Predicting Rain in Australia: Machine Learning Project Report

August 19, 2022

1 Introduction

In this report, we will be describing the experiments we ran on the dataset "weather AUS", aiming to predict whether or not it rained on the next day of a given day based on the weather of that day, its location and date. These experiments were conducted as part of the Machine Learning course for the Master of Digital Text Analysis in the University of Antwerp, Belgium.

2 Dataset

2.1 Context

The dataset, found on Kaggle, is called "Rain in Australia", and contains 10 years of daily weather observations from many locations across Australia. The variable we're aiming to predict is "RainTomorrow", which is a binary variable pertaining to whether or not it rained the next day.

2.2 Data Exploration

The first step when tackling any new dataset is data exploration, so we can get a better idea of what our data looks like. Our dataset includes 145460 rows, of which only 56420 remain after dropping missing values. Out of 23 columns, 5 are categorical, which we will need to process differently than the numerical columns.

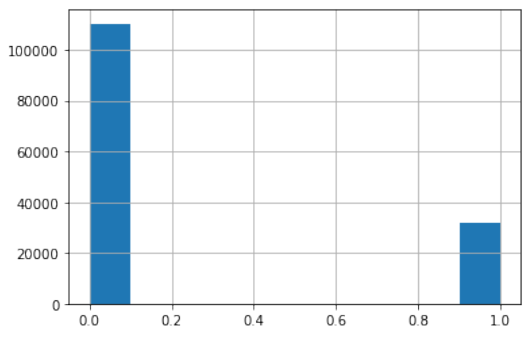


Figure 1: Data Imbalance

Our data is quite imbalanced, with only 22% of our rows being days where it rains the next day (after dropping missing values).

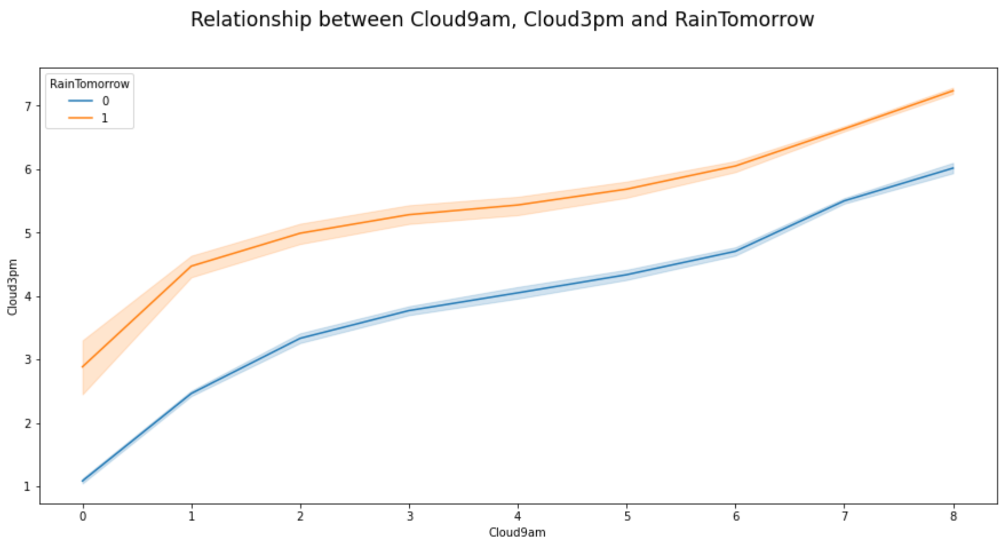


Figure 2: Clouds and RainTomorrow

We can observe in figure 2 that there seems to be a correlation between the amount of clouds the previous day, and RainTomorrow.

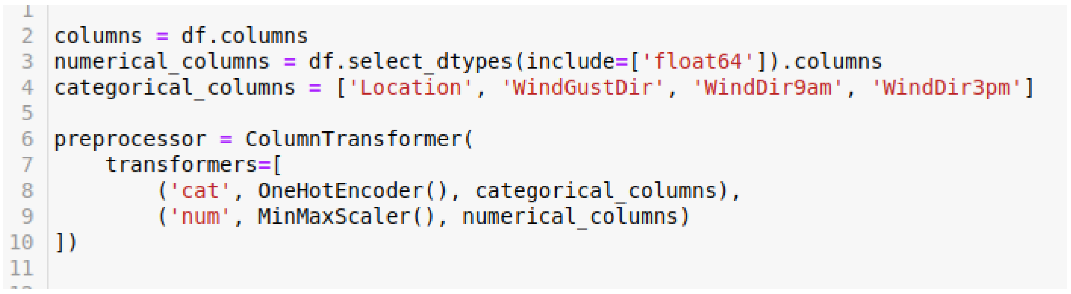
3 Classical Model

3.1 Preprocessing

Our preprocessing for the classical model was relatively simple, as the scaling of the numerical columns and the One-Hot Encoding of the categorical variables happened inside the Column Transformer. First, we extracted Year, Month and Day from our "Date" column. Then, we dropped all missing values, arguing that the 56420 remaining columns are still enough data to train and test our models with. And finally, we converted the "RainToday" and "RainTomorrow" columns into binary variables.

3.2 Column Transformer and Testing 5 models

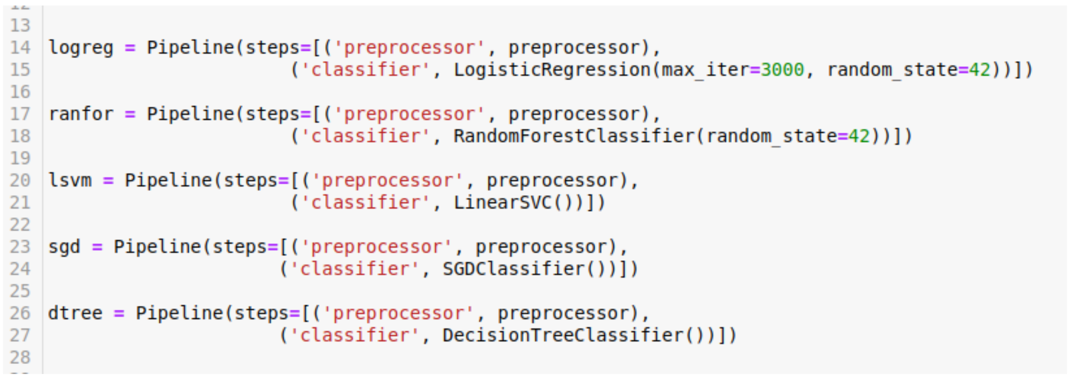
Rather than applying a One-Hot Encoder on the entire dataset, and writing more code to scale the train and test set while fitting only the train set, a Column Transformer was used for the Classical Model.



```python

columns = df.columns
numerical_columns = df.select_dtypes(include=['float64']).columns
categorical_columns = ['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm']

preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(), categorical_columns),
        ('num', MinMaxScaler(), numerical_columns)
])

```

Figure 3: Column Transformer

The `OneHotEncoder()` was applied to the categorical columns, and the `MinMaxScaler()` was applied on the numerical columns.



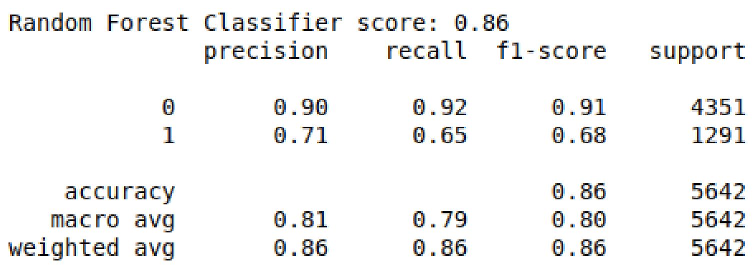
```

logreg = Pipeline(steps=[('preprocessor', preprocessor),
                        ('classifier', LogisticRegression(max_iter=3000, random_state=42))])

ranfor = Pipeline(steps=[('preprocessor', preprocessor),
                        ('classifier', RandomForestClassifier(random_state=42))])

lsvm = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', LinearSVC())])

sgd = Pipeline(steps=[('preprocessor', preprocessor),
                     ('classifier', SGDClassifier())])

dtree = Pipeline(steps=[('preprocessor', preprocessor),
                       ('classifier', DecisionTreeClassifier())])

```

Figure 4: Testing 5 models

Five classical models with no special parameters were then tested on the dataset with the Column Transformer, to find the two best-performing models. Out of the five tested models, the Random Forest Classifier had the best results, by only a small margin (macro f1 score of 78 and accuracy of 87, where three other models had a macro f1 of 78 and an accuracy of 86).

3.3 Randomized Search

After identifying the best models, a `RandomizedSearchCV()` was performed on both the `RandomForestClassifier()` and `LogisticRegression()`. For both models, the hyper-parameters identified by the `RandomizedSearchCV()` were then used to make predictions, resulting in only marginally better results for the `RandomForestClassifier()`, going from a macro f1 of 78 to 80.



```
Random Forest Classifier score: 0.86
              precision    recall  f1-score   support

           0       0.90      0.92      0.91      4351
           1       0.71      0.65      0.68      1291

    accuracy                           0.86      5642
   macro avg       0.81      0.79      0.80      5642
weighted avg       0.86      0.86      0.86      5642
```

Figure 5: Optimized Random Forest Classifier

3.4 Undersampling Train Set

Since our data is imbalanced, and our model's recall of the less common variable (rainy days) is low, we trained our 5 basic models on a undersampled training set. By reducing the amount of non-rainy days in our training data to be the same amount as the rainy days, we hoped to train our model to predict the rainy days more accurately. The results showed a substantially higher recall for rainy days, but a lower precision, and overall a slightly lower macro f1 score.

3.5 Data Subsections

Our dataset is not only imbalanced, but it also suffers from a potential bias: it's possible that our model is just correctly predicting days that had the same rain situation as the day before, rather than correctly identifying a change from Rainy day -¿ Non-rainy day or the other way around. In order to explore these shortcomings, our 5 basic models were further tested on 2 subsets of our data: one where we kept only days different than the day before and another where we kept only days same as the day before. The overall accuracy and the macro f1 score both drop considerably.

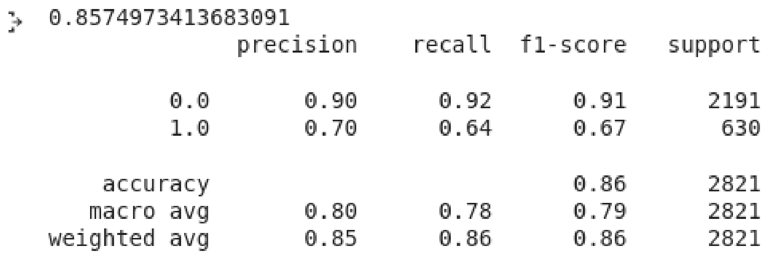
4 Neural Model

4.1 Preprocessing

For the neural model preprocessing, the "Date" column was dropped from the start, as well as the rows with missing values. Unlike with the classical model, the categorical columns were encoded with a Label Encoder. The data was split into train and test, and then the train set was further split into a validation set, which we used to test the model when fitting it. A `StandardScaler()` was then use to scale the dataset: fitting only on the training set and then transforming the train set, test set and validation set. After this, the scaled numerical columns were merged with the unscaled and label encoded categorical columns. Finally, we converted the train set, test set and validation set to a numpy array of "float32" type, as is important to do for `keras`.

4.2 Model

First, we built a simple recurrent neural network (RNN) model. We added a Dropout layer to prevent overfitting after finding our model had been doing so. We used the binary crossentropy loss, adam optimizer and accuracy as a metric. We used 20 epochs, and an Early Stopping patience of 5, which stopped our epochs at 13. When trying to run the first classification report, I ran into the following error: "ValueError: Classification metrics can't handle a mix of multilabel-indicator and multiclass targets", which was solved by adding a threshold of 0.5 and converting the predictions to a binary. After much trial and error, including undersampling the dataset like we did for the classical model, trying different scalers and trying with and without our categorical columns, we reached a macro f1 score of 78, which is lower than the best result from the classical model.



```
0.8574973413683091
              precision    recall  f1-score   support

         0.0       0.90      0.92      0.91      2191
         1.0       0.70      0.64      0.67       630

    accuracy                           0.86      2821
   macro avg       0.80      0.78      0.79      2821
weighted avg       0.85      0.86      0.86      2821
```

Figure 6: Neural Binary Classifier

We also tried running a binary classification model, with the same compilation settings as the RNN model (finding them to have the best accuracy and loss curves, after some trial and error). For this model, we achieved a macro f1 of 79, which is a bit higher than the RNN model, but still lower than the best classical model.

5 Discussion

This task has undoubtedly been a challenge, and an excellent learning opportunity. Avoiding data leakage was a challenge, as was dealing with a dataset with both categorical and numerical variables. Combining the column transformer with the randomized search also proved difficult, but it was an informative quest. For the neural model, I did a lot of trial and error, and would have liked to find a way to do something like a randomized search.