EcoShift: Analyzing Climate Changes and Their Roots

- Lokesh Kumar Rudhramurthi, Ananya Anil Singh, Kritya Shree Sivasakthi

Introduction

Climate change is a critical global concern, and analyzing climate data is crucial to understanding its causes, trends, and environmental impacts. This project explores weather data from the World Weather Repository, focusing specifically on capital cities.

Predicting air quality index (AQI) is particularly important due to its direct impact on human health. AQI levels are categorized into six classes: Good, Moderate, Unhealthy for Sensitive Groups, Unhealthy, Very Unhealthy, and Hazardous. Understanding these classifications and predicting AQI fluctuations can help individuals take precautions and protect their health.

In the following sections, we will delve into data preprocessing, model implementation, and results analysis. Our aim is to develop effective models that predict AQI accurately, ultimately contributing to public health awareness and improved environmental decision-making.

Data Preprocessing

The World Weather Repository dataset underwent several important steps to prepare it for analysis. Weather condition labels were standardized, irrelevant columns were removed, temperature units were converted to Celsius, and visibility units were converted to kilometers. These steps ensured data uniformity and facilitated a more insightful analysis of climate patterns and environmental impacts.

Model Implementation

Four models were implemented to predict the air quality index variable "air_quality_us-epa-index". Each of the five models were implemented three ways. First, the model was run without any parameter changes. Next, grid search was performed to hyper tune the parameters and find the optimal parameters for the model. Finally, to address the class imbalance (as shown in fig. 1), the data is oversampled and models are rerun. The models used in the project are as follows:

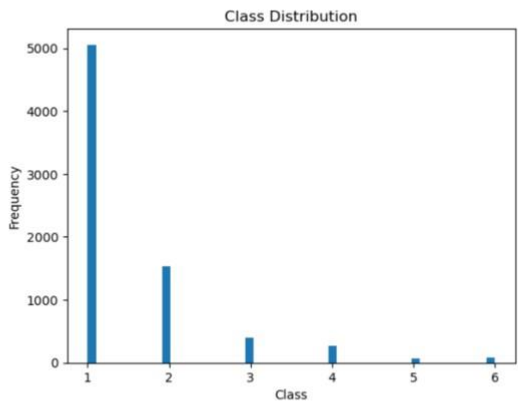
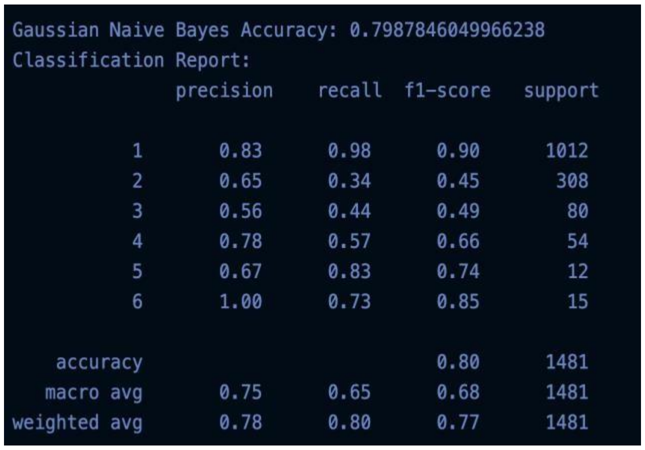


Figure 1. Distribution of records for each class

Gaussian Naïve Bayes

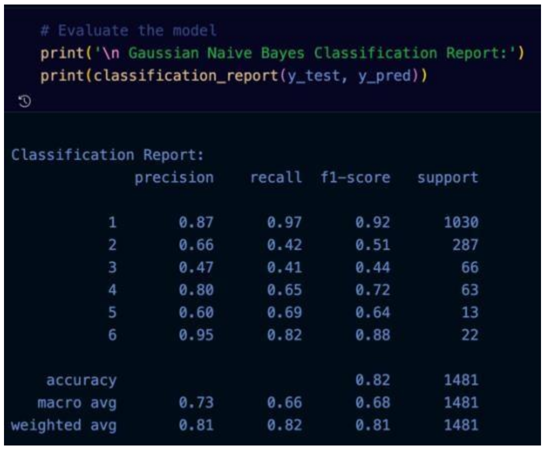
Hyper-parameter Tuned: 'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]

Best parameter: {'var_smoothing': 1e-09}



```
Gaussian Naive Bayes Accuracy: 0.7987846049966238
Classification Report:
              precision    recall  f1-score   support

           1       0.83      0.98      0.90      1012
           2       0.65      0.34      0.45       308
           3       0.56      0.44      0.49        80
           4       0.78      0.57      0.66        54
           5       0.67      0.83      0.74        12
           6       1.00      0.73      0.85        15

    accuracy                           0.80      1481
   macro avg       0.75      0.65      0.68      1481
weighted avg       0.78      0.80      0.77      1481
```

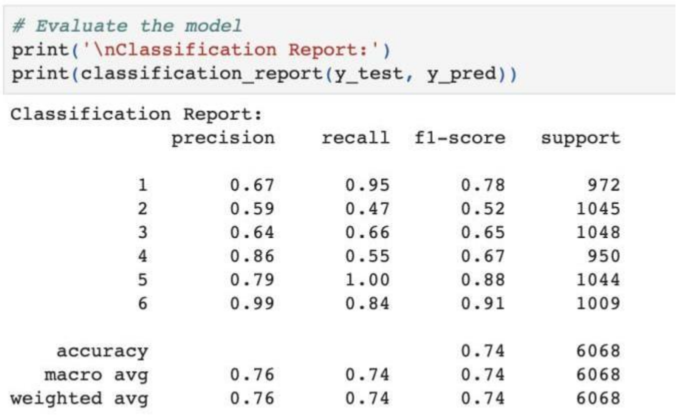
Figure 2. Gaussian Naive Bayes results



```
    # Evaluate the model
    print('\n Gaussian Naive Bayes Classification Report:')
    print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           1       0.87      0.97      0.92      1030
           2       0.66      0.42      0.51       287
           3       0.47      0.41      0.44        66
           4       0.80      0.65      0.72        63
           5       0.60      0.69      0.64        13
           6       0.95      0.82      0.88        22

    accuracy                           0.82      1481
   macro avg       0.73      0.66      0.68      1481
weighted avg       0.81      0.82      0.81      1481
```

Figure 3. Gaussian NB results with hyperparameter tuning



```
# Evaluate the model
print('\nClassification Report:')
print(classification_report(y_test, y_pred))
```

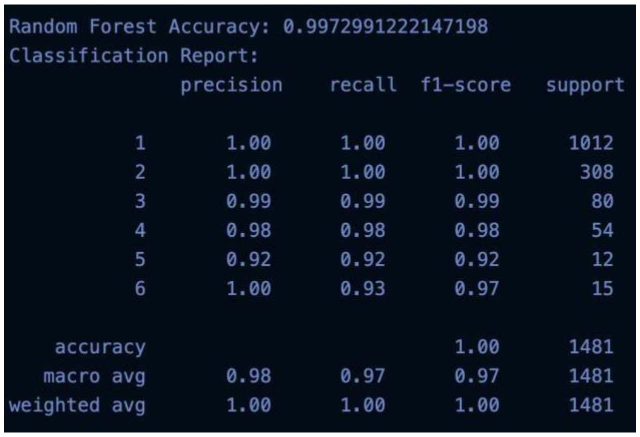
```
Classification Report:
              precision    recall  f1-score   support

           1       0.67      0.95      0.78       972
           2       0.59      0.47      0.52      1045
           3       0.64      0.66      0.65      1048
           4       0.86      0.55      0.67       950
           5       0.79      1.00      0.88      1044
           6       0.99      0.84      0.91      1009

    accuracy                           0.74      6068
   macro avg       0.76      0.74      0.74      6068
weighted avg       0.76      0.74      0.74      6068
```

Figure 4. Gaussian NB results after oversampling

Random Forest

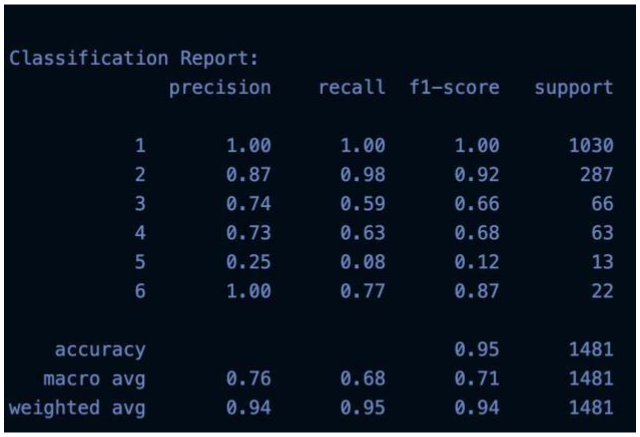
Hyperparameters Tuned: 'n_estimators': [50, 100, 200], 'max_depth': [None, 10, 20], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4]

Best parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 200}



```
Random Forest Accuracy: 0.9972991222147198
Classification Report:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      1012
           2       1.00      1.00      1.00       308
           3       0.99      0.99      0.99        80
           4       0.98      0.98      0.98        54
           5       0.92      0.92      0.92        12
           6       1.00      0.93      0.97        15

    accuracy                           1.00      1481
   macro avg       0.98      0.97      0.97      1481
weighted avg       1.00      1.00      1.00      1481
```

Figure 5. Random Forest results



```
Classification Report:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      1030
           2       0.87      0.98      0.92       287
           3       0.74      0.59      0.66        66
           4       0.73      0.63      0.68        63
           5       0.25      0.08      0.12        13
           6       1.00      0.77      0.87        22

    accuracy                           0.95      1481
   macro avg       0.76      0.68      0.71      1481
weighted avg       0.94      0.95      0.94      1481
```

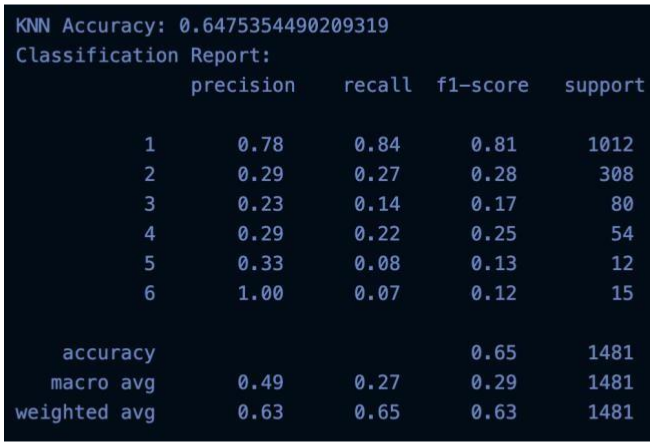
Figure 6. Random Forest with Hyperparameter Tuning

A challenge we faced during the implementation of the algorithm after oversampling with hyperparameter tuning was time complexity. Since the algorithm ran for over an hour and still did not give results, we have omitted the algorithm.

K-Nearest Neighbor

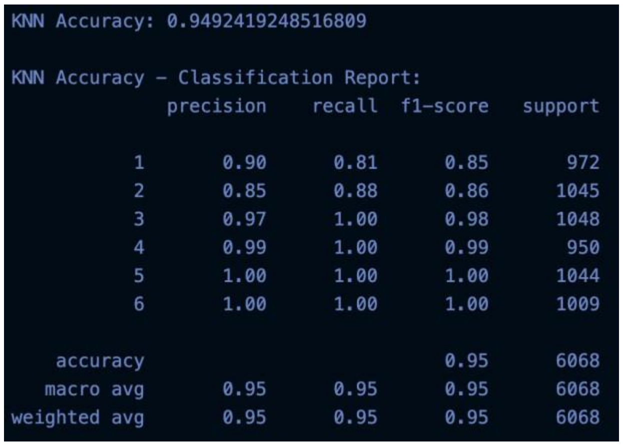
Hyperparameter tuned: 'n_neighbors': [3, 5, 7, 9, 11]

Best parameter: 3



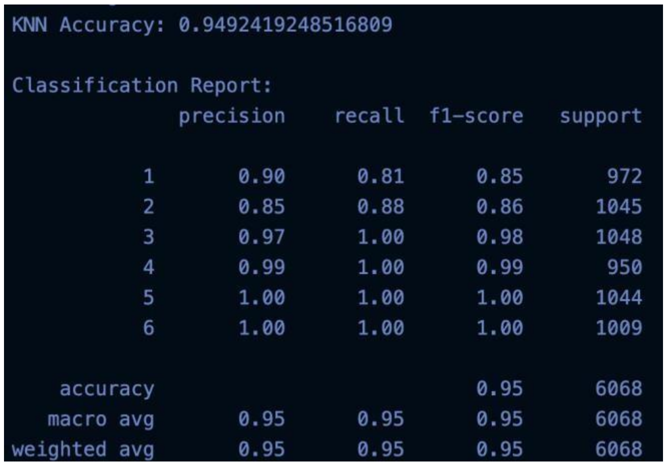
```
KNN Accuracy: 0.6475354490209319
Classification Report:
              precision    recall  f1-score   support

           1       0.78      0.84      0.81      1012
           2       0.29      0.27      0.28       308
           3       0.23      0.14      0.17        80
           4       0.29      0.22      0.25        54
           5       0.33      0.08      0.13        12
           6       1.00      0.07      0.12        15

    accuracy                           0.65      1481
   macro avg       0.49      0.27      0.29      1481
weighted avg       0.63      0.65      0.63      1481
```

Figure 7. KNN results



```
KNN Accuracy: 0.9492419248516809

KNN Accuracy - Classification Report:
              precision    recall  f1-score   support

           1       0.90      0.81      0.85       972
           2       0.85      0.88      0.86      1045
           3       0.97      1.00      0.98      1048
           4       0.99      1.00      0.99       950
           5       1.00      1.00      1.00      1044
           6       1.00      1.00      1.00      1009

    accuracy                           0.95      6068
   macro avg       0.95      0.95      0.95      6068
weighted avg       0.95      0.95      0.95      6068
```

Figure 8. KNN with hyperparameter tuning



```
KNN Accuracy: 0.9492419248516809

Classification Report:
              precision    recall  f1-score   support

           1       0.90      0.81      0.85       972
           2       0.85      0.88      0.86      1045
           3       0.97      1.00      0.98      1048
           4       0.99      1.00      0.99       950
           5       1.00      1.00      1.00      1044
           6       1.00      1.00      1.00      1009

    accuracy                           0.95      6068
   macro avg       0.95      0.95      0.95      6068
weighted avg       0.95      0.95      0.95      6068
```

Figure 9. KNN after oversampling

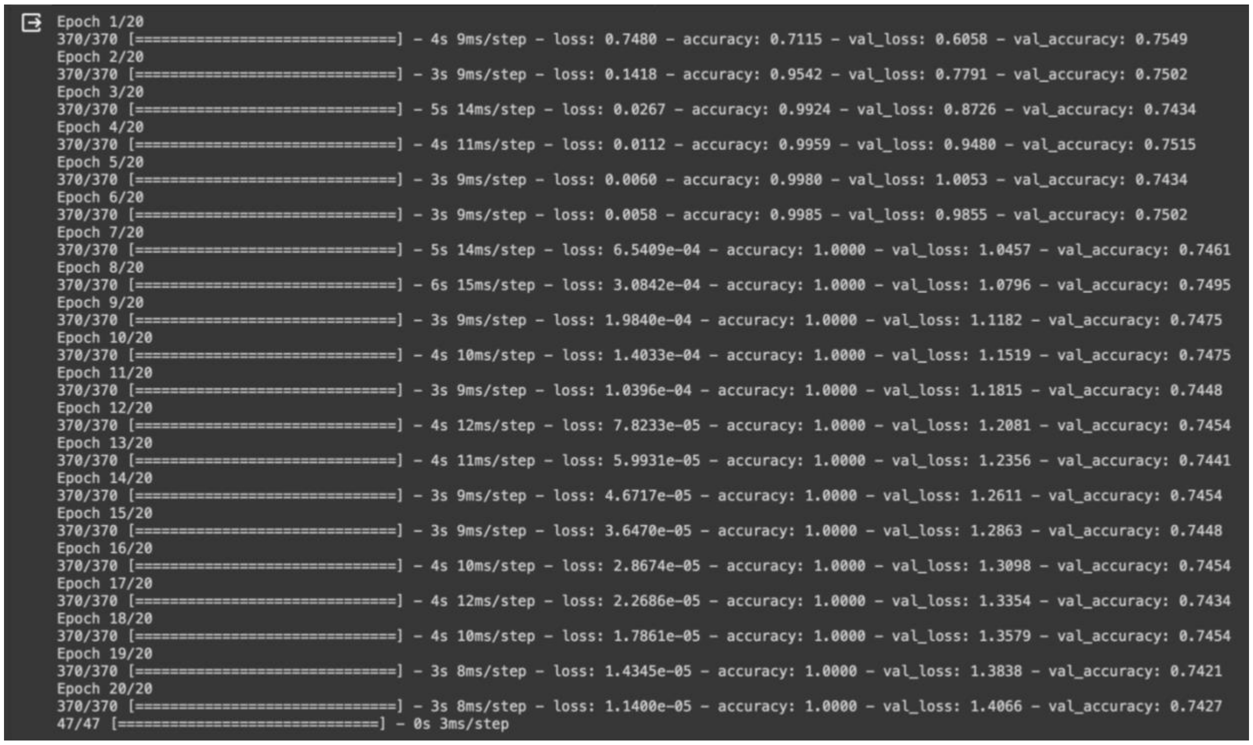
Artificial Neural Network

A simple base model was created using Keras API within TensorFlow. The model uses 3 layers:

Input: 128 neurons, ReLU activation

Hidden: 64 neurons, ReLU activation

Output: Softmax activation for multi-class classification



```
Epoch 1/20
370/370 [==============================] - 4s 9ms/step - loss: 0.7480 - accuracy: 0.7115 - val_loss: 0.6058 - val_accuracy: 0.7549
Epoch 2/20
370/370 [==============================] - 3s 9ms/step - loss: 0.1418 - accuracy: 0.9542 - val_loss: 0.7791 - val_accuracy: 0.7502
Epoch 3/20
370/370 [==============================] - 5s 14ms/step - loss: 0.0267 - accuracy: 0.9924 - val_loss: 0.8726 - val_accuracy: 0.7434
Epoch 4/20
370/370 [==============================] - 4s 11ms/step - loss: 0.0112 - accuracy: 0.9959 - val_loss: 0.9480 - val_accuracy: 0.7515
Epoch 5/20
370/370 [==============================] - 3s 9ms/step - loss: 0.0060 - accuracy: 0.9980 - val_loss: 1.0053 - val_accuracy: 0.7434
Epoch 6/20
370/370 [==============================] - 3s 9ms/step - loss: 0.0058 - accuracy: 0.9985 - val_loss: 0.9855 - val_accuracy: 0.7502
Epoch 7/20
370/370 [==============================] - 5s 14ms/step - loss: 6.5409e-04 - accuracy: 1.0000 - val_loss: 1.0457 - val_accuracy: 0.7461
Epoch 8/20
370/370 [==============================] - 6s 15ms/step - loss: 3.0842e-04 - accuracy: 1.0000 - val_loss: 1.0796 - val_accuracy: 0.7495
Epoch 9/20
370/370 [==============================] - 3s 9ms/step - loss: 1.9840e-04 - accuracy: 1.0000 - val_loss: 1.1182 - val_accuracy: 0.7475
Epoch 10/20
370/370 [==============================] - 4s 10ms/step - loss: 1.4033e-04 - accuracy: 1.0000 - val_loss: 1.1519 - val_accuracy: 0.7475
Epoch 11/20
370/370 [==============================] - 3s 9ms/step - loss: 1.0396e-04 - accuracy: 1.0000 - val_loss: 1.1815 - val_accuracy: 0.7448
Epoch 12/20
370/370 [==============================] - 4s 12ms/step - loss: 7.8233e-05 - accuracy: 1.0000 - val_loss: 1.2081 - val_accuracy: 0.7454
Epoch 13/20
370/370 [==============================] - 4s 11ms/step - loss: 5.9931e-05 - accuracy: 1.0000 - val_loss: 1.2356 - val_accuracy: 0.7441
Epoch 14/20
370/370 [==============================] - 3s 9ms/step - loss: 4.6717e-05 - accuracy: 1.0000 - val_loss: 1.2611 - val_accuracy: 0.7454
Epoch 15/20
370/370 [==============================] - 3s 9ms/step - loss: 3.6470e-05 - accuracy: 1.0000 - val_loss: 1.2863 - val_accuracy: 0.7448
Epoch 16/20
370/370 [==============================] - 4s 10ms/step - loss: 2.8674e-05 - accuracy: 1.0000 - val_loss: 1.3098 - val_accuracy: 0.7454
Epoch 17/20
370/370 [==============================] - 4s 12ms/step - loss: 2.2686e-05 - accuracy: 1.0000 - val_loss: 1.3354 - val_accuracy: 0.7434
Epoch 18/20
370/370 [==============================] - 4s 10ms/step - loss: 1.7861e-05 - accuracy: 1.0000 - val_loss: 1.3579 - val_accuracy: 0.7454
Epoch 19/20
370/370 [==============================] - 3s 8ms/step - loss: 1.4345e-05 - accuracy: 1.0000 - val_loss: 1.3838 - val_accuracy: 0.7421
Epoch 20/20
370/370 [==============================] - 3s 8ms/step - loss: 1.1400e-05 - accuracy: 1.0000 - val_loss: 1.4066 - val_accuracy: 0.7427
47/47 [==============================] - 0s 3ms/step
```

Figure 10. Results of ANN model

Results and Discussion

By comparing the results of all the models, the best performing model is Random Forest with hyperparameter tuning:

Accuracy: 95%,

Macro Avg Precision: 76%

Weighted Avg Precision: 94%

Macro Avg Recall: 68%

Weighted Avg Recall: 95%

Macro Avg F1-score: 71%

Weighted Avg F1-score: 94%

The Gaussian Naive Bayes (GNB) model effective in predicting Air Quality Index (AQI) levels, with particularly good performance on the "Good" (1) and "Hazardous" (6) classes. However, themodel's recall scores for the "Unhealthy for Sensitive Groups" and "Unhealthy" classes are relatively low. Additionally, the model's precision is consistently higher than its recall. This means that the model is more likely to correctly predict a class than to miss all of the instances ofthat class.

Hyperparameter tuning and oversampling significantly improved the performance of the **KNN model**, with an accuracy increase from 0.6475 to 0.95. This means that the model is able to correctly predict the AQI levels with a high accuracy.

From the training loss and accuracy of the **Artificial Neural Network model** we see that the model was able to learn the training data well. The training loss decreased from 0.7488 to 1.1400e-05 over the course of 20 epochs, and the training accuracy increased from 0.7115 to 1.0000. Furthermore, the validation loss and accuracy show that the model was able to generalizewell to unseen data. The validation loss increased slightly from 0.6058 to 1.4066 over the course of 20 epochs, but the validation accuracy remained high at 0.7427. This suggests that the model is not overfitting the training data.

In **Random Forest**, the model is able to identify and classify all AQI classes accurately. Overall, the results suggest that the random forest model with hyperparameter tuning is a very effective model for AQI prediction. This is likely due to the fact that random forests are able to learn complex patterns in the data and are robust to overfitting. However, the model struggled to predict the “Very Unhealthy” (5) class but low precision and recall.