idx+1

# Measure Performance
print("Entropy Training set score = ", clf_entropy.score(X_train_flag, y_train_flag['NEXTDAYPRECIPFLAG']))
print("Entropy Test set score = ", clf_entropy.score(X_test_flag, y_test_flag['NEXTDAYPRECIPFLAG']))
print('Entropy True Positive Rate = ', true_positive_entropy)

# Create Graphic
path_name = os.path.join(dir_name, "Weather_Data_DecisionTree_Entropy_NextDayPrecipFlag.png")
writegraphtofile(clf_entropy, feature_names, (str(labels[0]), str(labels[1])), path_name)
tree.export_graphviz(clf_entropy)

#%% Create Decision Tree - Gini
idx = 0
for i in range(3,9):
    clf_gini = tree.DecisionTreeClassifier(criterion = "gini", max_depth = i)
    clf_gini = clf_gini.fit(X_train_flag, np.array(y_train_flag['NEXTDAYPRECIPFLAG']))

    training_score = clf_gini.score(X_train_flag, y_train_flag['NEXTDAYPRECIPFLAG'])
    true_positive_gini, matrix_df_gini = get_true_positive(clf_gini, X_test_flag, y_test_flag)

    score_df.loc[idx,'Gini Training Score'] = training_score
    score_df.loc[idx,'Gini Test Score'] = true_positive_gini

    idx = idx+1

# Measure Performance
print("Gini Training set score = ", clf_gini.score(X_train_flag, y_train_flag['NEXTDAYPRECIPFLAG']))
print("Gini Test set score = ", clf_gini.score(X_test_flag, y_test_flag['NEXTDAYPRECIPFLAG']))
print('Gini True Positive Rate = ', true_positive_gini)


# Create Graphic
path_name = os.path.join(dir_name, "Weather_Data_DecisionTree_Gini_NextDayPrecipFlag.png")
writegraphtofile(clf_gini, feature_names, (str(labels[0]), str(labels[1])), path_name)
tree.export_graphviz(clf_gini)

#%% Linear Regression
# Target
y_precip_amt = df_final.loc[:, ['NEXTDAYPRECIPAMT']]
labels_amt = y_precip_amt['NEXTDAYPRECIPAMT'].unique()
```

```
# Split training/testing data by precip amt
X_train_amt, X_test_amt, y_train_amt, y_test_amt = train_test_split(X, y_precip_amt, test_size=0.3,

train_size=0.7, random_state=1996,

shuffle=True, stratify=None)

from sklearn.linear_model import LinearRegression
from utils_project1 import get_mse
from sklearn.metrics import mean_squared_error

linreg_model = LinearRegression().fit(X_train_amt, np.array(y_train_amt['NEXTDAYPRECIPAMT']))
linreg_model.score(X_test_amt, y_test_amt)

# Testing score
lin_model_pred_test = linreg_model.predict(X_test_amt)
mean_squared_error(y_test_amt, lin_model_pred_test)

print('Coefficients:', linreg_model.coef_)
print('Intercept:', linreg_model.intercept_)
print('Mean squared error (MSE): %.2f'
      % mean_squared_error(y_test_amt, lin_model_pred_test))

#%% RidgeCV
from sklearn.model_selection import cross_val_score, RepeatedKFold
from sklearn.linear_model import RidgeCV

ridge_model = RidgeCV().fit(X_train_amt, np.array(y_train_amt['NEXTDAYPRECIPAMT']))

# Testing score
ridge_model_pred_test = ridge_model.predict(X_test_amt)
mean_squared_error(y_test_amt, ridge_model_pred_test)

print('Coefficients:', ridge_model.coef_)
print('Intercept:', ridge_model.intercept_)
print('Mean squared error (MSE): %.2f'
      % mean_squared_error(y_test_amt, ridge_model_pred_test))
#%% Measure Mean Square Error

print("Mean Square Error = ", get_mse(linreg_model, X_test_amt, y_test_amt))
print("Mean Square Error = ", get_mse(ridge_model, X_test_amt, y_test_amt))

#%% Use of Prior Days Data - THIS TAKES SEVERAL MINUTES
from utils_project1 import create_prior_day_data_df
feature_names = df_final.columns.drop(['NEXTDAYPRECIPFLAG','NEXTDAYPRECIPAMT'])
df_prior_day = df_final.copy()
#Create the DF for prior day
create_prior_day_data_df(df_prior_day, feature_names)

#%% Setting up Training Data
# Data
feature_names = df_prior_day.columns.drop(['NEXTDAYPRECIPFLAG','NEXTDAYPRECIPAMT'])
X = df_prior_day[feature_names]

# Target
y_precip_flag = df_prior_day.loc[:, ['NEXTDAYPRECIPFLAG']]
labels = y_precip_flag['NEXTDAYPRECIPFLAG'].unique()

#%% Create Testing and Training data for Precip Flag

X_train_flag, X_test_flag, y_train_flag, y_test_flag = train_test_split(X, y_precip_flag, test_size=0.3,

train_size=0.7, random_state=1996,

shuffle=True, stratify=None)


#%% Create Decision Tree - Entropy
idx = 0
for i in range(3,9):
    clf_entropy = tree.DecisionTreeClassifier(criterion = "entropy", max_depth = i)
    clf_entropy = clf_entropy.fit(X_train_flag, np.array(y_train_flag['NEXTDAYPRECIPFLAG']))

    training_score = clf_entropy.score(X_train_flag, y_train_flag['NEXTDAYPRECIPFLAG'])
    true_positive_entropy, matrix_df_entropy = get_true_positive(clf_entropy, X_test_flag, y_test_flag)
```

```
        score_df.loc[idx,'Entropy Training Score Prior Day Data'] = training_score
        score_df.loc[idx,'Entropy Test Score Prior Day Data'] = true_positive_entropy

        idx = idx+1

    # Measure Performance
    print("Entropy Training set score = ", clf_entropy.score(X_train_flag, y_train_flag['NEXTDAYPRECIPFLAG']))
    print("Entropy Test set score = ", clf_entropy.score(X_test_flag, y_test_flag['NEXTDAYPRECIPFLAG']))
    print('Entropy True Positive Rate = ', true_positive_entropy)

    # Create Graphic
    path_name = os.path.join(dir_name, "Weather_Data_DecisionTree_Entropy_NextDayPrecipFlag_PriorDay.png")
    writegraphtofile(clf_entropy, feature_names, (str(labels[0]), str(labels[1])), path_name)
    tree.export_graphviz(clf_entropy)

    #%% Create Decision Tree - Gini
    idx = 0
    for i in range(3,9):
        clf_gini = tree.DecisionTreeClassifier(criterion = "gini", max_depth = i)
        clf_gini = clf_gini.fit(X_train_flag, np.array(y_train_flag['NEXTDAYPRECIPFLAG']))

        training_score = clf_gini.score(X_train_flag, y_train_flag['NEXTDAYPRECIPFLAG'])
        true_positive_gini, matrix_df_gini = get_true_positive(clf_gini, X_test_flag, y_test_flag)

        score_df.loc[idx,'Gini Training Score Prior Day Data'] = training_score
        score_df.loc[idx,'Gini Test Score Prior Day Data'] = true_positive_gini

        idx = idx+1

    # Measure Performance
    print("Gini Training set score = ", clf_gini.score(X_train_flag, y_train_flag['NEXTDAYPRECIPFLAG']))
    print("Gini Test set score = ", clf_gini.score(X_test_flag, y_test_flag['NEXTDAYPRECIPFLAG']))
    print('Gini True Positive Rate = ', true_positive_gini)

    # Create Graphic
    path_name = os.path.join(dir_name, "Weather_Data_DecisionTree_Gini_NextDayPrecipFlag_PriorDay.png")
    writegraphtofile(clf_gini, feature_names, (str(labels[0]), str(labels[1])), path_name)
    tree.export_graphviz(clf_gini)

    #%% Exporting Score DF
    score_df.to_excel('Decision Tree Scores.xlsx')

    #%% Linear Regression - Using Prior Day Data
    # Target
    y_precip_amt = df_final.loc[:, ['NEXTDAYPRECIPAMT']]
    labels_amt = y_precip_amt['NEXTDAYPRECIPAMT'].unique()

    # Split training/testing data by precip amt
    X_train_amt, X_test_amt, y_train_amt, y_test_amt = train_test_split(X, y_precip_amt, test_size=0.3,
                                                                        train_size=0.7, random_state=1996,
                                                                        shuffle=True, stratify=None)

    from sklearn.linear_model import LinearRegression
    from utils_project1 import get_mse

    linreg_model = LinearRegression().fit(X_train_amt, np.array(y_train_amt['NEXTDAYPRECIPAMT']))
    linreg_model.score(X_test_amt, y_test_amt)

    # Testing score
    lin_model_pred_test = linreg_model.predict(X_test_amt)
    mean_squared_error(y_test_amt, lin_model_pred_test)

    print('Coefficients:', linreg_model.coef_)
    print('Intercept:', linreg_model.intercept_)
    print('Mean squared error (MSE): %.2f'
          % mean_squared_error(y_test_amt, lin_model_pred_test))



    #%% RidgeCV
```

```
from sklearn.model_selection import cross_val_score, RepeatedKFold
from sklearn.linear_model import RidgeCV

ridge_model = RidgeCV().fit(X_train_amt, np.array(y_train_amt['NEXTDAYPRECIPAMT']))

# Testing score
ridge_model_pred_test = ridge_model.predict(X_test_amt)
mean_squared_error(y_test_amt, ridge_model_pred_test)

print('Coefficients:', ridge_model.coef_)
print('Intercept:', ridge_model.intercept_)
print('Mean squared error (MSE): %.2f'
      % mean_squared_error(y_test_amt, ridge_model_pred_test))

#%% Measure Mean Square Error

print("Mean Square Error = ", get_mse(linreg_model, X_test_amt, y_test_amt))
print("Mean Square Error = ", get_mse(ridge_model, X_test_amt, y_test_amt))
```

B. utils_project1.py

```
import pandas as pd
from sklearn import tree
import pydotplus
import collections
from tqdm import tqdm
from sklearn import metrics

def create_prior_day_data_df(df, feature_names):
    for feature in feature_names:
        for idx in tqdm(df.index):
            if idx == 0:
                df.loc[idx, 'PREV_'+feature] = 0
            elif idx > 0:
                df.loc[idx, 'PREV_'+feature] = df.loc[idx-1, feature]
    return df

def get_mse(model, x, y):
    model_pred_test = model.predict(x)
    sum_ = 0
    idx_pred = 0
    for idx_y in y.index:
        a = (y.loc[idx_y, 'NEXTDAYPRECIPAMT'] - model_pred_test[idx_pred])**2
        sum_ = sum_ + a
        idx_pred = idx_pred+1
    mse = (1/len(model_pred_test)*sum_)
    return mse

def get_true_positive(decision_tree, x_test, y_test):
    """
    Returns
    -------
    true_positive_value : float64
        Results from the total correctly predicited divided by total predictions.
    matrix_df : DataFrame
        Predicted Labels are Columns and True Labels are rows.

    """
    test_pred_decision_tree = decision_tree.predict(x_test)
    confusion_matrix = metrics.confusion_matrix(y_test, test_pred_decision_tree)
    #turn this into a dataframe
    matrix_df = pd.DataFrame(confusion_matrix)
    test_set_correctly_pred = matrix_df[0][0] + matrix_df[1][1]
    true_positive = test_set_correctly_pred / len(x_test)
    return true_positive, matrix_df

def replace_missing_values_avg(df, column_name, avg_value):
    """
    This function will take in a data frame and replace a missing value with
    the average.
    """
    missing_values_bool = df[column_name].isna()
    for idx in range(len(missing_values_bool)):
        if missing_values_bool[idx] == True:
            df.loc[idx, column_name] = avg_value
            print(f"Value Replaced for {column_name} at {idx}")
        elif missing_values_bool[idx] == False:
            pass

# for a two-class tree, call this function like this:
# writegraphtofile(clf, ('F', 'T'), dirname+graphfilename)
```

```
def writegraphtofile(clf, feature_labels, classnames, pathname):
    dot_data = tree.export_graphviz(clf, out_file=None,
        feature_names=feature_labels,
        class_names=classnames,
        filled=True, rounded=True,
        special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data)
    colors = ('lightblue', 'green')
    edges = collections.defaultdict(list)
    for edge in graph.get_edge_list():
        edges[edge.get_source()].append(int(edge.get_destination()))
    for edge in edges:
        edges[edge].sort()
        for i in range(2):
            dest = graph.get_node(str(edges[edge][i]))[0]
            dest.set_fillcolor(colors[i])
    graph.write_png(pathname)

class Weather_Data_CSV:
    def __init__(self, csv_path):
        self.data_raw = pd.read_csv(csv_path)
        self.df_prep

    def get_unique_column_values(df):
        """
        Identifying Unique Values of each Column in DF
        Output is a Dictionary of each Column
        """
        headers_unique = {}
        for label in tqdm(df.columns):
            headers_unique[label] = df[label].unique()
        #pbar.close()
        return headers_unique

    def prep_data(df, df_out, headers_unique):
        """
        Extract Values for Elements and insert into df_prep
        """
        index_ = 0
        for date in tqdm(headers_unique['DATE']):
            date_idx = df['DATE'] == date
            df_by_date = df[date_idx]
            df_out.loc[index_, 'DATE'] = date
            for idx in df_by_date['ELEMENT'].index:
                df_out.loc[index_, df_by_date['ELEMENT'][idx]] = df_by_date['VALUE1'][idx]
            index_ = index_+1

    def addCol(self, label):
        pass


class StatsReport:
    def __init__(self):
        self.statsdf = pd.DataFrame()
        self.statsdf['stat'] = ['cardinality', 'mean', 'median', 'n_at_median', 'mode', 'n_at_mode', 'stddev', 'min', 'max', 'nzero', 'nmissing']
        pass

    def addCol(self, label, data):
        self.statsdf[label] = [self.cardinality_(data), self.mean_(data),
            self.median_(data), self.n_at_median(data),
            self.mode_(data), self.n_at_mode(data),
            self.std_(data), self.min_(data),
            self.max_(data), self.nzero_(data),
            self.nmissing_(data)]

    def to_string(self):
        return self.statsdf.to_string()

    def cardinality_(self, d):
        try:
            return d.nunique()
        except:
            return "N/A"

    def mean_(self, d):
        try:
            return d.mean()
        except:
            return "N/A"

    def median_(self, d):
        try:
            return d.median()
```

```
        except:
            return "N/A"

    def n_at_median(self, d):
        try:
            n = d == d.median()
            return n.sum()
        except:
            return "N/A"

    def mode_(self, d):
        try:
            return int(d.mode())
        except:
            return "N/A"

    def n_at_mode(self, d):
        try:
            n = d == int(d.mode())
            return n.sum()
        except:
            return "N/A"

    def std_(self, d):
        try:
            return d.std()
        except:
            return "N/A"

    def min_(self, d):
        try:
            return d.min()
        except:
            return "N/A"

    def max_(self, d):
        try:
            return d.max()
        except:
            return "N/A"

    def nzero_(self, d):
        try:
            n = d == 0
            return n.sum()
        except:
            return "N/A"

    def nmissing_(self, d):
        try:
            n = d.isna()
            return n.sum()
        except:
            return "N/A"
```