

# Global Weather Pattern Prediction for Disaster Preparedness

Jonas Thalmeier, Ana Martinez

December 29, 2024

## 1 Introduction

Effective identification and prediction of heatwaves are critical for public health, agriculture, and energy management. To accomplish this, a dataset containing atmospheric measurements was downloaded from the Copernicus Climate Data Store (CDS)<sup>1</sup>. The dataset in use is ERA5, which provides hourly atmospheric reanalysis data with a high temporal and spatial resolution. As part of this project, steps were taken to extract spatially and temporally restricted subsets of ERA5 data and to apply percentile-based definitions of heatwaves.

This report details the intermediate steps undertaken: data acquisition, preprocessing, feature labeling, and initial approaches to building machine learning models for predicting future heat events. The approach followed best practices in data handling and model preparation, ensuring that the methodology can be refined and improved in subsequent phases.

## 2 Data Acquisition and Preprocessing

### 2.1 Dataset Selection and Constraints

The ERA5 dataset was accessed through the Climate Data Store after creating an account on the platform. Due to storage and download constraints, the spatiotemporal coverage had to be limited. Therefore, only data covering the territory of Spain were selected. Furthermore, since full hourly coverage would be voluminous, only two daily time points (00:00 and 12:00 UTC) were chosen.

For the first experiments a dataset with the following features was generated:

- 2m temperature (t2m)
- 2m dewpoint temperature (d2m)
- Mean sea level pressure (msl)
- Total cloud cover (tcc)
- Volumetric soil water layer 1 (swvl1)
- 10m u-component of wind (u10)
- 10m v-component of wind (v10)

The dataset encompassed locations from 36°south to 44°north and 10°west to 4°east with a spatial grid width of 1°as well as all months of the year, with the intention of capturing a wide range of meteorological conditions. It was later observed that including all months, even those unlikely to produce heatwaves, was not the most efficient approach. Therefore a new dataset was generated. It only includes the month April to September. Since less days per year are included in the dataset, it was possible to lower the spatial resolution to a grid width of 0.25°. Furthermore the following features were added:

- Surface Pressure (sp)
- Skin Temperature (skt)
- Boundary Layer Height (blh)
- Land-Sea Mask (lsm)
- Total Column Water Vapor (tcwv)

---

<sup>1</sup><https://cds.climate.copernicus.eu/>

## 2.2 Temporal and Spatial Restructuring

After downloading the data in GRIB format, standard Python libraries such as `xarray` and `pygrib` were used for reading and handling the dataset. The data were initially organized by time and location. To simplify the learning task, measurements recorded at 00:00 and 12:00 for the same day and location were combined, presenting them as features side-by-side within a single row. This reformatting streamlined the input for the machine learning model, ensuring that each daily record included both early morning and midday conditions.

## 3 Definition of Hot Days and Heatwaves

The Spanish meteorological service defines a heatwave as a sequence of at least three consecutive days during which the maximum daily temperature at a given location exceeds the 95th percentile of the daily maximum temperature distribution for July and August at that location. To implement this criterion, the temperatures at 12:00 for July and August were extracted from the ERA5 dataset. The 95th percentile threshold was then computed for each grid point (latitude-longitude pair).

Subsequently, any day whose 12:00 temperature exceeded the computed local 95th percentile threshold was labeled as a “hot day.” Rolling computations were carried out to identify sequences of three or more consecutive hot days (i.e., potential heatwave events). These computations confirmed the occurrence of multi-day hot periods in certain locations and provided an initial assessment of the frequency and distribution of heatwave events.

## 4 Labeling and Feature Engineering

To train a predictive model, labels indicating whether a heatwave would occur in the near future were needed. For each 30-day period (rolling window of consecutive days), a binary label was assigned: 1 if at least 3 hot days would occur in the following 7 days, and 0 otherwise. This labeling strategy provided a supervised learning setup, enabling the prediction of imminent heatwave conditions based on recent weather patterns. At this stage, a challenge arose: only about 2.09% of the labels were positive (indicating a future heatwave), resulting in a highly imbalanced dataset.

To facilitate machine learning, the data from all consecutive 30-day periods were stacked so that each record included the full sequence of daily measurements for those 30 days. However, this approach constrained the model to utilize data from a single location exclusively for forecasting that specific location. While this is not the standard approach for weather prediction, it was adopted to simplify data processing to a manageable level. Additionally, the project aimed to evaluate how accurately weather could be predicted without incorporating data from surrounding regions.

To address memory overflow issues that arose during data processing, a chunk-based approach was implemented to save intermediate results to disk. Instead of retaining the entire dataset in memory, the data for each year was processed incrementally. This involved extracting rolling features and aligning them with the corresponding labels for each year separately. The features and labels were then flattened and concatenated with additional attributes, such as latitude, longitude, and percentile thresholds, before being saved as PyTorch tensors to disk. This strategy enabled the handling of large datasets by reducing the memory footprint and allowed for efficient data loading during training, as individual yearly chunks could be accessed and processed independently without loading the entire dataset into memory.

## 5 Preliminary Modeling Approaches

Initial attempts to predict heatwaves involved testing several machine learning models:

1. A simple neural network (NN) with fully connected layers was trained on the transformed dataset.
2. A deeper neural network architecture, employing additional layers and dropout, was also tested.
3. A random forest classifier
4. A eXtreme Gradient Boosting classifier
5. A logistic regression model from torch
6. A logistic regression model from sklearn

## 5.1 Neural Networks

The initial trials were conducted using neural networks due to their versatility and ability to model complex relationships in data. Given their wide applicability across various domains, they were anticipated to deliver acceptable initial results for this task. For these trials, the initial smaller dataset was used. The first network architecture is shown below:

```
1 class HotDayPredictor(nn.Module):
2     def __init__(self, input_size):
3         super(HotDayPredictor, self).__init__()
4         self.fc = nn.Sequential(
5             nn.Linear(input_size, 128),
6             nn.ReLU(),
7             nn.Linear(128, 64),
8             nn.ReLU(),
9             nn.Linear(64, 1),
10            nn.Sigmoid()
11        )
12    def forward(self, x):
13        return self.fc(x)
```

Normalization was applied to the inputs using the global mean and standard deviation calculated from the training set. Despite these steps, the results were disappointing. The training loss decreased, but the validation loss stagnated or increased, indicating significant overfitting. As depicted in Figure 1, the loss for the validation set was much higher than that of the training set. To address the overfitting, several



Figure 1: Evolution of training and validation loss

iterations were undertaken to improve the network's generalization ability:

**Iteration 1: Adding Dropouts and Weighted Loss** Dropout layers were introduced to the architecture to mitigate overfitting by preventing co-adaptation of neurons. The modified architecture included dropout rates of 30%:

```
1 self.fc = nn.Sequential(
2     nn.Linear(input_size, 128),
3     nn.ReLU(),
4     nn.Dropout(0.3), # Add dropout with a rate of 30%
5     nn.Linear(128, 64),
6     nn.ReLU(),
7     nn.Dropout(0.3), # Add dropout
8     nn.Linear(64, 1)
```

9 | )

Additionally, a weighted loss function was implemented to address class imbalance:

```
1 global_pos_weight = (global_total_count - global_positive_count) /  
    global_positive_count  
2 loss_fn = nn.BCEWithLogitsLoss(pos_weight=torch.tensor(global_pos_weight, dtype  
    =torch.float32))
```

The learning rate was also reduced by a factor of 10 (to 0.0001). Despite these changes, overfitting persisted:

- Training Loss: 0.311259
- Validation Loss: 20.156125

**Iteration 2: Adding Geospatial Features** Longitude, latitude, and the 95th percentile temperature were added as additional features to each record. While this increased the dimensionality of the data, overfitting remained an issue, with a continued large discrepancy between training and validation loss. Validation loss increased further, indicating no improvement:

- Training Loss: 0.299057
- Validation Loss: 24.856614

**Iteration 3: Expanded Dataset and Temporal Filtering** An expanded dataset was generated, including additional variables as described in section 2.1. Despite these enhancements, overfitting persisted:

- Training Loss: 0.282310
- Validation Loss: 22.825080

The dataset was limited to land masses using the Land-Sea Mask (lsm) feature included in the dataset. This approach was based on the expectation that weather behavior differs significantly between land and sea, and thus predictions might be easier when focusing exclusively on land areas. However, this restriction significantly reduced the data diversity, leading to an even stronger overfitting effect:

- Training Loss: 0.161386
- Validation Loss: 31.200668

**Evaluation Metrics** The following evaluation metrics illustrate the poor generalization of the model:

- Accuracy: 0.8951
- Precision: 0.1166
- Recall: 0.0436
- F1 Score: 0.0635

Despite various attempts to mitigate overfitting, the neural network failed to generalize effectively, necessitating a shift to alternative modeling approaches. As shown in Figure 2, the model's performance remains unsatisfactory after all iterations. The training loss continues to decrease steadily, while the validation loss either stagnates or increases, indicating persistent overfitting and the inability of the model to generalize to unseen data.

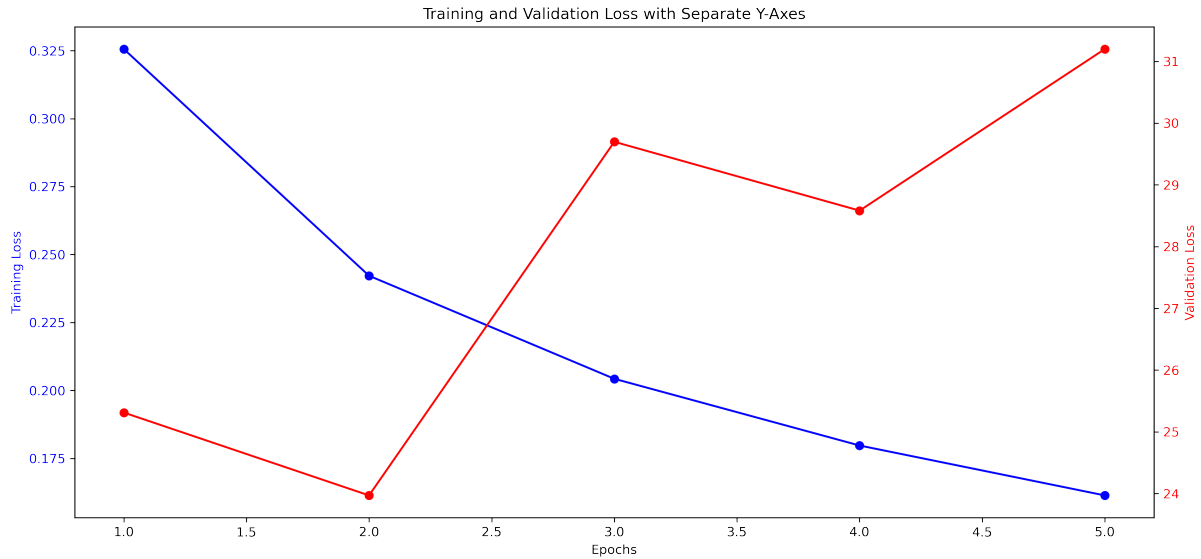


Figure 2: Evolution of training and validation loss after 3rd Iteration of NN model

## 5.2 Random Forest

Random forests were employed due to their robustness and ability to handle high-dimensional data without extensive preprocessing. They also naturally account for feature importance, which can provide insights into the data. For this experiment, the entire training set was used, and hyperparameters such as the number of estimators and maximum tree depth were tuned. The model was trained using the following parameters:

```

1 rf_model = RandomForestClassifier(
2     n_estimators=500,
3     max_depth=20,
4     random_state=42,
5     class_weight="balanced",
6     n_jobs=-1)

```

Data preprocessing included normalizing the features using the global mean and standard deviation calculated from the training set. The random forest achieved the following results on the test set:

- Accuracy: 0.9185
- Precision: 0.2318
- Recall: 0.0347
- F1 Score: 0.0604

While the accuracy was high, the low precision and recall indicated that the model struggled to detect positive samples. The imbalanced dataset likely contributed to these shortcomings, despite applying balanced class weights.

## 5.3 eXtreme Gradient Boosting Classifier

The eXtreme Gradient Boosting classifier was chosen as it often outperforms other tree-based models, especially when dealing with structured data. Similar to the random forest, it was trained on the full dataset with hyperparameters tuned for optimal performance:

```

1 xgb_model = XGBClassifier(
2     n_estimators=500,
3     max_depth=10,
4     learning_rate=0.1,
5     scale_pos_weight=scale_pos_weight,

```

```

6     random_state=42,
7     n_jobs = -1
8 )

```

The features were normalized in the same way as for the random forest. The XGBoost classifier provided the following evaluation metrics:

- Accuracy: 0.9132
- Precision: 0.1987
- Recall: 0.0478
- F1 Score: 0.0776

Compared to the random forest, XGBoost demonstrated slightly improved precision and recall. However, the metrics were still far from acceptable, suggesting the limitations of tree-based models for this specific task.

## 5.4 Logistic Regression with PyTorch

A logistic regression model was implemented in PyTorch to evaluate the performance of a simple linear classifier. The model was initialized as follows:

```

1 class LogisticRegressionModel(nn.Module):
2     def __init__(self, input_size):
3         super(LogisticRegressionModel, self).__init__()
4         self.linear = nn.Linear(input_size, 1)
5         self.sigmoid = nn.Sigmoid()
6     def forward(self, x):
7         return self.sigmoid(self.linear(x))

```

The loss function was defined as binary cross-entropy with a positive class weight to address class imbalance:

```

1 criterion = nn.BCEWithLogitsLoss(pos_weight=torch.tensor([61.63], dtype=torch.
    float32))

```

Despite applying normalization, weighted loss, and regularization, the logistic regression model still struggled with overfitting, as shown in Figure 3.

- Training Loss: 0.2671
- Validation Loss: 22.1748
- Accuracy: 0.8965
- Precision: 0.1223
- Recall: 0.0508
- F1 Score: 0.0721

## 5.5 Logistic Regression with Scikit-Learn

Logistic regression implemented using Scikit-Learn offered a computationally efficient alternative. The model was initialized with the saga solver to handle large datasets and incorporated balanced class weights to address class imbalance:

```

1 lr_model = LogisticRegression(
2     penalty="l2", # Regularization
3     C=1.0, # Regularization strength (1.0 is default)
4     solver="saga"
5     max_iter=500, # Increase iterations for convergence
6     random_state=42,
7     n_jobs=-1
8 )

```

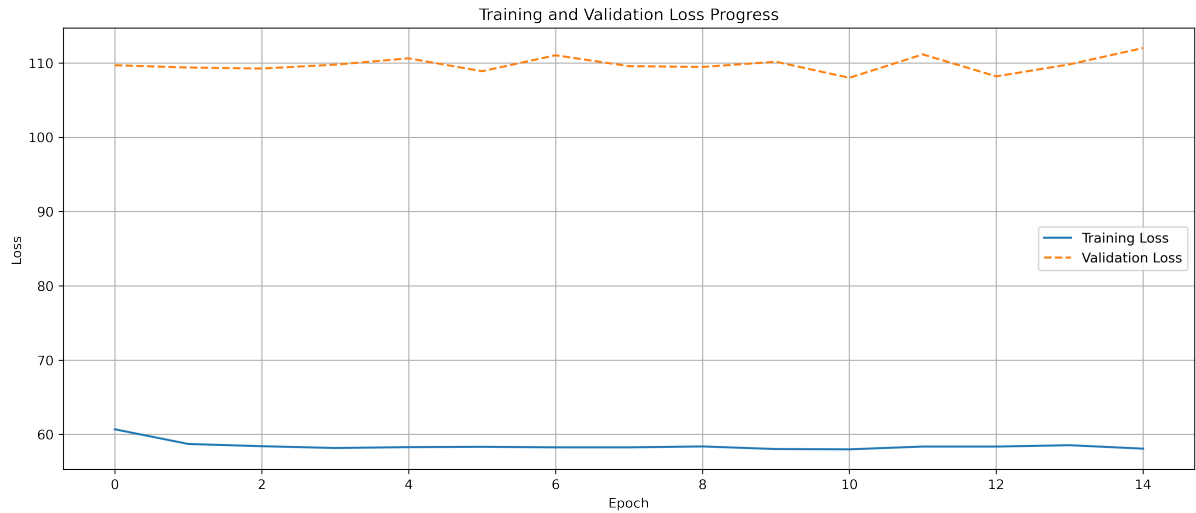


Figure 3: Training and validation loss for logistic regression (PyTorch)

Normalization was performed using the global mean and standard deviation. The Scikit-Learn implementation yielded the following results:

- Accuracy: 0.9023
- Precision: 0.1472
- Recall: 0.0625
- F1 Score: 0.0885

Although the Scikit-Learn implementation performed better than the PyTorch model in terms of F1 score, both approaches struggled with the highly imbalanced data, reaffirming the challenges of this task.

## 6 Allocation of tasks

- Setting up git: Jonas
- Generating Dataset: Jonas
- Data Preprocessing: Jonas
- Machine Learning, optimizing Neural Network: Both
- Writing Report: Both