Rain prediction in Australia

The dataset contains daily weather observations of Australian weather stations, and the goal here is to predict whether it will rain tomorrow based on the data. The target variable for prediction is “RainTomorrow”.

Analysis

In this analysis, I’m trying to test the various different Hyperparameter tuning techniques rather than just using GridSearchCV that I have used so far.

So I’ll pick a simple Logistic regression classifier and use both L1 (Lasso) and L2 (Ridge) regularization and will check and compare the results by using GridSearchCV, RandomizedSearchCV and Bayesian Optimization for hyperparameter tuning.

GridSearchCV can be extremely resource intensive. RandomizedSearchCV, as the name suggests, randomly selects hyperparameter combinations, unlike GridSearchCV where every possible combination is used. Bayesian Optimization, on the other hand, can help in getting the optimal combination compared to GridSearchCV or RandomizedSearchCV as it takes a probability based approach and learns from successes and failures to find appropriate combination in an efficient way.

As usual after reading the dataset, I check for the different columns and the datatypes in each column. There are a total of 23 columns and 40,000 rows. There are no missing values in the dataset -

```
In [8]: print(missing_values)
Date             0
Location         0
MinTemp          0
MaxTemp          0
Rainfall         0
Evaporation      0
Sunshine         0
WindGustDir      0
WindGustSpeed    0
WindDir9am       0
WindDir3pm       0
WindSpeed9am     0
WindSpeed3pm     0
Humidity9am      0
Humidity3pm      0
Pressure9am      0
Pressure3pm      0
Cloud9am         0
Cloud3pm         0
Temp9am          0
Temp3pm          0
RainToday        0
RainTomorrow     0
```

I strip the specific details such as day of the week, month, year, etc from the Date column into additional columns and then drop the Date column. The categorical columns such as Location and the 3 columns related to wind direction are encoded using OneHotEncoding.

Finally, after converting the RainToday and RainTomorrow columns into binary form, I scale the variables using StandardScaler. And then the dataset is split into 80% train set and 20% test set.

The logistic regression classifier with L1 (Lasso) and L2 (Ridge) regularization is used to fit the training datasets.

Finally for Hyperparameter tuning, I try different levels of regularization strength and max_iter params in GridSearchCV and RandomisedSearchCV using 5-fold cross validation. The regularization strengths used in GridSearchCV vary from 0.01 (strong regularization) to 100 (low regularization) in multiples of 10. The low regularization strengths imply it can lead to overfitting if the model is complex, whereas for stronger regularization, the model is penalised more for large coefficients. While RandomizedSearchCV samples a range of values for C to quickly explore the hyperparameter space. It does this by testing randomly chosen values and allows us to explore a wide range of values without checking every single possibility. Hence, by using values from np.logspace(-4, 4, 20), I'm trying a range of C values from 0.0001 to 10000 on a logarithmic scale.

Similarly, for Bayesian Optimization as well, I choose the 2 types of regularizations in Logistic regression and use the regularization strength C and max_iter by providing certain bounds within which the bayesian optimization explores and finds the best parameters. The optimization happens in the maximize() function for each Lasso and Ridge regularization where it first picks 5 random points (init_points =5) and then performs 50 more iterations (n_iter = 50) to find the best parameters.

After each combination of different hyperparameter tuning techniques and different regularization within Logistic regression is explored, I get the best parameters for each and evaluate the model combinations by checking for test accuracy.

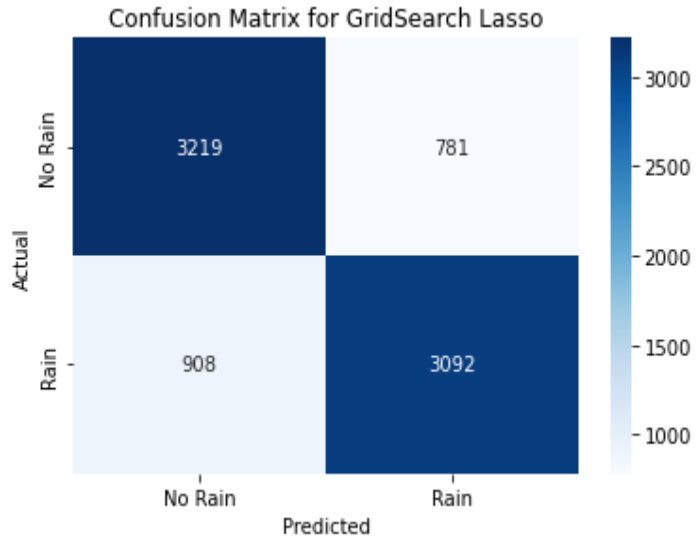
Model	**Train Accuracy**	**Test Accuracy**	**Best Hyperparameters**
Lasso (L1)	79.03%	78.75%	{'C': 0.1, 'max_iter': 100}
Ridge (L2)	79.04%	78.77%	{'C': 0.1, 'max_iter': 100}
GridSearchLasso	78.89%	78.89%	{'C': 0.1, 'max_iter': 100}
GridSearchRidge	78.74%	78.74%	{'C': 0.1, 'max_iter': 100}
RandomizedSearchLasso	78.75%	78.75%	{'C': 1.623776739188721, 'max_iter': 100}

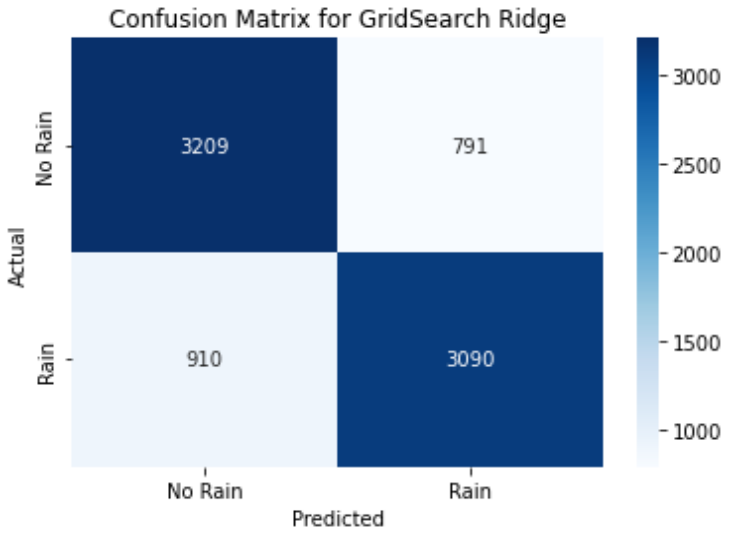
RandomizedSearchRidge	78.74%	78.74%	{'C': 0.08858667904100823, 'max_iter': 100}
BayesianSearchLasso	78.725%	78.725%	{'C': 233964.14931854614, 'max_iter': 9553}
BayesianSearchRidge	78.725%	78.725%	{'C': 828438.0027438939, 'max_iter': 3072}

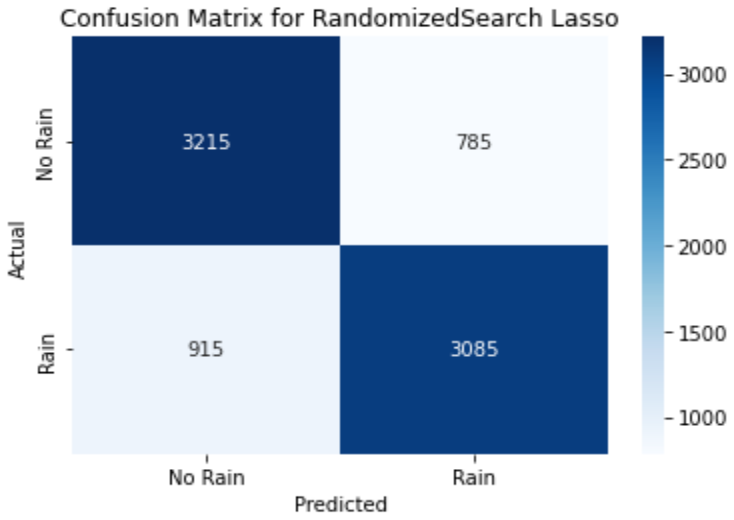
From the above table, we can see both Lasso (L1) and Ridge (L2) perform similarly in terms of train accuracy and test accuracy, indicating that neither regularization method drastically outperforms the other. Test accuracy for all models falls between 0.78725 and 0.7889, which is quite similar across different hyperparameter tuning methods.

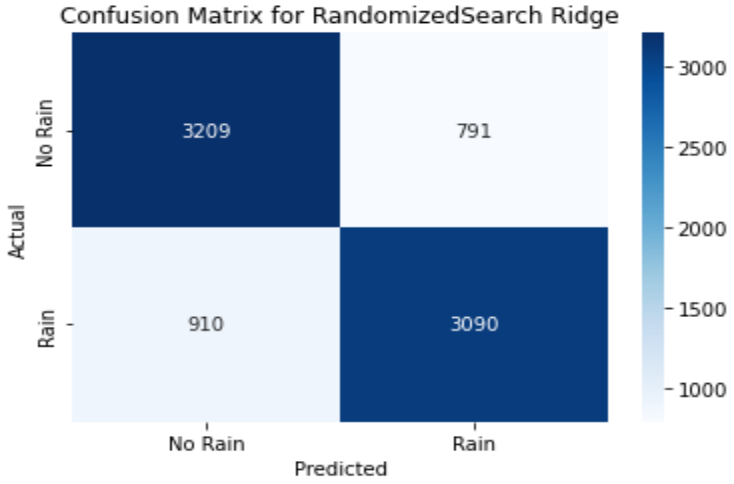
GridSearchCV results are very close to those of RandomizedSearchCV. It also gives more intuitive and smaller values for C (e.g., C = 0.1), which works well for both Lasso and Ridge. RandomizedSearchCV also yields similar test accuracies and hyperparameter results. Bayesian Optimization, on the other hand, leads to very high values of C, which isn't very intuitive and may also be the reason for the slight reduction in test accuracy. This might suggest that Bayesian Optimization might have overfit the data.

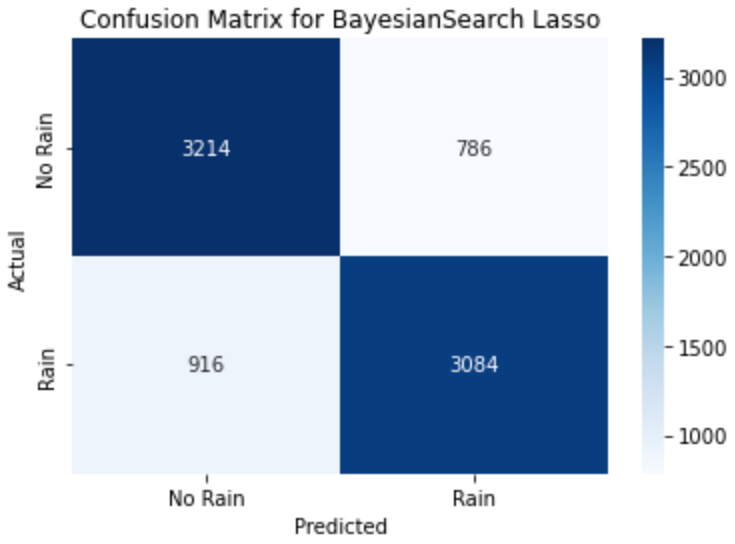
Finally, I have again consolidated the list of models with hyperparameter tuning using the best models and plotted the confusion matrix for each model as can be seen in the plots below-

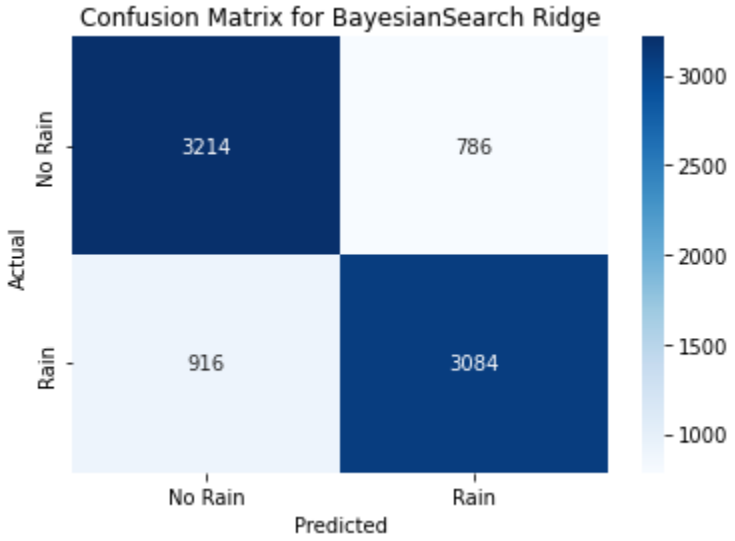



Confusion Matrix for GridSearch Ridge
3209
791
910
3090
No Rain
Rain
Actual
No Rain
Rain
Predicted
3000
2500
2000
1500
1000


Confusion Matrix for RandomizedSearch Lasso
3215
785
915
3085
No Rain
Rain
Actual
No Rain
Rain
Predicted
3000
2500
2000
1500
1000


Confusion Matrix for RandomizedSearch Ridge
3209
791
910
3090
No Rain
Rain
Actual
No Rain
Rain
Predicted
3000
2500
2000
1500
1000





For all models, the True Negatives (TN) (i.e. predictions for "No Rain") are considerably higher than the False Positives (FP) (predicting "Rain" when it's actually "No Rain"), which means the models are performing better when predicting "No Rain". The False Negatives (FN) (predicting "No Rain" when it actually rained) are relatively low compared to True Positives (TP), indicating that the models aren't missing too many instances of rain. The results from BayesianSearchLasso and BayesianSearchRidge show slightly higher false positives and false negatives compared to the results from GridSearch and RandomizedSearch, which may suggest that the Bayesian search resulted in overfitting.

This current project shows some of the limitations of the models chosen and sets it up for the next one where I'll try using ensemble modelling to check whether the results get any better.