HW1 Report Approach

B10915008 汪萬丁

A. Program Structure and How to run

The whole program is put inside a Jupyter Notebook file, Weather_Analysis_Updated.ipynb. It is done so that each section of the program (incrementally analysis and model building) can partially gives its output along the execution.

To quickly reproduce the result that is submitted by the model, one can simply Execute all boxes from the top to bottom, and user will get the output csv that is outputted from the model.

From here on out, a quick summarization of each cell in the .ipynb will be stated, detailed explanation will be explained in part B.

1. Do libraries importing, rename each Attribute1 – Attribute17 to self customized names for easy understanding. Convert yes/no entry to 1 and 0 respectively, extract year, day and month from dates.

   Also, define the columnGroups dictionary that will come in handy for fast pace experiment when trying different column inputs.
2. Counting the distribution percentage of when tomorrow is raining or when tomorrow is not raining.
3. Comparing the distribution of the target variable with countplot
4. Comparing the cardinality of station area, strongest wind direction, wind direction, and also whether today is raining or not.
5. Counting how many nulls in percentage.
6. Seeing the correlation heatmap of each variable with each other
7. Plotting the data distribution of each training dataset variable
8. Plotting the data distribution of each test dataset variable
9. Plotting the data distribution of each training dataset variable if data point entry that any of its feature has null (except 3pm temperature and evaporation excluded) is dropped
10. Convert direction of strongest wind dir and 3pm wind dir from string to number encoding.
11. Model experiment and test result output. Many details, so it will be further explained in part B.

B. Program Explanation

Now more detailed explanation will be elaborated for the previous said cells.

1. Importing is done and definition of convertDirectionInt(strIn) is made, which will comfortably convert each string input of direction (a total of 16 directions) into its integer counterpart, used later in 10^{th} cell.

   The train.csv and test.csv are both read, and the features are renamed as follows:

Original Name	Renamed Attribute
Attribute1	date
Attribute2	stationArea
Attribute3	minTemp
Attribute4	maxTemp
Attribute5	rainDrop
Attribute6	evaporation
Attribute7	daySunAppearHour
Attribute8	strongestWindDir
Attribute9	srongestWindSpd
Attribute10	3pmWindDir
Attribute11	3pmPRevAvgTemp
Attribute12	3pmRltvHumid
Attribute13	3pmPrevAtmPres
Attribute14	3pmCloupProp
Attribute15	3pmTemp
Attribute16	TdyRain
Attribute17 (Target)	TomorrowRain (Target)

By default, stationArea since has integer input, will be regarded as numerical type, so I change the datatype of that feature to be 'object' (categorical).

I furthermore extract the year, day, and month from the date, and drop the date. This way, we can easily analyze each entry (especially later for correlation). Additionally

For the "Yes" and "No" inputs in TomorrowRain and TdyRain, I label them to 1 and 0 respectively.

Then, I print out the describe of the dataset of both train and test for comparing, we can see that for their mean, std, min, 25%, 50%, 75% and max has similar for train and test.

The column groups are also defined in advance, so that in the further cells, I can easily group each cell with a 'nickname', then can reference the cell names from the nickname. For example, when I want to call the all of the X variables, I can just easily call columnGroups['XCategoryDefault'].

2. We can see how greatly imbalance the target dataset be, the 0 entry of TomorrowRain has 82% and 1 only placed 18%.
3. Just in case number is hard to show how greatly imbalanced the dataset is, another vertical graph is plot to show the imbalanced y.
4. Shows the cardinality of the categorical variables, which is the stationArea, strongestWindDir, 3pmWindDir, TdyRain. We can see that stationArea has 49 variations, wind direction variables have 17 (16 + 1 for NaN since in next cell we can see that there is NaN). The high cardinality of stationArea might pose problem for model, for now this is only taken as a note to self when further processing.

5. The datasets have few variables which contain NaN. But the worst part is evaporation, daySunAppearHour and 3pmCloudProp has really high percentage (40% more). One might be tempted to directly drop those data, but I will first test using correlation later whether these variables are valuable or not.
6. The important takeaway from this correlation analysis are as follows:
   - maxTemp and 3pmTemp has an almost perfect correlation (0.99), which means this variable might be a duplicate of each other. One of these must be dropped, I decide will drop 3pmTemp to reduce redundant variables.
   - Evaporation, daySunAppearHour, and 3pmCloudProp has a lot of nulls. But the contrast is that the first said variable has a really low correlation with target, while the two others have a relatively high correlation with target (compared with other variables' correlation to target).
   - The newly added year, day, month, looks life have a long relation with the target variable, but further experiment will be tried first whether the result is better or not with these dates included.
7. Some of the variables here are badly skewed, (either left or right skewed). This shows us that it is better for us to do skew fix later, which will be done with PowerTransform.
8. Fortunately, there isn't a test set distribution which has drastically different distribution than the one in training set.
9. I try to drop the datapoints where it has NaN feature (except for 3pmTemp since it is most likely very similar to maxTemp, and also except for Evaporation, because it has a lot of NaN but low correlation). Fortunately, the graph distribution doesn't change much (except for stationArea where the middle section lost a lot of distribution, causing a valley shaped curve). Also, some of the data are still badly skewed, for example raindrop.
10. For now, I will do label encoding to the wind direction variables using the function defined in the first section. There is a possibility that in the end, what I am interested is actually is a one hot encoding not a label encoding to be fed to the model, but it will still output the same result regardless if I directly one hot encoded these variables or I actually do label encoding to integer first then do one hot encoding later.
11. This particular cell is the one that is personally responsible for building the model and final csv. So, first of all, before I begin, I import functions and modules from well known machine learning libraries.

    Before explaining further, I place two very important dictionaries, datasetConfig and also kernelConfig, which can support my fast pace modelling, I first implement all the source code of all process (for example, scaling, power transform, etc), nevertheless whether I end up using it in the end or not. Then, I will use these two config dictionaries to be the indicator whether I want to run that process. Now, each content of the two configs will be explained:

datasetConfig keys	datasetConfig values and meanings
columnGroups	this correspond to the nickname of the group of columns that is defined in first cell, for example: 'XAllNo3pmTempNoStation'
val?	Boolean, True or False, whether we want to split train.csv to train and validation set
valRatio	if val? == True, then this will determine the split ratio
NullKNN?	Boolean, True or False, whether try imputing Null with KNN?
KNNReplacement	If NullKNN is set to True, how many n_neighbors we want for KNNImputer?
FillNumNull	The case if we want to use statistic to fill null, we can choose for "Mean", "Median", "Mode", applied only to numerical columns
FillCatNull	The case if we want to use statistic to fill null of categorical columns, we can choose the same options just like the above options, but often categorical columns are filled with Mode
DropNulls?	Boolean, do we want to just drop all nulls?
skewFix?	Boolean, If the dataset is skewed, do we want to fix it using PowerTransformer? Works either way for left or right skewed.
oneHotEncoding	Boolean, do one hot encoding for categorical variables? (If False, then it will stay as categorical encoding, from cell 10).
Scaler	Scale the data? Four options available: "Standard", "MinMax", "Robust", and None for no scaling.
Oversampling?	Boolean, since our data is greatly imbalanced, do we want to oversample data to make it balance?
Oversampling_method	Only one method is defined, which is "SMOTE"
FeatureSelection?	Boolean, Whether to execute sklearn.feature_selection.RFECV
PCA?	Boolean, whether to execute Principal Component Analysis of the input features
FitWithEval?	Boolean, whether the user want to plot the learning curve for training and

	validation set, need to assert that datasetConfig[“val?”] is True.

kernelConfig keys	kernelConfig values
LazyPredict	Boolean, whether the model used for predicting used the one provided in LazyPredict libraries (do cross validation for 29 sklearn models with default parameter).
OptunaMode	Boolean, whether the pipeline will execute optuna functions (a module widely known to cleverly aid finding the possible choices of best model hyperparameters).
NotOptunaFinalPredict	Boolean, if OptunaMode is False, do we want to output the result of test.csv with the model we have just trained? If we do, set it into True
thresh	For the final model (for displaying training score, validation score, and outputting the final .csv), what threshold do we want to set for setting the result to 1 and 0
Model	Desired Model, only “XGBOOST, “RF” (Random Forest), “LOGREG” (Logistic Regression), and “LOGREG_balanced” (Logistic Regression with class weight used) defined.

Now, the pipeline of the cell is explained:

1. First, extract the column we want to input for the X.
2. Decide whether we want to drop nulls that exist inside X.
3. After then, decide whether we want to do train – validation split. Using random_state = 1 to make sure having same result, and also do stratify split, to maintain the distribution of the target variable on train and validation. It is gamble whether for the validation, we should remain to have an unbalanced target variable, but through the suggestions given online, it is recommended if we have no proof on the test dataset distribution, distribution should be remained as it is, even though it is unbalanced.
4. In the case where number 2 is False, it means there might be possibly null inside X. Decide the method to fill null, either using KNN (both for categorical columns and numerical) or Mean, Mode, Median (can define different method for categorical and numerical column). Make sure that when we use any KNN, Mean, Mode, Median, X that will be used for validation is not involved for fitting, to avoid data leak.

5. If we want to fix the skew, fit PowerTransformer only for numerical data. Remember to not fit validation x to avoid data leak.
6. If we want to scale the data, we can use Robust, Standard or MinMax scaler chosen by the user, scale both numerical and categorical data.
7. If we want to do Principal Component Analysis, fit the PCA() to the numeric columns of X, remember to not fit validation x to avoid data leak.
8. If we want to avoid one hot encoding for the categorical columns, one hot encoding will be done here. Fortunately, all the choices that appear on test.csv appear on train.csv, so one hot encoding can be done simply using function in sklearn OneHotEncoder.
9. Concatenate both the processed category and numerical data of training and validation.
10. If we want to do oversample, which is only defined for SMOTE, to balance the dataset, it is done now. Using random_state = 2 to make sure we have same result no matter how many times we run it.
11. If we want to execute RFECV feature selection from sklearn, it will be done here, and again it is fitted for train data, not involving validation to avoid data leak.
12. From now on, the operation to process the data is finished, and what will remained is what operation we want to do for the dataset. One of the operations below will be chosen:
    a. Execute lazy predict, which is a combination of 29 sklearn models.
    b. Optuna Mode: define the objective in a function (logreg_objective, logreg_balanced_objective, or xgboost objective), which will execute the pipeline defined in a function, and return the prioritized objective that we want to pay special attention. We can choose to find the parameter that appear to make maximum/minimum approach in numerous trial. Here, the return result is set as f1 score, not accuracy score in particular, because unbalanced dataset really has a bad indicator if we use accuracy score (for example if we have 80% answer as 0, when model predict all 0, it already has 80% score).
    c. Regular training pipeline, which will fit a model defined in kernelConfig, to fit the train data. If it is desired, we can find the best hyperparameters found using optuna. Then print the accuracy, precision, recall, and f1 score of train dataset. If val set is defined, then report the same previous scores of val dataset, and also from roc curve false positive rate and true positive rate try to estimate the best binary threshold using G-Mean.

    If NotOptunaFinalPredict is true, then use the previous scalers, transformers, one-hot encoders, etc used for training to

transform test.csv data and predict the result, output to .csv. The name of the csv is set in particular as the config values of the experiment.

C. Submitted Model Explanation

This particular section will explain the two models picked for final submission, along its short reasons. Two of the model config inputs have been pasted inside the last cell comments, namely the “XGBoost best” and “best logreg”. The way I work is using this pipeline:

1. Choose a reasonable dataset column group for X.
2. Experiment the datasetConfig (whether use PCA or not, KNN Input, etc).
3. Use LazyPredict to find the best model with decent score (which ranking is dominated by xgboost, random forest, and logistic regression)
4. Find Optuna of best hyperparameter for that model (best validation f1 score).
5. Find the best threshold using g-means
6. Submit trial to Kaggle.

First Model: XGBoost oriented
Choosen column: XAllNo3pmTempNoStation
Reason: 3pm temp is possibly a duplicate of maxTemp, so it won’t be used. Categorical variables need to be one-hot encoded for XGBoost, including stationArea. But stationArea has a lot of cardinalities, which makes tree-based tree suffer with underfit/overfit of other variables not being pay attentioned (analyzed with feature importance of xgboost). So Station is not used, which yield decent result.

- Numerical: minTemp, maxTemp, raindrop, evaporation, daySunAppearHour, strongestWindSpd, 3pmPrevAvgTemp, 3pmRltvHumid, 3pmPrevAtmPres, 3pmCloudProp, year, month, day
- Categorical: strongestWindDir, 3pmWindDir, TdyRain

Handling Null: dropping nulls, filling nulls with KNN has been tried, but yield worse result, the approach fills categorical nulls with mode and numerical nulls with median, since some of the variables are badly skewed, which makes using median more durable when meeting outlier in data compared to mean.

Fix the skew with PowerTransformer, since tree-based model is affected with skewed data.

One hot encoding is done, to show that categorical variable is not in any order/strength

Although scaler is not necessarily needed for tree-based model, but robust scaler is used because robust can handle outlier better compared to other scaler, which is a good approach since some of the dataset is badly skewed.

Oversample is used to make even distribution dataset, because uneven dataset will most likely result uneven tree structure. Oversample is used as opposed to undersample to make more test cases.

The given best threshold, as estimated by g-means, is 0.2.

Second Model: Logistic Regression oriented
The first model is using tree based, the second model is tried to approach using non tree, which is logistic regression.

Choosen column: XAllNo3pmTempEvaporation
Reason: This particular example will try to drop the nulls, to test whether if we use (considered, since we have no info to check the precision of nonnull data in real life) precise data. As usual, 3pmTemp is almost a duplicate of maxTemp so it is dropped, then stationArea, in contrast with the previous model, can actually still handled well in logistic regression, so it is kept. Since we want to drop nulls, we need to minimize null data, and evaporation is one of the columns that has a lot of null, but least correlation to target variable. To prevent a lot of data being drop due to evaporation, which might not be that significant entry, evaporation is not used.

- Numerical: minTemp, maxTemp, raindrop, daySunAppearHour, strongestWindSpd, 3pmPrevAvgTemp, 3pmRltvHumid, 3pmPrevAtmPres, 3pmCloudProp, year, month, day
- Categorical: stationArea, strongestWindDir, 3pmWindDir, TdyRain

Handling Null: As explained, this particular example will try to drop nulls, see if using a real entry, not entry filled with statistic, yield better result.

Fix the skew with PowerTransformer, since logistic regression model is affected with skewed data.

One hot encoding is done, to show that categorical variable is not in any order/strength.

Scaler is crucially needed to converge for logistic regression. The approach will use Standard Scaler, because PCA will be used later, and it needs data to be scaled to mean = 0 and unit variance.

Oversample is used to make even distribution dataset, because uneven dataset influence logistic regression. Another approach has been tried using class weight for logistic regression, but didn’t yield better result.

Use PCA because high cardinality input might pose problem for logistic regression, and logistic regression will use every input variable in the

beginning, as opposed to tree based where they can choose which variable they use.

The given best threshold, as estimated by g-means, is 0.4633.