Precipitation Classification and Amount Prediction using Machine Learning

Bhuvaneswari Gnanasekar, Prateek Kammili, Sasweth Rajanarayanan, Vinay Shanmukh Aradhya
Group - H

***Abstract*—Weather prediction and classification is often regarded as a difficult learning problem in the machine learning space, owing to the humongous amount of factors involved that affect it. The predictors invloved also do not always have a linear relationship to the next day's weather and could sometimes be uncorrelated too. Modelling a classifier or a regression model for this task is thus challenging. Models that are trained to classify or model upcoming weather, like other learning models, should demonstrate good generalization towards unseen data. It is only after this, that a model can be deemed reliable. In this work, we attempt weather classification and prediction by means of a decision tree and a linear regression model respectively. The latter is also compared to a RidgeCV regression model on the prediction performance on the test set. Further, the best performing decision tree and linear regression models are also trained on an augmented dataset with next day parameter values also as predictors. Different hypothesis pertaining to the quality and proficiency of the dataset are also formulated and validated.**

I. Introduction

This work aims at predicting the existence of precipitation and its amount by modelling classification and regression models, respectively. The models are benchmarked on the test accuracy, in the case of the classification problem and the mean squared error of the predictions in the case of regression. This report entails details on dataset pre-processing and its exploratory analysis prior to discussions on a decision tree's performance on the dataset as a measure of the classification performance and a Linear Regression model's performance, as a measure of the regression performance. The dataset allotted to this team (team-H) is USW00094014.csv, which belongs to Williston Sloulin Intl Ap, North Dakota [1], hosting weather data of all days from 1894-2020.

II. Data Preprocessing

The dataset when downloaded, hosted the statistical measurements shown in Fig. 1. The values -999 for is representative of "Not Applicable" and is filled under a statistic that is not applicable to non-numeric data, in Fig. 1.

Labels	Mean	Median	Mode	n_mode	n_median	cardinality	n_zero	max	min	stddev	nmissing
Date	-999	-999	-999	-999	-999	45243	0	-999	-999	-999	0
Element	-999	-999	-999	-999	-999	47	0	-999	-999	-999	0
Value	148.5825	30	0	110894	2905	2171	110894	9999	-530	407.6893	0
MFlag	-999	-999	-999	-999	-999	3	0	-999	-999	-999	344321
QFlag	-999	-999	-999	-999	-999	4	0	-999	-999	-999	412310
SFlag	-999	-999	-999	-999	-999	6	0	-999	-999	-999	0
Value2	2400	2400	2400	63264	63264	1	0	2400	2400	0	349722

Figure 1. DQR - Raw Dataset

The dataset after clean-up hosted for each day, all the parameter measurements in columns.The dataset was imputed for missing values with the mean value of the respective column, and with zeroes. Imputing zeroes gave the best performance, both in the case of decision trees and linear regression and the data quality report with zero-imputation for missing values has been shown in Fig. 2. Two additional columns 'PRECIPFLAG' and 'PRECIPAMT' have been created, that represent the presence/absence of precipitation on a given date and the amount if there exists one, respectively. Further, a pair of target columns has also been created, that mirror the next day's values for 'PRECIPFLAG' and 'PRECIPAMT' on a given day. These columns are named 'NEXTDAYPRECIPFLAG' and 'NEXTDAYPRECIPAMT' respectively and serve as target columns for the classification and regression problems respectively. A correlation matrix was plotted for the dataset and the predictors that had very less correlation to the 'NEXTDAYPRECIPFLAG' column has been removed. The final dataset hosted a total of 45243 rows and 37 columns with 0s in the 'NEXTDAYPRECIPFLAG' occuring 33731 times and 1s occuring 11512 times. The histogram for the values in 'NEXTDAYPRECIPAMT' is shown in fig. 19 (at the end of the work). For a few other columns like WT**, where ** represent values from 01-22, imputation for missing values has been done based on the values of the relevant columns in the dataset.

Labels	Mean	Median	Mode	n_median	n_mode	cardinality	n_zero	max	min	stddev	nmissing
ACMH	13.16999	0	0	35270	35270	11	35[illegible]	100	0	27.83708	0
ACSH	15.46361	0	0	34695	34695	11	34[illegible]	100	0	31.413	0
AWND	11.75097	0	0	32193	32193	130	32[illegible]	156	0	20.93669	0
FMTM	326.4621	0	0	35061	35061	1351	35061	9999	0	738.188	0
GAHT	-14.0214	0	0	41813	41813	105	41813	0	-530	51.13438	0
PGTM	379.0116	0	0	32635	32635	1401	32635	2359	0	683.8855	0
PRCP	10.12307	0	0	33731	33731	195	33731	1252	0	37.81749	0
PSUN	11.01026	0	0	37647	37647	101	37647	100	0	27.25057	0
SNOW	2.321464	0	0	41054	41054	91	41054	381	0	12.7303	0
SNWD	18.96041	0	0	38412	38412	30	38412	610	0	59.68657	0
TAVG	7.067436	0	0	40246	40246	529	40246	322	-341	47.26688	0
THIC	0.287448	0	0	45240	45240	4	45240	6604	0	38.72826	0
TMAX	116.0438	128	256	499	791	183	578	433	-339	144.2486	0
TMIN	-13.6582	0	100	848	914	192	848	256	-456	126.3719	0
TSUN	169.2637	0	0	30232	30232	1001	30232	9919	0	322.8887	0
WDF1	29.26	0	0	39309	39309	37	39309	360	0	83.82099	0
WDF2	41.38012	0	0	36656	36656	37	36656	360	0	97.06375	0
WDF5	40.49709	0	0	36682	36682	37	36682	360	0	95.87907	0
WSF2	40.68886	0	0	37513	37513	14	37513	359	0	97.84434	0
WDFM	28.71991	0	0	39818	39818	9	39818	360	0	84.79581	0
WESD	41.46398	0	0	41612	41612	181	41612	8367	0	443.4347	0
WSF1	10.7683	0	0	39268	39268	43	39268	282	0	29.38647	0
WSF2	17.32513	0	0	36655	36655	55	36655	443	0	38.45579	0
WSF5	21.70424	0	0	36681	36681	72	36681	353	0	48.36315	0
WSFG	20.18308	0	0	37459	37459	59	37459	339	0	47.33181	0
WSFM	10.69542	0	0	39818	39818	55	39818	295	0	31.2032	0
WT09	0.098093	0	0	40805	40805	2	40805	1	0	0.297443	0
WT14	0.098093	0	0	40805	40805	2	40805	1	0	0.297443	0
WT15	0.093473	0	0	41014	41014	2	41014	1	0	0.291097	0
WT16	0.315165	0	0	30984	30984	2	30984	1	0	0.464587	0
WT17	0.094003	0	0	40990	40990	2	40990	1	0	0.291837	0
WT18	0.316778	0	0	30911	30911	2	30911	1	0	0.465225	0
WV03	0.000774	0	0	45208	45208	2	45208	1	0	0.027803	0
PRECIPFLAG	0.254448	0	0	33731	33731	2	33731	1	0	0.435555	0
PRECIPAMT	0.051279	0	0	33425	33425	733	33425	4.929134	0	0.175601	0
NEXTDAYPF	0.254448	0	0	33731	33731	2	33731	1	0	0.435555	0
NEXTDAYPF	0.051279	0	0	33425	33425	733	33425	4.929134	0	0.175601	0

Figure 2. DQR for the Final Dataset

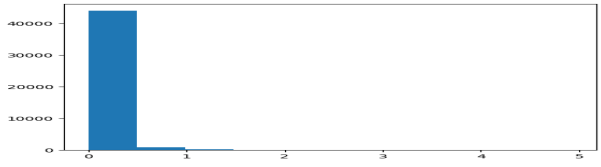

Figure 3. Histogram for the NEXTDAYPRECIPAMT target column

Prior to modelling, the dataset has also been normalized and split into training and test sets, with a 70-30 train-test split ratio.

III. Modelling and Results - Part 1

The modelling attempted in this work pertains to a classification and a regression problem from the dataset. Decision Trees with different depths have been used for the classification problem and linear regression and RidgeCV regression models have been used for the regression problem. The linear regression model has also been trained and tested for different sizes of the dataset to understand the effects of dataset size on the model's performance.

A. Classification - Decision Trees

The decision tree built in this work has been trained on the version of the dataset with a total of 37 predictors. The training regime consisted of varying the criterion between "Gini" and "Entropy" and the max_depth between 4 and 19. The model offered a good generalization to unseen examples when the tree's max_depth was set to 4, as signified by a reduced gap between the train and test accuracy. The True Positive Rate for the model has been recorded along with the tree structure. The model's performances with for the Entropy criterion for different depths of the tree has been recorded in table I, while that of the Gini Index criterion has been recorded in table II.

Max Depth	True Positive Rate
5	75.002
7	74.950
9	74.412
10	74.265
11	74.184
19	72.069

Table I
True Positive Rates for varying Tree Depths for the Entropy criterion.

Max Depth	True Positive Rate
5	74.788
7	74.906
9	74.390
10	74.228
11	73.735
19	71.075

Table II
True Positive Rates for varying Tree Depths for the Gini Index criterion.

The comparisons above show that, with increase in the depth of the tree, the true positive rate on the test set decreases. This observation is conducive to the fact that a deeper tree offers poor generalization on unseen data. The true positive rate not crossing 75 percent is also an indication to the amount of variance in the dataset. The dataset also has a lot of predictors that could be having a non-linear relationship with the target column, which could also be a reason why the decision tree struggles to achieve a good true positive rate. Between the Gini Index and Entropy criterion, the decision trees do not seem to be showing a significant difference in the True Positive Rates, with the Entropy-based trees showing a slight (insignificant) performance improvement over the Gini Index-based trees, for the same tree depths. Figures 4 - 7 show the Entropy-based decision trees, while figures 8 - 11 show the Gini Index-based trees.

B. Regression - Linear and RidgeCV Regression

The regression modelling considers the 'NEXTDAYPRECIPAMT' column in the dataset as the target column. Two regression-based models - Linear and RidgeCV has been trained on the dataset with different training regimes. Besides training on the training set split from the entire dataset, the models are also trained on different sections of the dataset, to understand the effect that the dataset has on the models' performance. Besides this exercise, the models have also been trained on augmented versions of the dataset, housing polynomial features of different order. A detailed comparison of the Mean Squared Errors of a Linear Regression model for different versions and sizes of the dataset is attempted in table III, while that for RidgeCV regression has been attempted in table IV. The training times for each of the versions of the dataset for both the linear and RidgeCV models are shown in table V, which serves as an advisory for anyone aiming to reproduce this work.

Polynomial features order	MSE	R2	Time to Train (seconds)
1	0.029	0.035	0.008
2	0.029	0.062	0.343
3	0.029	0.079	1.718
4	0.023	0.179	26.55

Table III
Mean Square Errors on the entire dataset - Linear Regression

Polynomial features order	MSE	R2	Time to Train
1	0.029	0.031	0.002
2	0.03	0.004	0.031
3	0.03	0.129	0.367

Table IV
Mean Square Errors for all days in 2000-2020 - Linear Regression

Polynomial features order	MSE	R2	Time to Train
1	0.034	0.013	0.001
2	0.036	0.048	0.023
3	0.443	0.095	0.294

Table V
Mean Square Errors for all days in 2010-2020 - Linear Regression

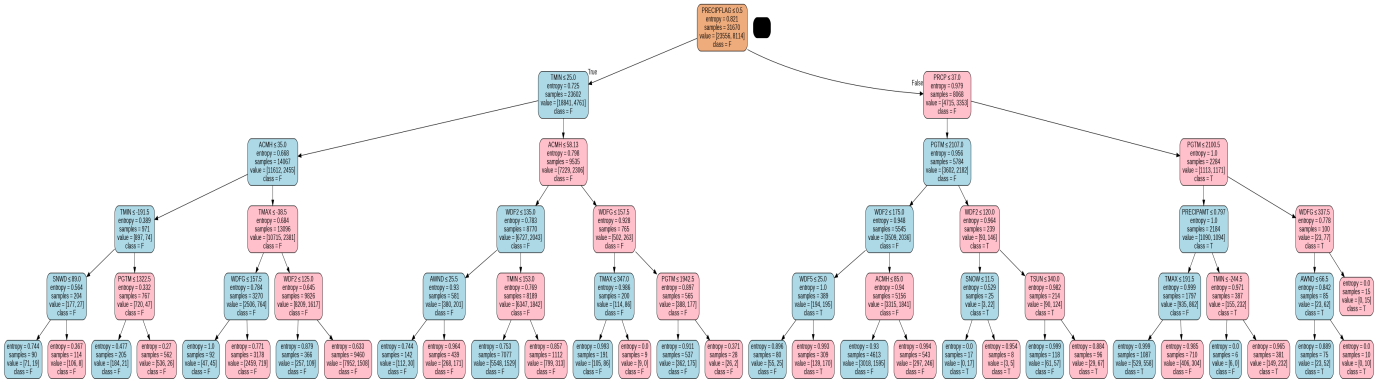

Figure 4. Entropy-based decision tree - Depth = 5

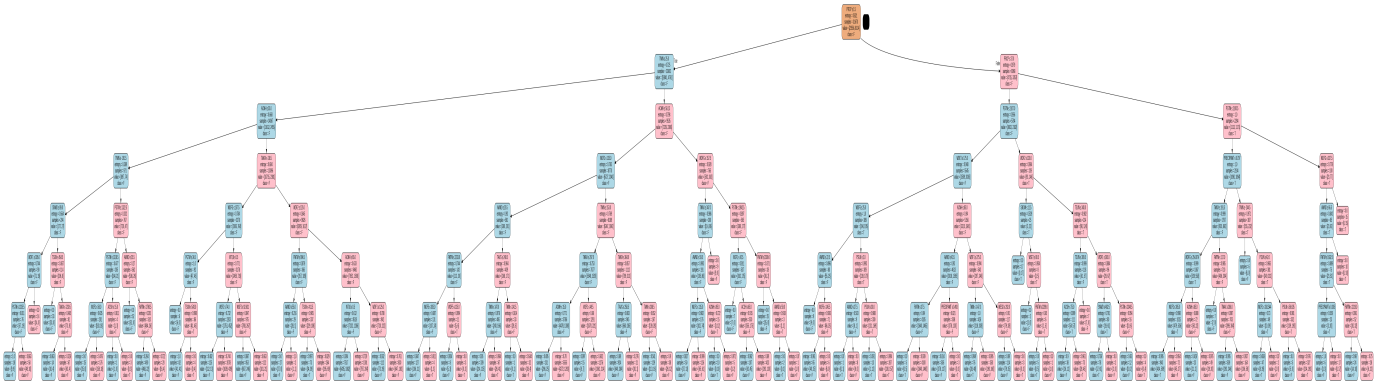

Figure 5. Entropy-based decision tree - Depth = 7

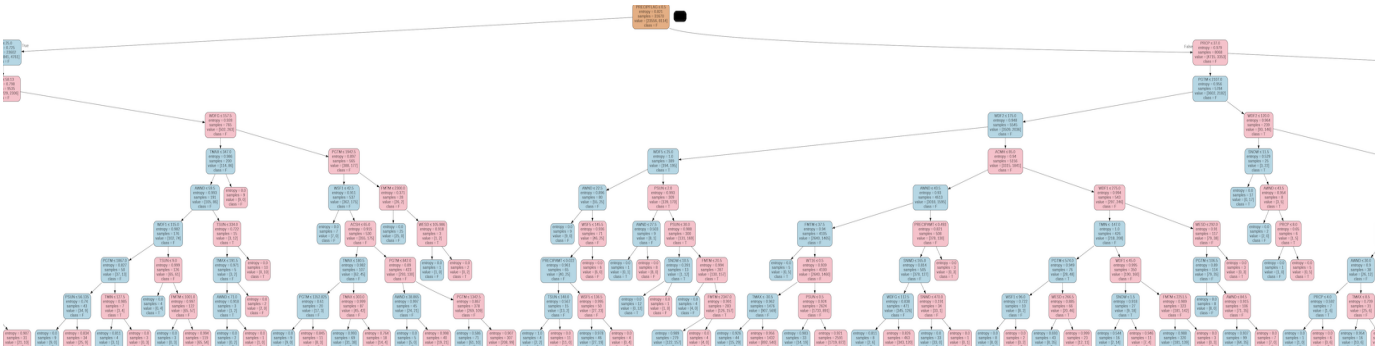

Figure 6. A part of an Entropy-based decision tree - Depth = 9

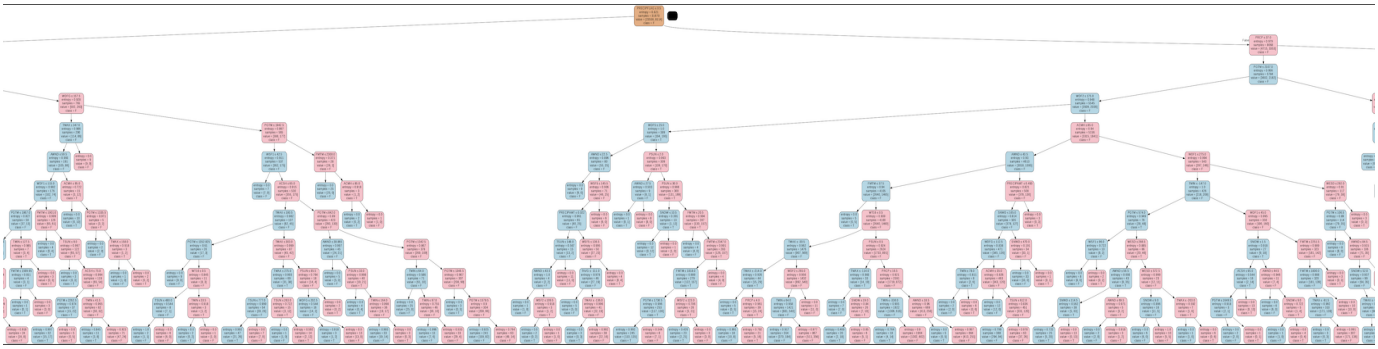

Figure 7. A part of an Entropy-based decision tree - Depth = 11

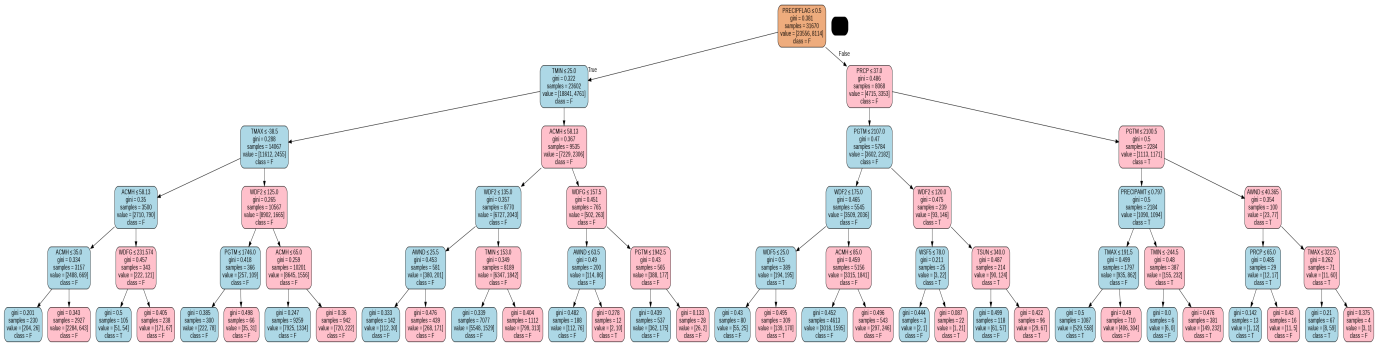

Figure 8. Gini Index-based decision tree - Depth = 5

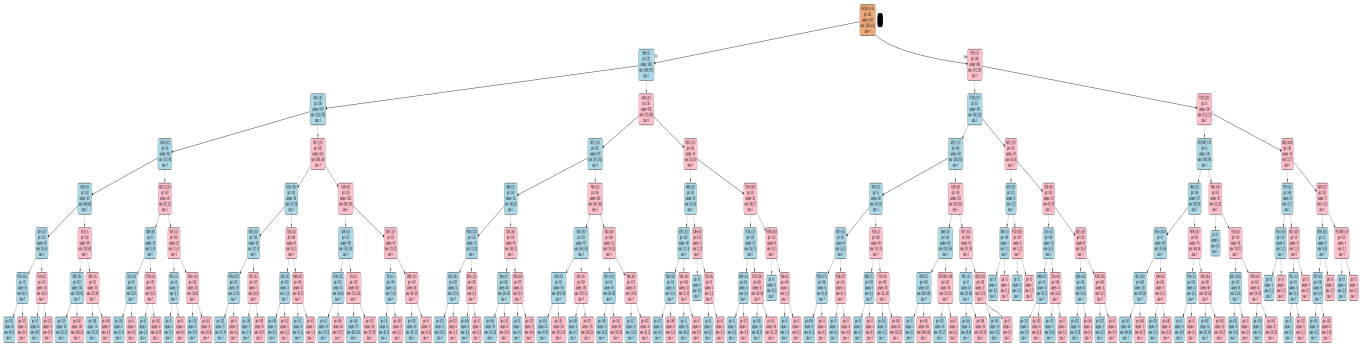

Figure 9. Gini Index-based decision tree - Depth = 7

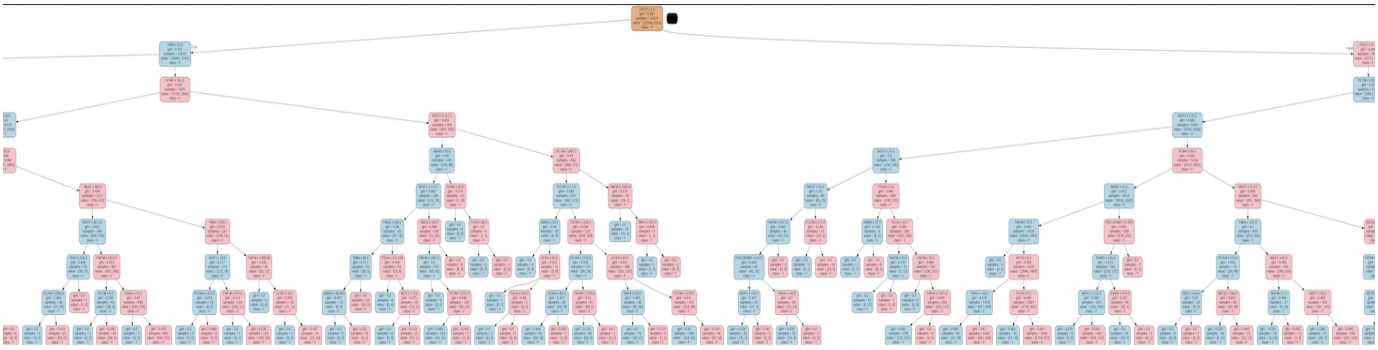

Figure 10. A part of the Gini Index-based decision tree - Depth = 9

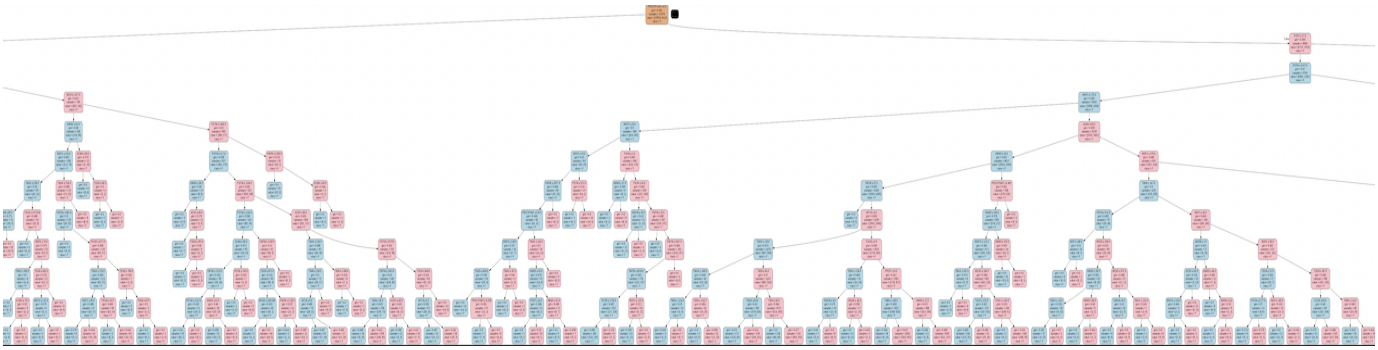

Figure 11. A part of the Gini Index-based decision tree - Depth = 11

Polynomial features order	MSE	R2	Time to Train (seconds)
1	0.031	0.037	0.014
2	0.028	0.048	0.329
3	0.027	0.056	1.787
4	0.026	0.155	17.466

Table VI
MEAN SQUARE ERRORS ON THE ENTIRE DATASET - RIDGECV REGRESSION

Polynomial features order	MSE	R2	Time to Train (seconds)
1	0.029	0.032	0.015
2	0.029	0.033	0.04
3	0.029	0.037	0.484
4	0.029	0.039	2.666

Table VII
MEAN SQUARE ERRORS FOR ALL DAYS IN 2000-2020 - RIDGECV REGRESSION

Polynomial features order	MSE	R2	Time to Train (seconds)
1	0.034	0.016	0.003
2	0.034	0.015	0.033
3	0.034	0.017	0.415
4	0.034	0.019	0.453

Table VIII
MEAN SQUARE ERRORS FOR ALL DAYS IN 2010-2020 - RIDGECV REGRESSION

The tables for both Linear Regression and RidgeCV show, besides the Mean Square Error for each polynomial order for every version of the dataset, the R-squared (R2) value on the test set. This is a measure of how well the model fits the dataset. The tables show that, while the training time increases for the models when higher order polynomial features are included in the dataset, the R2 value also increases, thereby indicating a better fit on the data. The MSE values however, do not show a consistent upward or downward trend. The reason for a better fit on the version of the dataset with higher order polynomials could be because some of the predictors have a non-linear relationship with the target variable and that while in the first order they may have a less correlation to the target variable, in the *n*th order, they could be having a high correlation to the target variable. This could have improved the model's fit on the dataset. This effort however, does not translate to decrements in MSE in a lot of cases. In RidgeCV regression, however, none of the aforementioned trends were visible.

IV. MODELLING AND RESULTS - PART 2

For the second part of this work, we attempt the classification and regression problems as described above, to an augmented dataset, which now contains, besides the values of the predictors on the given day, the values of the predictors on the following day too. The dataset therefore gives scope for the models to predict the next day's weather based on 2 previous days' data. In case of both regression and classification, the performances of other models, besides the best performing model from part 1 have also been compared, to spot inconsistencies, if any. A final comparison of the performance of the best models from part - 1 is done in table XV.

A. Classification - Decision Trees

The decision tree in this case, has been trained on the version of the dataset with a total of 76 predictors. The tables IX and X show the performance of different depths of decision trees trained on the dataset using the entropy and the Gini Index criterion respectively.

Max Depth	Test Accuracy
5	74.191
7	74.118
9	73.764
10	73.668
11	73.388
19	72.512

Table IX
TRUE POSITIVE RATES FOR VARYING TREE DEPTHS FOR THE ENTROPY CRITERION.

Max Depth	Test Accuracy
5	74.132
7	74.147
9	73.786
10	73.713
11	73.550
19	71.915

Table X
TRUE POSITIVE RATES FOR VARYING TREE DEPTHS FOR THE GINI INDEX CRITERION.

The comparisons above show that, in resonance with the previous observations, with increase in the depth of the tree, the true positive rate on the test set decreases. This observation is conducive to the fact that a deeper tree offers poor generalization on unseen data. The performance on the test set is however comparable to that from Part 1. The addition of extra predictors, to make the learning problem more robust, did not cause any impact to the performance of the model. Once again, we hypothesize here, that, the dataset also has a lot of predictors that could be having a non-linear relationship with the target column. Between the Gini Index and Entropy criterion, the decision trees do not seem to be showing a significant difference in the test accuracy, with the entropy-based trees showing a slight (insignificant) performance improvement over the Gini Index-based trees, for the same tree depths. The best performing tree from part 1 was an entropy-based tree with depth = 5, which also emerges as the best performing tree in part 2. Figures 12 - 15 show the Entropy-based decision trees, while figures 16 - 19 show the Gini Index-based trees. From the above comparisons, we find that for either of the parts, the entropy-based decision tree with depth equal to 5 is the best performing tree as far as generalization to unseen data is concerned.

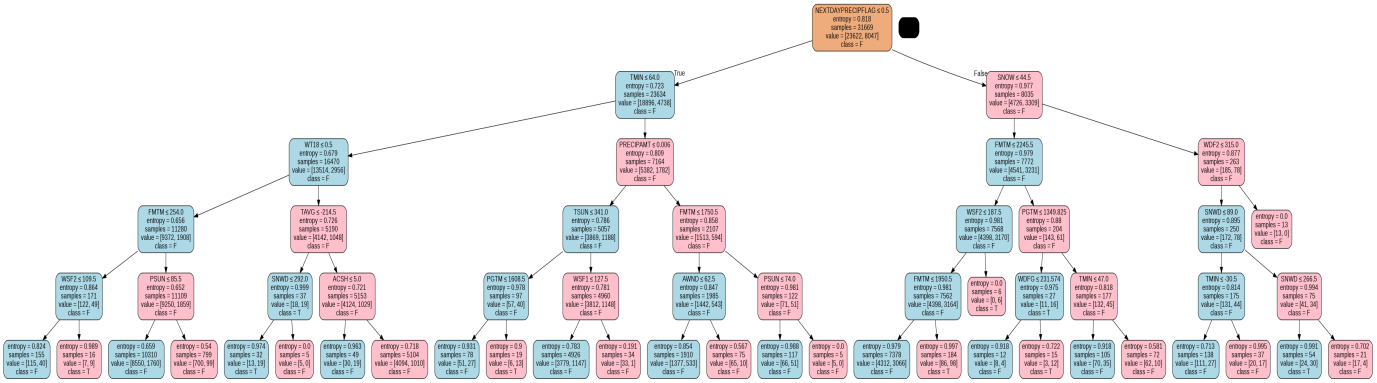

Figure 12. Entropy-based decision tree - Depth = 5 (**The best performing decision tree from Part - 1).**

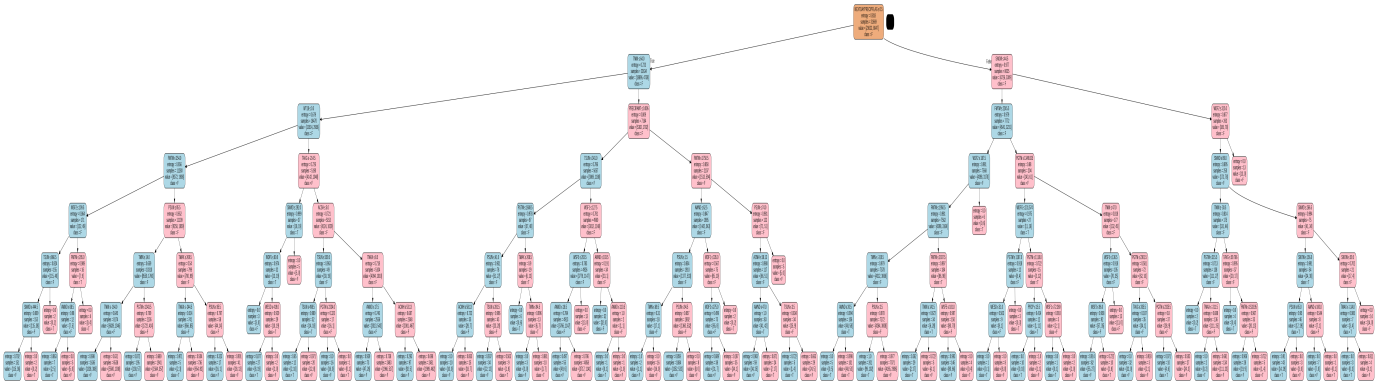

Figure 13. Entropy-based decision tree - Depth = 7

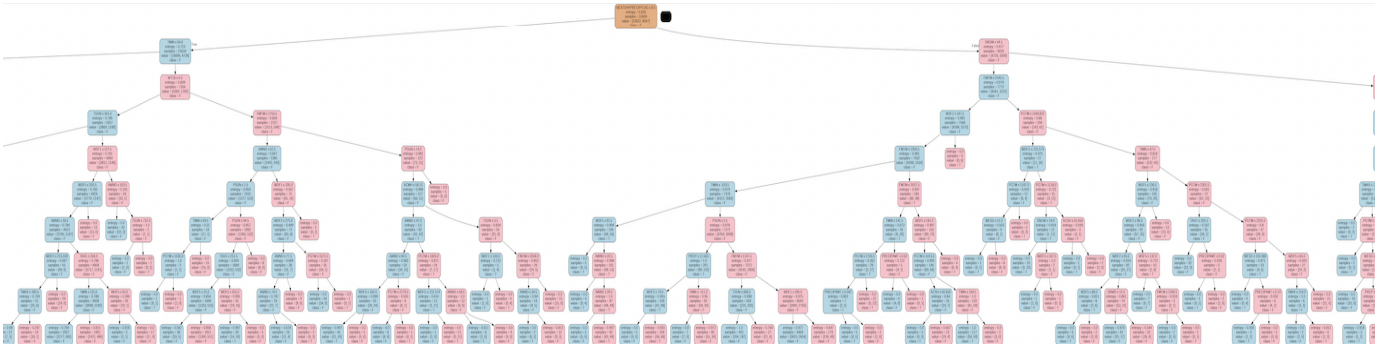

Figure 14. A part of an Entropy-based decision tree - Depth = 9

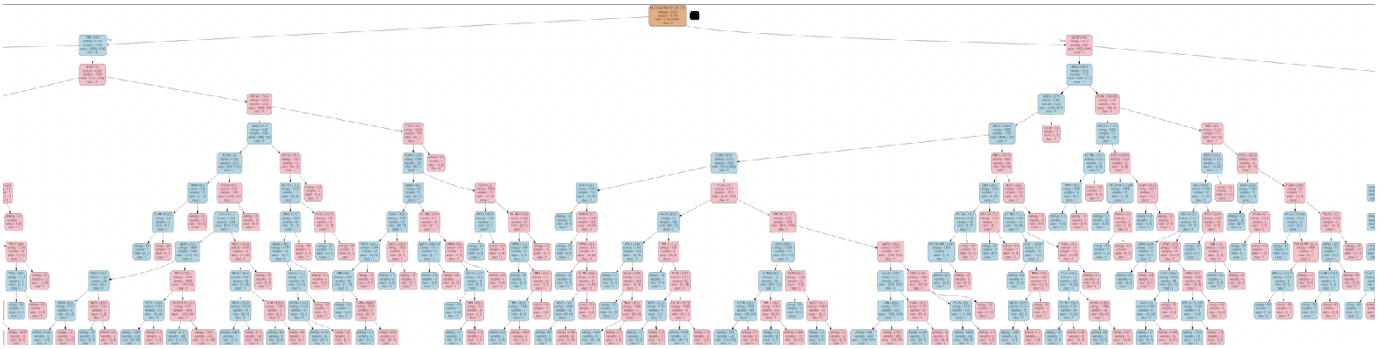

Figure 15. A part of an Entropy-based decision tree - Depth = 11

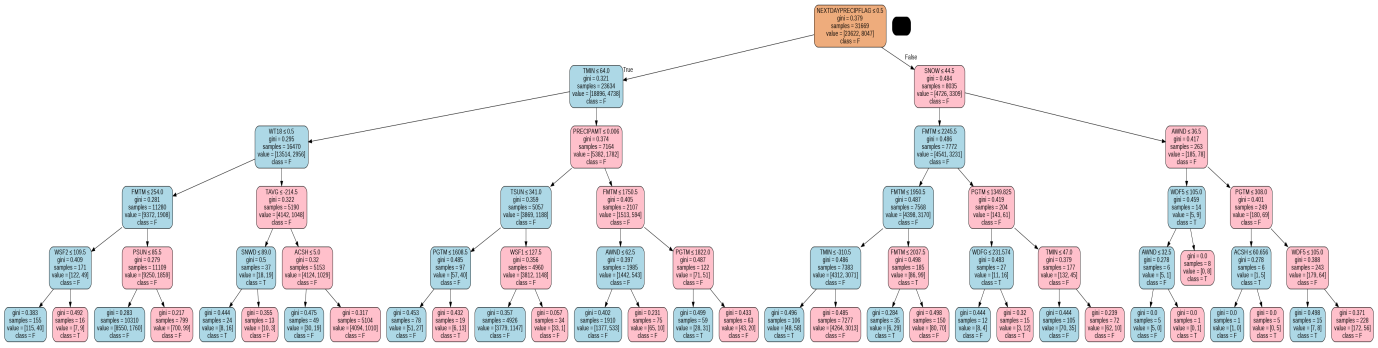

Figure 16. Gini Index-based decision tree - Depth = 5

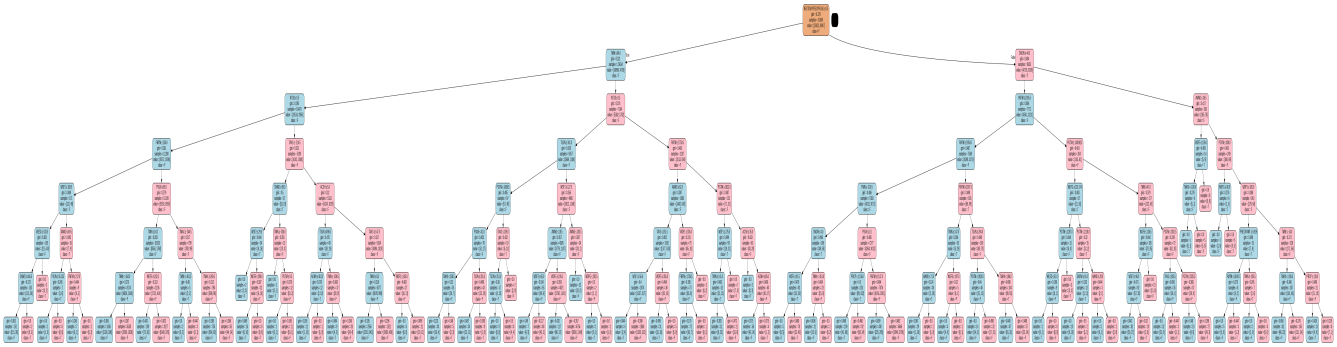

Figure 17. Gini Index-based decision tree - Depth = 7

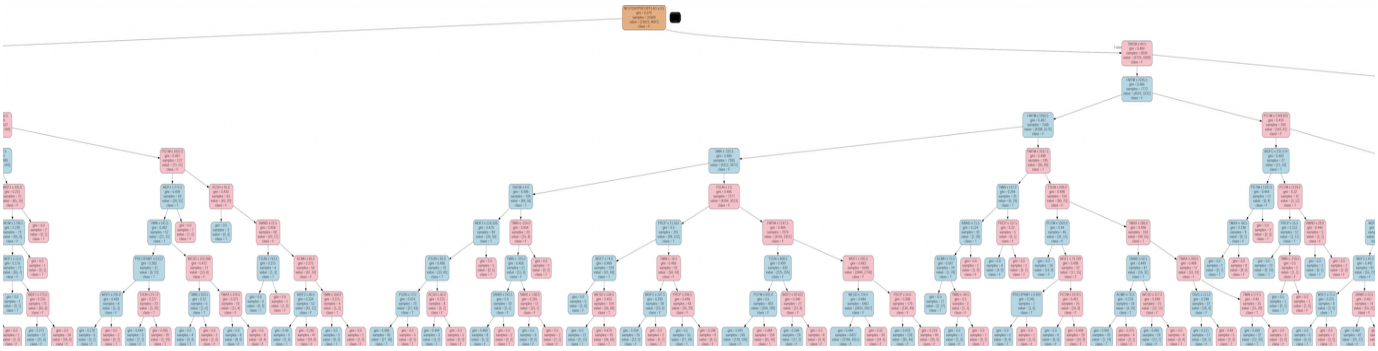

Figure 18. A part of the Gini Index-based decision tree - Depth = 9

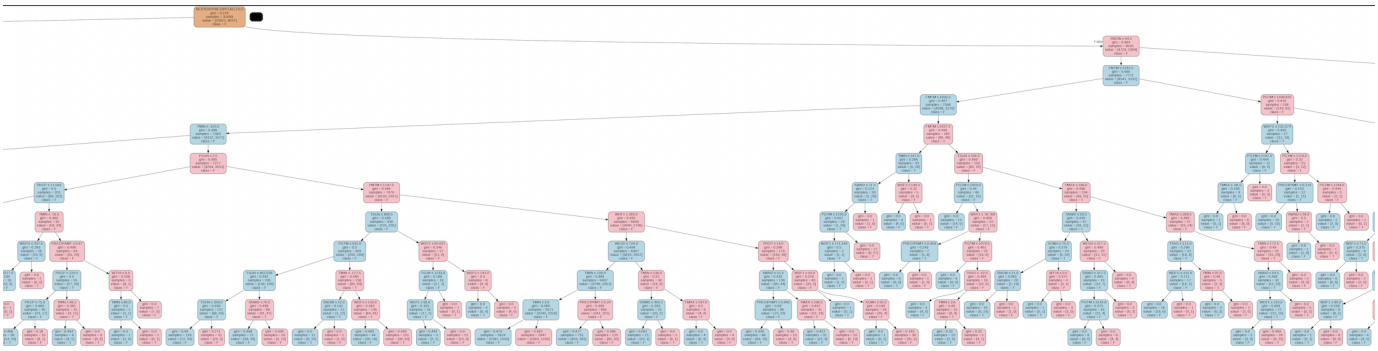

Figure 19. A part of the Gini Index-based decision tree - Depth = 11

B. Regression - Linear and RidgeCV Regression

The regression performance for Linear and RidgeCV regression models have been demonstrated in tables XI - XIV for different versions of the dataset (1894 - 2020, 2000 - 2020, 2010 - 2020). Table XI shows the performance of the linear regression model on first order polynomial features. The best performing model from part - 1 was a RidgeCV regression model trained on the 1894-2020 part of the dataset hosting polynomial features of 4th order. However, to observe inconsistencies and hidden trends, we also compare the model performances on other versions of the dataset too. Moreover, the reason why we trained the models on the 2000 - 2020 and 2010 - 2020 sections of the dataset is because, with the advent of technology, precise measurement practices could have been enforced over time. So, we hypothesized that the dataset, as time passed, would hold precise measurements of the listed parameters (predictors) and the model would be subjected to lesser outliers than before. Moreover, the data from 1894-2020 could have had skewed measurements owing to less accurate measurement practices. While there is no certainty on this, the trial helped us understand the plausibility of our hypothesis.

Date	MSE	R2	Time to Train (seconds)
1894 - 2020	0.031	0.033	0.1952
2000 - 2020	0.031	0.047	0.0257
2010 - 2020	0.029	0.045	0.0177

Table XI
MEAN SQUARE ERRORS - LINEAR REGRESSION

Polynomial Order	MSE	R2	Time to Train (seconds)
1	0.031	0.019	0.038
2	0.031	0.022	1.157
3	0.031	0.023	10.057
4	0.031	0.023	423.901

Table XII
MEAN SQUARE ERRORS FOR ALL DAYS IN 1894-2020 - RIDGECV REGRESSION

Polynomial features order	MSE	R2	Time to Train (seconds)
1	0.027	0.007	0.006
2	0.027	0.007	0.166
3	0.028	0.018	3.000
4	0.028	0.013	18.76

Table XIII
MEAN SQUARE ERRORS FOR ALL DAYS IN 2000-2020 - RIDGECV REGRESSION

From the observations above, we find that the Linear Regression model with order 1 polynomial features for the 2010-2020 version of the dataset strikes the perfect balance between the mean squared error and the R2 value on the test set. It also takes 0.0177 seconds to train, while the presence of 4th order polynomial features in the same version of the dataset makes the RidgeCV model struggle to achieve a good R2 and a lower MSE. The fit of the RidgeCV model on the test set was infact better for part 1 than part 2, when the 2010-2020 section of the dataset is considered, while the MSE showed a reverse trend. The final results of part 1 and part 2 has been shown in table XV.

Polynomial features order	MSE	R2	Time to Train (seconds)
1	0.034	0.017	0.006
2	0.034	0.021	0.121
3	0.034	0.021	3.18
4	0.034	0.003	3.588

Table XIV
MEAN SQUARE ERRORS FOR ALL DAYS IN 2010-2020 - RIDGECV REGRESSION

Dataset/Part	MSE of best Linear regression model	Test Accuracy of best Decision Tree
Part 1	0.026 (1894-2020, Order 4 Polynomial features)	75.002
Part 2	0.029 (2010-2020, Order 1 Polynomial features)	74.191

Table XV
MEAN SQUARE ERRORS AND TEST ACCURACIES OF LINEAR REGRESSION AND DECISION TREE CLASSIFIERS FOR PART 1 2

V. DISCUSSIONS AND CONCLUSIONS

This work presented a classification and regression learning problem to classify and predict the next day's weather in terms of the existence and amount of precipitation, respectively. A variety of decision trees and linear regression models were trained and their performances were compared. For part 1, where the models were trained only on the current day's weather parameters to predict/classify the next day's weather, the entropy-based decision tree with a depth of 5 presented a good generalization to unseen data. While the generalization was not very high, a test accuracy of that sorts questions the variance of the dataset. A dataset of high variance would indeed make it difficult for a model to generalise to unseen data. This can also be confirmed from the Data Quality Report of the final dataset, as shown in Fig. 2. The variance of certain features are very high and these could have caused the model to overfit. This is synchronous to the trends observed in Part 2, where the entropy-based decision tree with depth equal to 7 presented a better generalization to unseen data than decision trees deeper than itself. The performance of the best performing decision tree from part 1 was slightly reduced when trained on the augmented dataset in part 2.

The regression performances were also almost similar when part 1 and 2 are compared. Infact, the MSE of the best performing model from Part - 1 (RidgeCV Regression model with 4th order polynomial features) did not perform as good in Part 2. The model that performed the best in Part 2 was a linear regression model, when it was trained on the augmented dataset from 2010-2020, as it striked the perfect balance between the MSE and R2 values. As the results show, for Part

2, the model demonstrated a better performance on the 2010-2020 part of the dataset, which partly proves our hypothesis that data measurement techniques could have become more robust over time allowing for more accurate measurements of parameters. However, there are cases, like in part 1, where this hypothesis also falls apart.

REFERENCES

[1] M. J. Menne, I. Durre, R. S. Vose, B. E. Gleason, and T. G. Houston, "An overview of the global historical climatology network-daily database," *Journal of Atmospheric and Oceanic Technology*, vol. 29, no. 7, pp. 897 – 910, 2012. [Online]. Available: https://journals.ametsoc.org/view/journals/atot/29/7/jtech-d-11-00103_1.xml

Appendix

Data Preprocessing

```
import pandas as pd
import numpy as np
import xlsxwriter
import matplotlib.pyplot as plt

df = pd.read_csv("USW00094014.csv")
df = pd.DataFrame(df)
df.columns = ['ID', 'Date', 'Element', 'Value',
              'MFlag', 'QFlag', 'SFlag',
              'Value2']
df1 = pd.DataFrame({'ID': ["USW00094014"],
                    'Date': ["18940101"],
                    'Element': ["TMAX"],
                    'Value': [0],
                    'MFlag': [np.NaN],
                    'QFlag': [np.NaN],
                    'SFlag': ["X"],
                    'Value2': [np.NaN]})
df = pd.concat([df1, df], ignore_index=True)
del df['ID']

means = []
median = []
mode = []
cardinality = []
n_median = []
n_mode = []
n_zero = []
max = []
min = []
stddev = []
n_missing = []
print(df)
for label in df.columns:
    card = 0
    sum = (df[label].isnull()).sum()
    if (sum != 0):
        card = 1
    count = 0
    if label not in ['Value', 'Value2']:
        # Write -999 if stat not
        # applicable
        means.append(-999)
        median.append(-999)
        # medVal = df[label].median()
        mode.append(-999)
        modeVal = df[label].mode()
        cardinality.append(
            str(len(df[label].unique())
                - card))
        n_median.append(-999)
        n_mode.append(-999)
        n_zero.append(
            str((df[label] == 0).sum()))
        max.append(-999)
        min.append(-999)
        stddev.append(-999)
        n_missing.append(
            str((df[label].isnull()).sum()
                +(df[label] == -9999).sum()))
        continue

    means.append(
        df[label].mean(skipna=True))
    median.append(
        df[label].median(skipna=True))
    medVal = df[label].median(skipna=True)
    mode.append(
        df[label].mode(dropna=True))
    modeVal = df[label].mode(dropna=True)
    cardinality.append(
        len(df[label].unique()) - card)
    n_median.append(
        (df[label] == medVal).sum())
    n_mode.append(
        (df[label] == modeVal[0]).sum())
    n_zero.append(
        (df[label] == 0).sum())
    max.append(
        df[label].max())
    min.append(df[label].min())
    stddev.append(
        df[label].std())
    n_missing.append(
        (df[label].isnull()).sum()
        + (df[label] == -9999).sum())

modes = []
for i in mode:
    modes.append(i)

df9 = pd.DataFrame({'Labels':
                        df.columns,
                    'Mean': means,
                    'Median':
                        median,
                    'Mode' :
                        modes,
                    'n_mode' :
                        n_mode,
                    'n_median':
                        n_median,
                    'cardinality':
                        cardinality,
                    'n_zero':
                        n_zero,
                    'max': max,
                    'min':
                        min,
                    'stddev':
                        stddev,
                    'nmissing':
                        n_missing})
```

```
# DQR of raw dataset stored as Excel file
df9.to_excel("DQR_RAW_DATASET.xlsx")
df2 = df.pivot(index="Date",
               columns="Element",
               values="Value")

eliminators = df2.columns[0:26]

for label in eliminators:
    mean = df2[label].mean()
    df2[label] = df2[label].fillna(
        value=0)
df2['WV03'] = df2['WV03'].fillna(
    value=0)
#print(df2)
wts = df2.columns[26:46]

# Set the NaN values in all the WT** columns
# to a random value other than 1 and 0.
# Then, replace this value with a 0 or 1
# based on the PRCP and SNOW columns.
# If PRCP column has a value that is not 0,
# then a flag like WT18 which indicates
# "Unknown Source of Precipitation" will
# be 1, as there was precipitation. Further
# , WT16 which indicates Rain will be set
# to 1 if SNOW value in the row is not 0.
# This is because we consider that 7 inches
# of snow is 1 inch of rain. So any non 0
# value of SNOW must be equivalent to some
# amount of rain, even if it is negligible.
for label in wts:
    df2[label] = df2[label].fillna(value
                                   =450)

colsToRetain = []
for i, row in df2.iterrows():
    if row['WT14'] == 450:
        if row['SNOW'] == 0.0:
            row['WT14'] = 0
        else:
            row['WT14'] = 1
        colsToRetain.append('WT14')
    if row['WT15'] == 450:
        if row['SNOW'] == 0.0:
            row['WT15'] = 0
        else:
            row['WT15'] = 1
        colsToRetain.append('WT15')
    if row['WT16'] == 450:
        colsToRetain.append('WT16')
        if row['SNOW'] != 0 \
                or row['PRCP'] != 0:
            row['WT16'] = 1
        else:
            row['WT16'] = 0
    if row['WT17'] == 450:
        if row['SNOW'] == 0.0:
            row['WT17'] = 0
        else:
            row['WT17'] = 1
        colsToRetain.append('WT17')
    if row['WT18'] == 450:
        if row['SNOW'] != 0 \
                or row['PRCP'] != 0:
            row['WT18'] = 1
        else:
            row['WT18'] = 0
        colsToRetain.append('WT18')
    if row['WT09'] == 450:
        if row['SNOW'] == 0:
            row['WT09'] = 0
        else:
            row['WT09'] = 1
        colsToRetain.append('WT09')


for label in wts:
    if label not in colsToRetain:
        del df2[label]

precipflag = []
precipamt = []

for i, row in df2.iterrows():
    if row['PRCP'] > 0:
        precipflag.append(1)
    else:
        precipflag.append(0)
    prcpamt = (row['PRCP']
               / (10 * 25.4)) + \
              (row['SNOW']
               / (25.4 * 8))
    precipamt.append(prcpamt)

df2['PRECIPFLAG'] = precipflag
df2['PRECIPAMT'] = precipamt

nextdayflag = []
nextdayamt = []
flag = 0
amt = 0
count = 0
for i, row in df2.iterrows():
    if count == 0:
        flag = row['PRECIPFLAG']
        amt = row['PRECIPAMT']
        count += 1
        continue
    nextdayflag.append(row['PRECIPFLAG'])
    nextdayamt.append(row['PRECIPAMT'])

nextdayflag.append(flag)
nextdayamt.append(amt)
df2['NEXTDAYPRECIPFLAG'] = nextdayflag
```

```
df2['NEXTDAYPRECIPAMT'] = nextdayamt

means = []
median = []
mode = []
cardinality = []
n_median = []
n_mode = []
n_zero = []
max = []
min = []
stddev = []
n_missing = []

for label in df2.columns:
    card = 0
    sum = (
        df2[label].isnull()).sum()
    if(sum != 0):
        card = 1
    count = 0
    means.append(
        df2[label].mean())
    median.append(
        df2[label].median())
    medVal = df2[label].median()
    mode.append(
        df2[label].mode())
    modeVal = df2[label].mode()
    cardinality.append(
        len(df2[label].unique()) - card)
    n_median.append(
        (df2[label] == medVal).sum())
    n_mode.append(
        (df2[label] == modeVal[0]).sum())
    n_zero.append(
        (df2[label] == 0).sum())
    max.append(
        df2[label].max())
    min.append(
        df2[label].min())
    stddev.append(
        df2[label].std())
    n_missing.append(
        (df2[label].isnull()).sum()
        +(df2[label] == -9999).sum())

modes = []
for i in mode:
    modes.append(i[0])

df4 = pd.DataFrame({'Labels':
                        df2.columns,
                    'Mean':
                        means,
                    'Median':
                        median,
                    'Mode':
                        modes,
                    'n_median':
                        n_median,
                    'n_mode':
                        n_mode,
                    'cardinality':
                        cardinality,
                    'n_zero':
                        n_zero,
                    'max': max,
                    'min': min,
                    'stddev':
                        stddev,
                    'nmissing':
                        n_missing})
# Final Dataset's DQR saved as Excel file.
df4.to_excel("DQR_CLEANED_DATASET.xlsx")

corr = df2.corr()
corr.to_excel("correlations.xlsx")

plt.hist(df2['NEXTDAYPRECIPAMT'])
# plt.show() # -> uncomment to show the
# histogram

freq = df2['NEXTDAYPRECIPFLAG'].value_counts()
print("NEXTDAYPRECIPFLAG Frequency counts",
      freq)

# Save the dataset by Pickling it

df2.to_hdf("TheDataset.h5", key='ML')

df2.to_pickle('Cleaned_Dataset.pkl')

df5 = pd.read_hdf('TheDataset.h5', key='ML')

df6 = pd.read_pickle('Cleaned_Dataset.pkl')
print(df5)
```

Decision Tree - Part 1 and 2

```
# -*- coding: utf-8 -*-


import pandas as pd
import sklearn
from sklearn import tree
from sklearn.model_selection import \
  train_test_split
import pydotplus
import collections
import xlsxwriter
from sklearn.preprocessing import MinMaxScaler

#read dataset-define model-predict on
# test data-graph

class decision_tree():

  def __init__(self):
      self.scaler = MinMaxScaler()

  def dataset_next_day(self,path):
    # data processing for part 1
    df = pd.read_csv(path)
    y = df['NEXTDAYPRECIPFLAG']
    x = df[df.columns[1: -2]]
    X_train, X_test, y_train, y_test = \
      train_test_split(
      x, y, test_size=0.3, random_state=42)
    self.scaler.fit(X_train)
    x_train = self.scaler.transform(X_train)
    x_test = self.scaler.transform(X_test)
    return X_train, X_test, y_train, y_test

  def dataset_day_after_tomorrow(self,path):
    # data processing for part 2
    df = pd.read_csv(path)
    df.drop(df.tail(1).index,inplace=True)
    y = df['day_after_tomorrow']
    x = df[df.columns[1: -1]]
    x.drop(['NEXTDAYPRECIPAMT'],
           axis = 1, inplace = True)
    X_train, X_test, y_train, y_test = \
      train_test_split(
      x, y, test_size=0.3, random_state=42)
    self.scaler.fit(X_train)
    x_train = self.scaler.transform(X_train)
    x_test = self.scaler.transform(X_test)
    return X_train, X_test, y_train, y_test

  def model_tree(self,depth,crit,
                 X_train, y_train):
    #define decision tree model
    clf = tree.DecisionTreeClassifier(
      criterion=crit,max_depth=depth)
    clf = clf.fit(X_train, y_train)
    return clf

  def pred(self,model,X_test):
    #make prediction on test dataset
    prediction=model.predict(X_test)
    return prediction

  def t_p_r(self,y_test,prediction):
    # calcualte true positive rate
    y_t=list(y_test)
    type(y_t)
    count_tp=0
    p=prediction
    for i in range (len(prediction)):
        if y_t[i]==p[i]:
            count_tp+=1
    trp=count_tp/len(p)
    return trp*100

  def writegraphtofile(self,classifier,
                       features, classnames,
                       pathname):
      # write tree to image
      dot_data = tree.export_graphviz(
        classifier, out_file=None,
        class_names=classnames,
        feature_names=features,
        filled=True, rounded=True,
        special_characters=True)
      graph = pydotplus.graph_from_dot_data(
        dot_data)
      colors = ('lightblue', 'pink')
      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)

# predict precipitation for part 1 and 2

class prediction():
  #prediction class
  def __init__(self) :
      self.t=decision_tree()

  def pred_next_day(self,path,
                    depth,crit,
                    png_path):
    # predicting next day preciptation
    X_train, X_test, y_train, y_test=\
      self.t.dataset_next_day(path)
```

```
    model= self.t.model_tree(
      depth,crit,X_train, y_train)
    prediction=self.t.pred(model,X_test)
    tpr=self.t.t_p_r(y_test,prediction)
    features=X_train.columns
    self.t.writegraphtofile(model,
                            features,
                            ('F','T'),
                            png_path)
    return tpr

  def pred_day_after_tomorrow(self,path,
                              depth,crit,
                              png_path):
    # predicting for part 2
    X_train, X_test, \
    y_train, y_test=\
      self.dataset_day_after_tomorrow(path)
    model= self.t.model_tree(
      depth,crit, X_train, y_train)
    prediction=self.t.pred(model,X_test)
    tpr=self.t.t_p_r(y_test,prediction)
    features=X_train.columns
    self.t.writegraphtofile(model,
                            features ,
                            ('F','T'),
                            png_path)
    return tpr

#fetch and run

class run():
  #last class -> fetch file, display
  # and save results
  def __init__(self):
    self.t=prediction()
  def next_day(self):
    # for next day prediction with
    # criterion entropy
    path='/content/file.csv'
    p="/content/entropy/depth"
    crit='entropy'
    print('*'*10,'entropy','*'*10)
    for depth in range(4,20):
      png_path=p+str(depth)+'.png'
      tpr=self.t.pred_next_day(
        path,depth,crit,png_path)
      print('depth of the model=',depth,
            "and true positive rate=",
            tpr)

    # for next day prediction
    # with criterion gini
    p="/content/gini/depth"
    crit='gini'
    print('*'*10,'gini','*'*10)
    for depth in range(4,20):
      png_path=p+str(depth)+'.png'
      tpr=self.t.pred_next_day(path,depth,
                               crit,
                               png_path)
      print('depth of the model=',depth,
            "and true positive rate=",tpr)

  def day_after_tomorrow(self):
    # for day after tomorrow prediction
    # with criterion entropy
    path='/content/file.csv'
    p="/content/entropy/depth"
    crit='entropy'
    print('*'*10,'entropy','*'*10)
    for depth in range(4,20):
      png_path=p+str(depth)+'.png'
      tpr=self.t.pred_day_after_tomorrow(path,
                                         depth,
                                         crit,
                                         png_path)
      print('depth of the model=',depth,
            "and true positive rate=",tpr)

    # for day after tomorrow prediction
    # with criterion gini
    p="/content/gini/depth"
    crit='gini'
    print('*'*10,'gini','*'*10)
    for depth in range(4,20):
      png_path=p+str(depth)+'.png'
      tpr=self.t.pred_day_after_tomorrow(path,
                                         depth,crit
                                         png_path)
      print('depth of the model=',depth,
            "and true positive rate=",tpr)

df = pd.read_pickle('Cleaned_Dataset.pkl')
df.to_csv('DatasetCleaned.csv')
path='DataCleaned.csv'
t=run()
t.next_day()
t.day_after_tomorrow()
```

Data Preprocessing for Part 2

```
import pandas as pd

df5 = pd.read_hdf('TheDataset.h5', key='ML')

for label in df5.columns:
    name = label
    count = 0
    nextCol = []
    for i, row in df5.iterrows():
        if count == 0:
            count += 1
            continue
        nextCol.append(row[label])
    nextCol.append(0)
    name = name + "2"
    df5[name] = nextCol

count1 = 0
flag = []
amt = []
for i, row in df5.iterrows():
    if count1 == 0:
        count1 += 1
        continue
    flag.append(row['NEXTDAYPRECIPFLAG2'])
    amt.append(row['NEXTDAYPRECIPAMT2'])

flag.append(0)
amt.append(0)

df5['NEXTDAYPRECIPFLAG3'] = flag
df5['NEXTDAYPRECIPAMT3'] = amt

df5.to_hdf("DatasetPart2.h5", key='ML')

df = pd.read_hdf("DatasetPart2.h5", key='ML')

print(df)
```

Linear Regression - Part 1

```
import time

from sklearn import preprocessing as \
    preproc
from sklearn import linear_model as \
    linmod
import pandas as pd
from sklearn.linear_model import \
    LinearRegression
from sklearn import model_selection
from sklearn import metrics
import numpy as np
import pickle

def showStats(W, X, Yact, Ypred):
    print("R2 = %f, MSE = %f" % (
        mlr.score(X, Yact),
        metrics.mean_squared_error(
            Yact, Ypred)))


# Fitting on the entire dataset
df = pd.read_pickle('Cleaned_Dataset.pkl')

x = df[['PRCP', 'WT18',
        'PRECIPFLAG', 'PRECIPAMT',
        'SNOW', 'WT16', 'ACMH', 'ACSH',
        'TMIN', 'PSUN', 'WDF5', 'WDF1', 'WDF2',
        'TSUN']]
y = df['NEXTDAYPRECIPAMT']

scaler = preproc.MinMaxScaler()
(x_train, x_test, y_train, y_test) = \
    model_selection.train_test_split(x, y,
                                     test_size=1/3,
                                     random_state=0)

scaler.fit(x_train)
scaler.fit(x_test)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)

print()
t1 = time.time()
reg = LinearRegression().fit(x_train, y_train)
print(time.time() - t1)

y_pred = reg.predict(x_test)

print("MSE Without Polynomial Test: ",
      metrics.mean_squared_error(y_test, y_pred))
print("R2 Without Polynomial Test Score: ",
      reg.score(x_test, y_test))

print("R2 Without Polynomial Training Score: ",
      reg.score(x_train, y_train))

poly = preproc.PolynomialFeatures(1)
bigTrainX = poly.fit_transform(x_train)
bigTestX = poly.fit_transform(x_test)
mlr = linmod.LinearRegression()

print()
t = time.time()
mlr.fit(bigTestX, y_test)
print("Test Fit Time", time.time() - t)

print()
t = time.time()
mlr.fit(bigTrainX, y_train)
print("Training Fit Time", time.time() - t)
print()


showStats(np.append(np.array(mlr.intercept_),
                    mlr.coef_), bigTestX, y_test,
          mlr.predict(bigTestX))
print()
showStats(np.append(np.array(mlr.intercept_), mlr.
          bigTrainX, y_train, mlr.predict(
        bigTrainX))
```

RidgeCV Regression - Part 1

```
import time

from sklearn import preprocessing as preproc
from sklearn import linear_model as linmod
import pandas as pd
from sklearn.linear_model import RidgeCV
from sklearn import model_selection
from sklearn import metrics
import numpy as np

def showStats(W, X, Yact, Ypred):
    print("R2 = %f, MSE = %f" % (
        mlr.score(X, Yact),
        metrics.mean_squared_error(Yact, Ypred)))


df = pd.read_pickle('Cleaned_Dataset.pkl')


x = df[['PRCP', 'WT18', 'PRECIPFLAG', 'PRECIPAMT',
        'SNOW', 'WT16', 'ACMH', 'ACSH', 'TMIN',
        'PSUN', 'WDF5', 'WDF1', 'WDF2', 'TSUN']]
y = df['NEXTDAYPRECIPAMT']

scaler = preproc.MinMaxScaler()
(x_train, x_test, y_train, y_test) = \
    model_selection.train_test_split(x, y,
                                     test_size=1/3,
                                     random_state=0)

scaler.fit(x_train)
scaler.fit(x_test)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)

print()
t1 = time.time()
reg = RidgeCV().fit(x_train, y_train)
print(time.time() - t1)

y_pred = reg.predict(x_test)

print("MSE Without Polynomial Test: ",
      metrics.mean_squared_error(y_test, y_pred))
print("R2 Without Polynomial Test Score: ",
      reg.score(x_test, y_test))

print("R2 Without Polynomial Training Score: ",
      reg.score(x_train, y_train))

poly = preproc.PolynomialFeatures(1)
bigTrainX = poly.fit_transform(x_train)
bigTestX = poly.fit_transform(x_test)
mlr = linmod.RidgeCV()

print()
t = time.time()
mlr.fit(bigTestX, y_test)
print("Test Fit Time", time.time() - t)

print()
t = time.time()
mlr.fit(bigTrainX, y_train)
print("Training Fit Time", time.time() - t)
print()

showStats(np.append(
    np.array(mlr.intercept_), mlr.coef_),
    bigTestX, y_test, mlr.predict(
        bigTestX))
print()
showStats(np.append(
    np.array(mlr.intercept_), mlr.coef_),
    bigTrainX, y_train, mlr.predict(
        bigTrainX))
```

Linear Regression - Part 2

```
import time
from sklearn import preprocessing \
    as preproc
from sklearn import linear_model \
    as linmod
import pandas as pd
from sklearn.linear_model \
    import LinearRegression
from sklearn import \
    model_selection
from sklearn import metrics
import numpy as np

def showStats(W, X, Yact, Ypred):
    print("R2 = %f, MSE = %f" % (
        mlr.score(X, Yact),
        metrics.mean_squared_error(
            Yact, Ypred)))


df = pd.read_hdf("DatasetPart2.h5", key='ML')

x = df[['NEXTDAYPRECIPAMT2', 'TMIN2', 'TMIN',
        'WT162','TMAX', 'PRECIPFLAG2', 'TMAX2',
        'WT16', 'WT182',
        'PRCP2', 'NEXTDAYPRECIPAMT',
        'PRECIPAMT2',
        'PRECIPFLAG', 'WT18', 'PRCP',
        'TAVG2',
        'TAVG', 'PRECIPAMT']]
y = df['NEXTDAYPRECIPAMT3']

scaler = preproc.MinMaxScaler()
(x_train, x_test, y_train, y_test) = \
    model_selection.train_test_split(
        x, y, test_size=1/3, random_state=0)

scaler.fit(x_train)
scaler.fit(x_test)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)

print()
t1 = time.time()
reg = LinearRegression().fit(
    x_train, y_train)
print(time.time() - t1)




y_pred = reg.predict(x_test)
print("MSE Without Polynomial Test: ",
      metrics.mean_squared_error(
          y_test, y_pred))
print("R2 Without Polynomial Test Score: ",
      reg.score(
          x_test, y_test))
print("R2 Without Polynomial Training Score: ",
      reg.score(x_train, y_train))

poly = preproc.PolynomialFeatures(1)
bigTrainX = poly.fit_transform(x_train)
bigTestX = poly.fit_transform(x_test)
mlr = linmod.LinearRegression()

t = time.time()
mlr.fit(bigTestX, y_test)
print("Test Fit Time", time.time() - t)

t = time.time()
mlr.fit(bigTrainX, y_train)
print("Training Fit Time", time.time() - t)
print()

showStats(
    np.append(np.array(mlr.intercept_), mlr.coef_)
    bigTestX, y_test, mlr.predict(bigTestX))
print()
showStats(
    np.append(np.array(mlr.intercept_), mlr.coef_)
    bigTrainX, y_train, mlr.predict(bigTrainX))
```

RidgeCV Regression - Part 2

```
import time

from sklearn import preprocessing \
    as preproc
from sklearn import linear_model \
    as linmod
import pandas as pd
from sklearn.linear_model \
    import RidgeCV
from sklearn import \
    model_selection
from sklearn import metrics
import numpy as np

def showStats(W, X, Yact, Ypred):
    print("R2 = %f, MSE = %f" % (
        mlr.score(X, Yact),
        metrics.mean_squared_error(
            Yact, Ypred)))


df = pd.read_hdf("DatasetPart2.h5", key='ML')

x = df[['NEXTDAYPRECIPAMT2', 'TMIN2', 'TMIN',
        'WT162', 'TMAX', 'PRECIPFLAG2', 'TMAX2',
        'WT16', 'WT182', 'PRCP2', 'NEXTDAYPRECIPAMT',
        'PRECIPAMT2', 'PRECIPFLAG', 'WT18', 'PRCP',
        'TAVG2', 'TAVG', 'PRECIPAMT']]
y = df['NEXTDAYPRECIPAMT3']

scaler = preproc.MinMaxScaler()
(x_train, x_test, y_train, y_test) = \
    model_selection.train_test_split(
        x, y, test_size=1/3, random_state=0)

scaler.fit(x_train)
scaler.fit(x_test)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)

print()
t1 = time.time()
reg = RidgeCV().fit(x_train, y_train)
print(time.time() - t1)

y_pred = reg.predict(x_test)

print("MSE Without Polynomial Test: ",
      metrics.mean_squared_error(y_test, y_pred))
print("R2 Without Polynomial Test Score: ",
      reg.score(x_test, y_test))

print("R2 Without Polynomial Training Score: ",
      reg.score(x_train, y_train))

poly = preproc.PolynomialFeatures(1)
bigTrainX = poly.fit_transform(x_train)
bigTestX = poly.fit_transform(x_test)
mlr = linmod.RidgeCV()

print()
t = time.time()
mlr.fit(bigTestX, y_test)
print("Test Fit Time", time.time() - t)


t = time.time()
mlr.fit(bigTrainX, y_train)
print("Training Fit Time", time.time() - t)


showStats(np.append(
    np.array(mlr.intercept_), mlr.coef_),
    bigTestX, y_test, mlr.predict(
        bigTestX))

showStats(np.append(
    np.array(mlr.intercept_), mlr.coef_),
    bigTrainX, y_train, mlr.predict(
        bigTrainX))
```