Machine Learning Optimization for Rain in Australia Dataset for predicting rain the next day

Srinivasaraghavan Seshadri[1]

Abstract The objective of this paper is to optimize machine learning algorithms to achieve a decent prediction of the rainfall in the following dataset which is a classification type. Different classification algorithms have been run after a series of preprocessing techniques, along with feature selection, hyperparameter tuning and research in KNN imputation technique for categorical data as well which resulted in the development of a robust universal KNN imputer.

1 Introduction

DATASET: Rain in Australia dataset contains about 10 years of daily weather observations from many locations across Australia. It contains the following features- temperature, rainfall, evaporation, sunshine, wind gust, wind direction, wind speed, humidity, pressure, cloud and rainfall on that day and the target class column consists if there will be a rainfall the next day.

The dataset contains 142193 rows and 24 columns out of which 85773 rows contain empty values (almost 60%).

1 Cork Institute of Technology, Cork, IE
srinivasa.raghavan@mycit.ie

The task is to deal with this dataset and perform the above-mentioned objective.

2 Research

This dataset contains plenty of rows that contain at least one empty feature. The imputation techniques for this dataset would be a great area to research in.
SK-Learn's KNN imputer works only on numerical data. The only available function in SK-Learn is the simple Imputer which can fill the categorical data with the most frequent one. In this part, we research the above-mentioned technique to build a robust KNN Imputer which can handle datasets with multiple missing values in multiple features even randomly. The test will be done to see if there is any performance improvement over the traditional technique.

Please refer to Part3.ipynb. The following [<Cell number>] corresponds to the cell number in the appropriate ipynb. For example [1-2] corresponds to the cells 1 and 2 in the ipython notebook.

2.1 The whole process

[1-2]After importing all required libraries, the given dataset is loaded into a pandas DataFrame. The RISK_MM and Dates columns were removed since dates aren't necessary because we aren't dealing with time series techniques and the risk_mm should be removed since it has feature importance of 1, meaning that it directly corresponds to the target class.

[3]Any features containing 30% or more empty values have been removed

[4] Label Encoding is done for every categorical feature ignoring the empty values

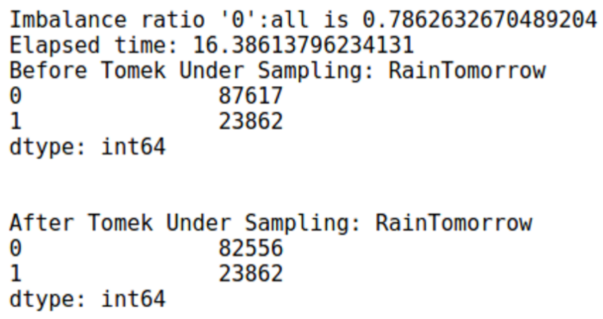
[5] KNN imputer is run to fill up all missing values, even the numerically encoded categorical values are imputed, rounded and converted into integers.

[6] Isolation Forest algorithm is used to remove 2% of outliers. It won't matter much even if a little bit of normal data gets lost due to the huge number of data points.

[7] Splitting the dataset into training and testing data (80% & 20% respectively).

[8] Performing MinMax scaling which rescaled the individual values in between 0 and 1.

[9] Tomek Link undersampling is done which removes the data points near the classification border so that the machine will have a clearer classification boundary resulting in better performance with the 78% imbalanced data.



```
Imbalance ratio '0':all is 0.7862632670489204
Elapsed time: 16.38613796234131
Before Tomek Under Sampling: RainTomorrow
0               87617
1               23862
dtype: int64


After Tomek Under Sampling: RainTomorrow
0               82556
1               23862
dtype: int64
```

[10]It contains a function that'll run the chosen best 3 tuned classifier algorithms(Quaradic Discriminative Analysis, Logistic Regression and AdaBoost Classifier).

[11]The data is tested with the 3 algorithms and results are noted down as follows (detailed classification report in ipynb).

```
Tuned QDA 0.8334409759598134
Tuned LR 0.8406171510584858
Tuned AB 0.8477215644061715
```

[13-14] A function to perform feature selection using the K-best algorithm is tested on a range of K values and the reasonable k = 10 value was found out for the top 10 weighted features.

[16] The algorithms are re-run with the feature selected data and the results are as follows.

```
Tuned QDA 0.835378543236455
Tuned LR 0.841227125941873
Tuned AB 0.8459275206315034
```

[17] A robust KNN based imputer was built as a function knn_own_imputer. It does the following. It accepts data with numerical and categorical features. It label encodes the categorical features, then it'll ber un by KNN imputer. The categorical values will be rounded off and matched to the closest value possible in the above classes. The categorical values will be decoded into a string before returning. This is a robust method and has given higher accuracy in the model when compared to the median method for numeric data and most frequent for categorical data.

[18-20] Testing out the above function.

2.2 Our knn_own_imputer function code

```
def knn_own_imputer(dat):
    #to convert the given data into a pandas DataFrame
    if isinstance(dat,pd.core.frame.DataFrame):
        df_2 = dat
    else:
        df_2 = pd.DataFrame(dat)
    #To print the number of empty values
    try:
        init_null = df_2.isnull().sum()
        print("NULL Value report of input:", init_null)
    except:
        pass
    #dataframes being converted in to appropriate data types
    df_2 = df_2.convert_dtypes()
    df_encoded = df_2.copy()
```

```
   encoding_info = {}#stores the individual column's
transformational models so that it can be inversed later
   encoding_info_classes = {}#stores the individual column's
encoding details
   #Loading the label encoder
   le = preprocessing.LabelEncoder()
   #Encoding the catagorical features excluding the empty values
   for i in df_encoded.columns[df_encoded.dtypes=='string']:
      fit_by = pd.Series([j for j in df_2.loc[:,i] if isinstance(j,str)])
      le.fit(fit_by)
      encoding_info[i]=le
      encoding_info_classes[i] = le.classes_
      df_encoded[i] = fit_by.apply(lambda x: le.transform([x])[0] if
type(x) == str else x)
   #KNN Imputing the empty values
   imp1 = KNNImputer()
   df_knn_imputed_float_columns =
imp1.fit_transform(df_encoded)
   df_knn_imputed_float_columns =
pd.DataFrame(df_knn_imputed_float_columns, columns =
list(df_2.columns))
   df_new = df_knn_imputed_float_columns
   #Function to match a value to the closest value in the list
   def closest(lst, K):
      return lst[min(range(len(lst)), key = lambda i: abs(lst[i]-K))]
   #Converting the imputed continuous data back into catagorical by
rounding them
   for i in encoding_info.keys():
      df_new[i] = np.round(df_new[i].values,0).astype(int)
   #Matching the rounded values to the closest available classes and
decoding them
   for i in encoding_info_classes.keys():
      encoding_info[i].classes_ = encoding_info_classes[i]
      df_new[i] = df_new[i].apply(lambda x: encoding_info[i]\
            .inverse_transform([x])[0] if x in
np.arange(len(encoding_info[i].classes_))\
                        else
closest(np.arange(len(encoding_info[i].classes_)),x))
```

```
    #To print the number of empty values
    try:
        final_null = df_new.isnull().sum()
        print("\n\nNULL Value report of input:", final_null)
    except:
        pass

    return df_new
```

3 Methodology

Part 1: Establishing a Baseline

1.1. Pre-processing:
   - 1.1.1. Removing unwanted columns -RISK_MM and Dates
   - 1.1.2. Deleting features which contain more than 30% empty values
   - 1.1.3. Categorical Features imputed with most frequent technique
   - 1.1.4. Numerical Features imputed with the median strategy to avoid any skew caused by outliers if any
   - 1.1.5. Label Encoding
   - 1.1.6. Outlier removal using Isolation Forest algorithm
   - 1.1.7. MinMax Scaling performed
   - 1.1.8. Tomek Links under sampling

1.2. Building a wide range of ML models
   - 1.2.1. Test out untuned K Nearest Neighbours, Linear SVC(Support Vector Machine), RBF SVC, Decision

tree, Random Forest, AdaBoost, GaussianNB, Quadratic Discriminant Analysis, Logistic Regression classifiers to obtain a baseline

1.3. Hyper-parameters tuning
    1.3.1. The top 3 algorithms were chosen and HP tuned with GridSearchCV (Quadratic Discriminant Analysis, Logistic Regression, AdaBoost)

1.4. The selected best hyperparameters were used in the selected models to run using the dataset and baselines were noted

Part 2: Basic Experimentation - The same as above except

1.1.1. One Hot Encoding was done instead out Label Encoding
1.1.2. SMOTE synthetic oversampling was done for the minority class instead of Tomek Links
1.1.3. K-Best univariate feature selection was performed for various k values and the reasonably best one was chosen

Part 3: Research

1.1. Pre-processing:
    1.1.1. Removing unwanted columns -RISK_MM and Dates
    1.1.2. Deleting features which contain more than 30% empty values
    1.1.3. Label Encoding
    1.1.4. KNN Imputation
    1.1.5. Outlier removal using Isolation Forest algorithm
    1.1.6. MinMax Scaling performed
    1.1.7. Tomek Links under sampling

The 3 selected algorithms with selected best hyperparameters were run and tested out

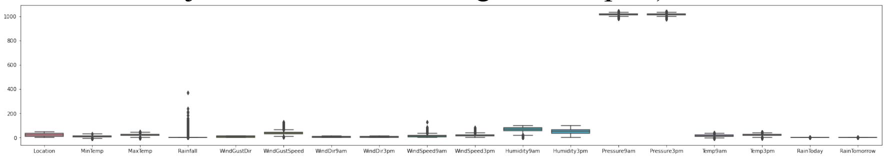
1.2. Building a robust KNN Imputer

4 Evaluation

Part 1: Establishing a Baseline

Please refer to Part1.ipynb
[2]4 features which had more than 30% empty data was removed

[7]After Removal of Outliers:
Box plot for outlier visualization (It was zoomed in and analyzed. It is zoomed out just for accommodating in the report)



[10]Before and after Tomek Link Sampling

```
Imbalance ratio '0':all is 0.7846988496508766
Before Tomek Sampling: RainTomorrow
0               87501
1               23978
dtype: int64


After Tomek Sampling: RainTomorrow
0               82939
1               23978
dtype: int64
```

[11]Baseline Outputs of Various Algorithms

```
Elapsed time: 0.2759127616882324
[[19031  2815]
 [ 2786  3238]] Naive Bayes 0.7990312163616792
Elapsed time: 0.8079788684844971
[[19657  2189]
 [ 2839  3185]] QDA 0.8195909580193756
Elapsed time: 2.993070363998413
[[20543  1303]
 [ 3128  2896]] Logistic Regression 0.8410118406889128
Elapsed time: 3.8424761295318604
```

```
[[18548  3298]
 [ 2864  3160]] Decision Tree 0.7789020452099031
Elapsed time: 7.877079725265503
[[20586  1260]
 [ 3105  2919]] AdaBoost 0.8433799784714747

Elapsed time: 20.904700994491577

[[20262  1584]
 [ 3048  2976]] Nearest Neighbors 0.833799784714747
Elapsed time: 24.686269283294678
[[20666  1180]
 [ 2944  3080]] Random Forest 0.852027269465375
Elapsed time: 192.30679392814636
[[20732  1114]
 [ 3312  2712]] Linear SVM 0.8411912450663797

Elapsed time: 252.6173026561737
[[20965   881]
 [ 3349  2675]] RBF SVM 0.8482238966630786
```

[13]Tuning AdaBoost Model- Best HyperParameters

```
AB Best: 0.850333 using {'learning_rate': 1.0, 'n_es
timators': 500}
```

[14]Tuning Logistic regression – Best Hyperparameters

```
LR Best: 0.842308 using {'C': 10, 'penalty': 'l2'}
```

[15]Tuning QDA – Best Hyperparameters

```
QDA Best: 0.832478 using {'reg_param': 0.001}
```

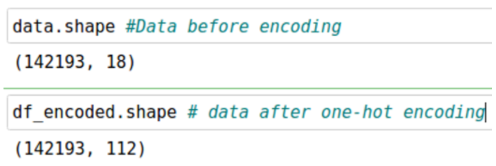
[17]Outputs with Tuned Classifiers

```
[[19732  2114]
 [ 2729  3295]] Tuned QDA 0.8262289199856476
[[20530  1316]
 [ 3119  2905]] Tuned LR 0.8408683171869393
[[20624  1222]
 [ 3000  3024]] Tuned AB 0.8485109436670255
```

Part 2: Basic Experimentation - The same as above except

Please refer to Part2.ipynb

1.1. One Hot Encoding was done instead out Label Encoding



```
data.shape #Data before encoding
(142193, 18)
```

```
df_encoded.shape # data after one-hot encoding
(142193, 112)
```

1.2. SMOTE synthetic oversampling was done for the minority class instead of Tomek Links

```
Imbalance ratio '0':all is 0.7863350293148856
Elapsed time: 169.81941986083984
Before SMOTE Over Sampling: Rain_Tomorrow
0                 87737
1                 23742

After SMOTE Over Sampling: Rain_Tomorrow
1                 87737
0                 87737
dtype: int64
```

1.3. K-Best univariate feature selection was performed for various k values and the reasonably better one was chosen in terms of accuracy, temporal and computational cost

Performance Before Feature Selection:

```
[[12439  9399]
 [ 1525  4507]] Tuned QDA 0.6080373161105131
[[17325  4513]
 [ 1499  4533]] Tuned LR 0.7842841765339075
[[19539  2299]
 [ 2415  3617]] Tuned AB 0.8308575529242913
```

Performance after Feature Selection

```
K =  10
X_train shape: (175474, 111) X_train_featured shape (175474, 10)
X_test shape: (27870, 111) X_test_featured shape (27870, 10)
[[17336  4502]
 [ 1770  4262]] Tuned QDA 0.7749551489056333
[[17051  4787]
 [ 1561  4471]] Tuned LR 0.7722282023681378
[[18959  2879]
 [ 2235  3797]] Tuned AB 0.8165052027269465
```

Part 3: Research

Please refer to Part3.ipynb

The preprocessing techniques were done along with proposed research method and the results follow

After Tomek Link Under Sampling:

```
Imbalance ratio '0':all is 0.7862632670489204
Elapsed time: 16.38613796234131
Before Tomek Under Sampling: RainTomorrow
0               87617
1               23862

After Tomek Under Sampling: RainTomorrow
0               82556
1               23862
```

Run with the selected models with the tuned hyperparameters after Tomek Sampling

```
[[20487  1461]
 [ 3181  2741]] Tuned QDA 0.8334409759598134
[[20550  1398]
 [ 3044  2878]] Tuned LR 0.8406171510584858
[[20592  1356]
 [ 2888  3034]] Tuned AB 0.8477215644061715
```

Performance after Feature Selection :

```
K =  10
X_train shape: (106418, 17) X_train_featured shape (106418,
10)
X_test shape: (27870, 17) X_test_featured shape (27870, 10)
[[20626  1322]
 [ 3266  2656]] Tuned QDA 0.835378543236455

[[20566 1382]

   [ 3043  2879]] Tuned LR 0.841227125941873
  [[20595  1353]
   [ 2941  2981]] Tuned AB 0.8459275206315034
```

5 Conclusion

Part 1:

Naïve Bayes, QDA, Logistic Regression, Decision Tree are the fastest to compute accordingly followed by AdaBoost, k Nearest Neighbours, Random Forest, Linear SVM and RBF SVM.

Tuning the Hyperparameters didn't show a marginal improvement due to the nature of the dataset with multiple missing values which were later imputed with not so robust algorithm.

Part 2:
One Hot Encoding increased the number of features by almost 5 times which increased the computational and temporal time for all the following operations highly significantly.

SMOTE oversampling further increased the number of records increasing the costs.

Further feature selection reduced the costs, but showed varying accuracies depending upon the algorithm used – QDA showed a significant performance improvement but the other 2 showed a slight decrease in accuracy.

Part 3:
After our own KNN based imputation, QDA showed a slight increase in performance but the other 2 didn't much difference. Even though this technique didn't show much different due to the nature of the dataset, it could be beneficial in some cases – yet to be tested out in various datasets.

Overall this is a highly complex dataset with a huge number of records with high bias, it would be highly optimal to use Neural Networks for it.

We also built a robust KNN imputer in which a dataset can be passed without any encoding, it'll do everything, impute even the categorical data and return a fully imputed dataset that contains strings in the categorical features and numerical value in the numerical features which might be useful for future applications.

FUTURE WORK:

- To further generalize the own_knn_imputer function
- To build an intelligent imputer using other algorithms

- To optimize the code with pipelines
- To write an intelligent program that’ll automatically tune and test out various combinations of tuning and algorithms, to perhaps using genetic algorithm to escape brute forcing
- To build a program that’ll intelligently automatically pre-process the data intelligently and in the best possible way

References [2]

1. Zhang S (2012) Nearest neighbour selection for iteratively kNN imputation. J Syst Softw. https://doi.org/10.1016/j.jss.2012.05.073

2. Wong WE, Zhao J, Chan VKY (2006) Applying the statistical methodology to optimize and simplify software metric models with missing data. In: Proceedings of the ACM Symposium on Applied Computing

3. Poulos J, Valle R (2018) Missing Data Imputation for Supervised Learning. Appl Artif Intell. https://doi.org/10.1080/08839514.2018.1448143

4. Jordanov I, Petrov N, Petrozziello A (2018) Classifiers Accuracy Improvement Based on Missing Data Imputation. J Artif Intell Soft Comput Res. https://doi.org/10.1515/jaiscr-2018-0002