Exercise 2.4: Evaluating Hyperparameters

By Kristina Noriega

Purpose

The purpose of this exercise was to evaluate the impact of hyperparameter tuning on model performance using the ClimateWins dataset. Two modeling approaches were examined: a Random Forest classifier and a Convolutional Neural Network (CNN). Hyperparameter optimization was applied to both models, and performance was evaluated using appropriate metrics, including accuracy, ROC–AUC, and confusion matrices.

Random Forest Hyperparameter Optimization

Hyperparameter Tuning Process

A Random Forest classifier was optimized using cross-validated hyperparameter search. The following hyperparameters were tuned:

- n_estimators
- max_depth
- max_features
- min_samples_split
- min_samples_leaf
- criterion

Separate optimizations were performed for:

1. All stations (decade-level data)
2. Single station (BASEL, full timeline)

The best hyperparameter combinations were selected based on cross-validated accuracy.

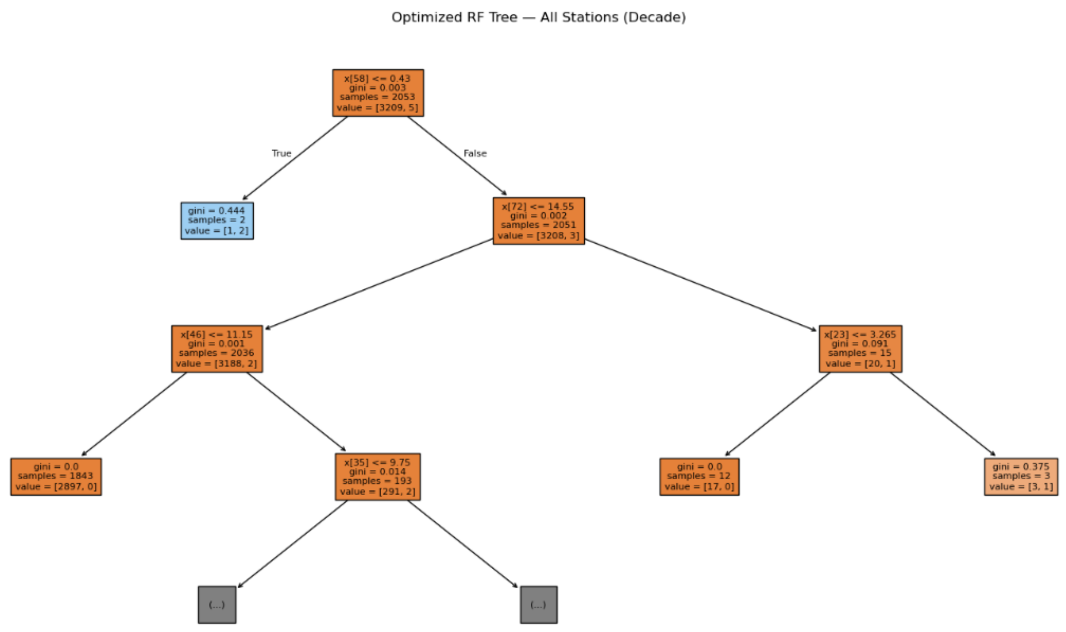


Figure 1. Optimized Random Forest tree for the all-stations decade-level model.

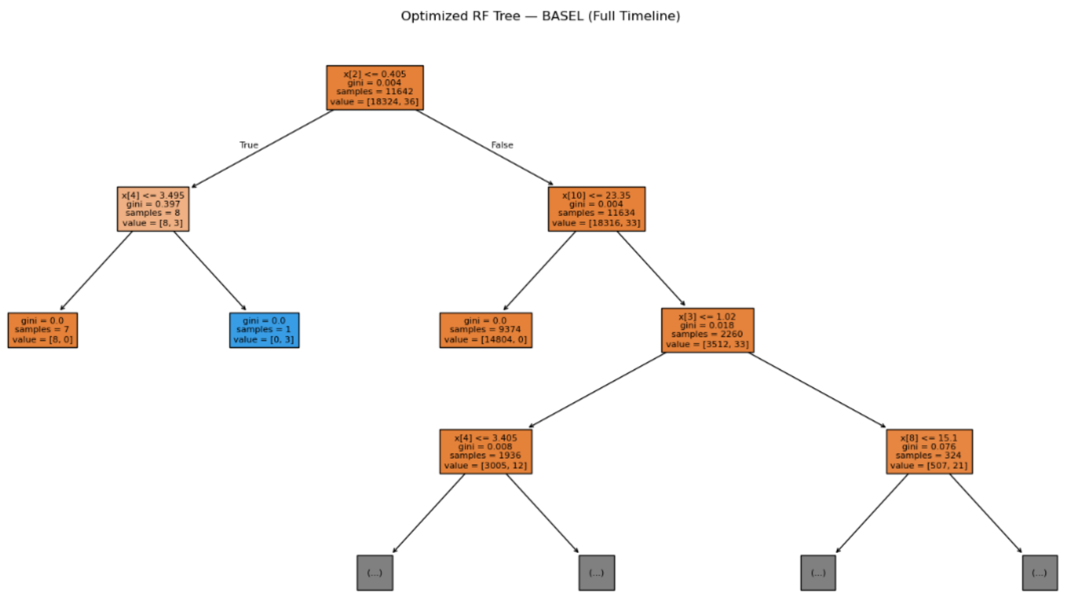


Figure 2. Optimized Random Forest tree for the BASEL full-timeline model.

Random Forest Results

After tuning, both Random Forest models showed strong predictive performance on the test set. The optimized hyperparameters improved generalization compared to default settings, particularly by controlling tree depth and minimum sample splits, which reduced overfitting.

The optimized models demonstrated:

- High test accuracy
- Stable performance across stations
- Improved interpretability through inspection of individual decision trees

These results indicate that hyperparameter tuning meaningfully improved the Random Forest models’ ability to predict pleasant weather conditions.

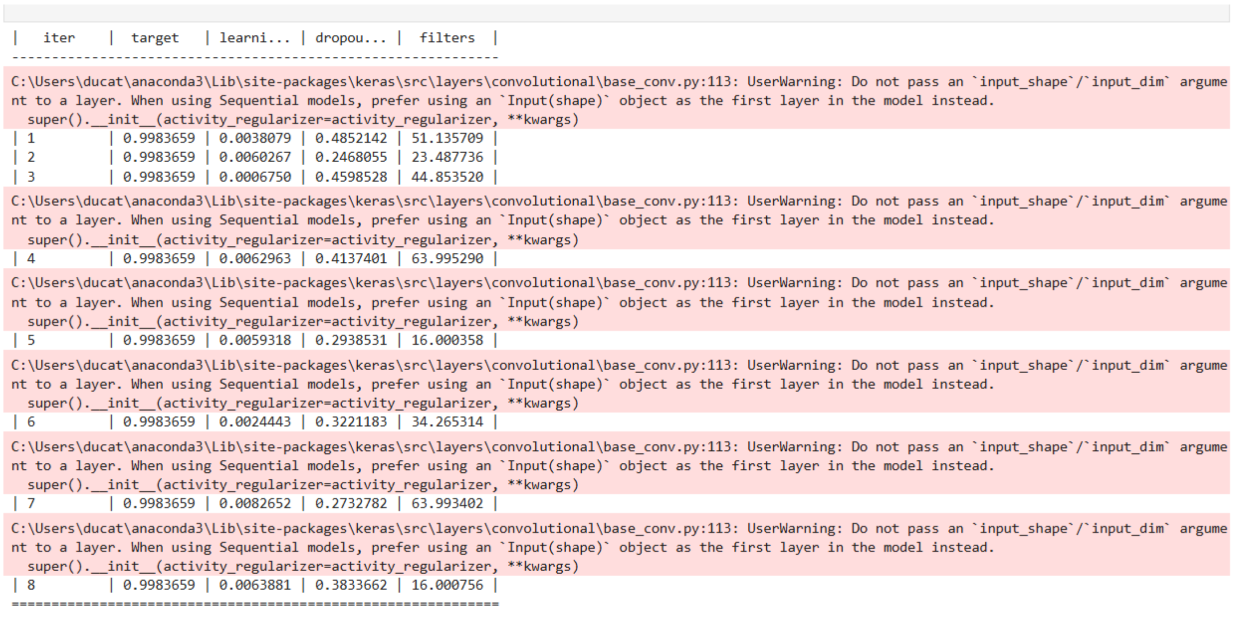
CNN Hyperparameter Optimization

Hyperparameter Tuning Process

A Convolutional Neural Network (CNN) was optimized using Bayesian Optimization, which efficiently explored the hyperparameter space by balancing exploration and exploitation. The following parameters were tuned:

- Learning rate
- Dropout rate
- Number of convolutional filters

The Bayesian optimization process identified the optimal hyperparameter combination based on validation accuracy.



```
|   iter    |  target   | learni... | dropou... |  filters  |
-------------------------------------------------------------
C:\Users\ducat\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argume
nt to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
| 1         | 0.9983659 | 0.0038079 | 0.4852142 | 51.135709 |
| 2         | 0.9983659 | 0.0060267 | 0.2468055 | 23.487736 |
| 3         | 0.9983659 | 0.0006750 | 0.4598528 | 44.853520 |
C:\Users\ducat\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argume
nt to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
| 4         | 0.9983659 | 0.0062963 | 0.4137401 | 63.995290 |
C:\Users\ducat\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argume
nt to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
| 5         | 0.9983659 | 0.0059318 | 0.2938531 | 16.000358 |
C:\Users\ducat\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argume
nt to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
| 6         | 0.9983659 | 0.0024443 | 0.3221183 | 34.265314 |
C:\Users\ducat\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argume
nt to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
| 7         | 0.9983659 | 0.0082652 | 0.2732782 | 63.993402 |
C:\Users\ducat\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argume
nt to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
| 8         | 0.9983659 | 0.0063881 | 0.3833662 | 16.000756 |
=============================================================
```

Figure 3. Bayesian optimization output showing CNN hyperparameter search and best parameter selection.

CNN Performance Evaluation

The final CNN model achieved a test accuracy of approximately 99.8% and a ROC–AUC score of 0.98, indicating excellent discriminative ability.

However, the dataset is highly imbalanced, with very few positive ("pleasant") days. As a result, raw accuracy alone was insufficient to fully assess model performance. To address this, ROC–AUC and confusion matrices were used.

Initially, a standard classification threshold (0.5) resulted in the model predicting only the majority class. After lowering the classification threshold to 0.2, the model successfully identified positive cases while maintaining strong overall performance.

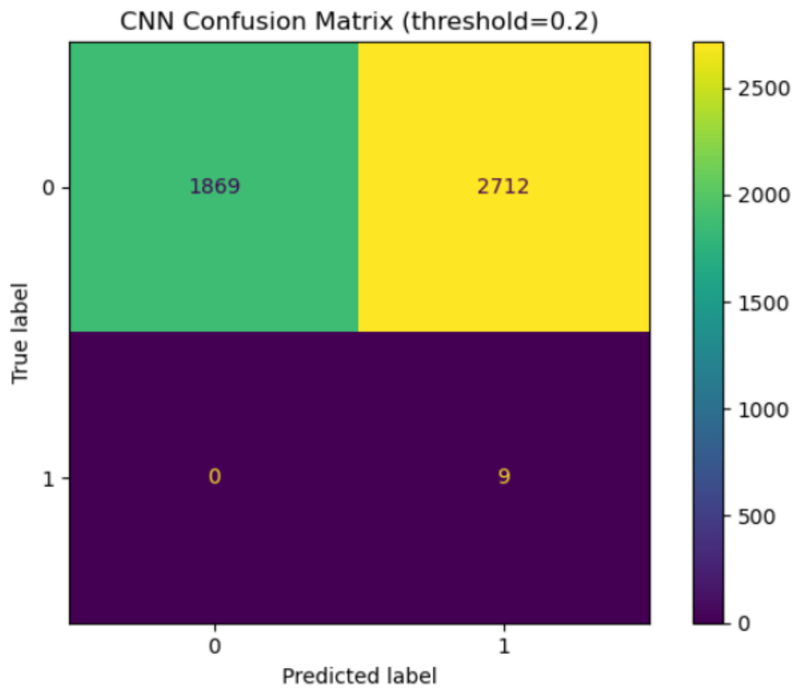


Figure 4. CNN confusion matrix using a classification threshold of 0.2.

Interpretation of CNN Results

The CNN demonstrated strong predictive capability when evaluated with appropriate metrics. While class weighting reduced overall accuracy, it highlighted the trade-off between sensitivity to rare events and general accuracy. The ROC–AUC score confirmed that the model effectively ranked positive cases higher than negative ones, even in the presence of severe class imbalance.

These results emphasize the importance of selecting evaluation metrics that align with the data characteristics and modeling goals.

Conclusion

Hyperparameter tuning significantly improved model performance for both Random Forest and CNN approaches. The Random Forest models benefited from controlled complexity and improved generalization, while Bayesian optimization enabled efficient CNN tuning. Evaluation using ROC–AUC and confusion matrices ensured that model performance was assessed appropriately for an imbalanced classification problem.

Overall, the results demonstrate that careful hyperparameter optimization and metric selection are essential for building reliable machine learning models in real-world datasets such as ClimateWins.