Weather prediction ML final model report

Introduction

Weather predictions remain some of the toughest tasks within the world of Computer Science. Modern state-of-the-art models utilise supercomputers and large scale modelling to predict weather patterns. However, despite the huge resources invested in forecasting, the actual end result is mixed. While short-term forecasting is relatively good, trying to predict things such as rainfall beyond a couple of days takes a huge hit across all metrics and becomes extremely unreliable. The goal of this project is to explore various Machine Learning methods and attempt to reach a good weather prediction model within the constraints of a regular gaming laptop (i7 processor, 3070ti graphics card). The model will attempt to predict rainfall in the future for up to a week in advance in an Amazonian rainforest (one of the hardest environments to predict rainfall in)

Project results, overview and strategy

Starting dataset

The project utilizes a temporal era5 dataset. The starting point includes following columns: valid_time, u10, v10, d2m, t2m, sp, tp, latitude, longitude. The data contains hourly measurements from 01/2018 to 01/2025. This is a very limited number of features for a complex task, but that is mitigated through feature engineering later on.

Data processing and setup.

The developed model becomes complex very quickly. The data processing aspect of the code is started off by developing a grid. The model attempts to capture weather fronts across a 9-point grid. Each point is a single coordinate that contains the same data only differing by location. Data is cleanly loaded and joined into one dataframe. Features are then engineered, please see the code as there are too many to list, but a brief overview: lag and rolling features, cyclical-time features, synoptic gradient, wind, interaction and front features are all extracted and created from the dataset. The end result is X = 126! Since rainfall is the main goal, tp becomes the y. The dataset is then split accordingly for the hybrid needs, a split for the classifier and the regressor is done.

Brief overview of the model

The proposed model utilizes a hybrid architecture, containing a regressor and a classifier. The sequence of data fed to the model is in hours and the model makes daily predictions for 7 days. Please refer to the code and comments for full implementation.

The model architecture has 2 encoders, one for the short-term regressor input and one for the long-tern classifier input.
Core block contains a 2-step attention, Temporal and Spatial. Each of these attention blocks contains multiple layers, Multi-head Attention, feedforward networks and Residual connections. Positional embeddings are implemented for both temporal and spatial dimensions.
Two transformers.
Two decoders.

The classifier uses a custom loss function, weighted categorical cross entropy with the weights derived from the dataset based on imbalance.

Even though regression is the primary goal, a classifier implementation into the model helped raise its performance, especially at the mid-to-later days. Each part is fed its own sequence (create_sequence function). Varying times were tested for each sequence. Around 5 hours for the regressor and 96 for the classifier resulted in the best performance to time ratio. Regressor performed best at 5 hours. Classifier was a bit more nuanced however due to the amount of data (9 regions each with 126 features per hour), 96 hours ended up with the best trade-off.

Best model metrics:

Overall R^2 score average across all days (unscaled regression): 0.354

📈 R^2 scores for each forecast day (unscaled regression):

Day 1: R^2 = 0.837

Day 2: R^2 = 0.580

Day 3: R^2 = 0.398

Day 4: R^2 = 0.268

Day 5: R^2 = 0.180

Day 6: R^2 = 0.120

Day 7: R^2 = 0.094

```
                        precision    recall  f1-score   support

       No Rain (≤0.1mm)       0.94      0.96      0.95     76042
   Light Rain (0.1-2mm)       0.16      0.36      0.22      2488
Moderate Rain (2-10mm)        0.15      0.10      0.12      3069
    Heavy Rain (>10mm)        0.00      0.00      0.00      2184

              micro avg       0.87      0.89      0.88     83783
              macro avg       0.31      0.36      0.32     83783
           weighted avg       0.86      0.89      0.87     83783
```

Note on the classifier missing heavy rain, while weak, it still helps the regressor which is the main goal, this is a form of a weak auxiliary task, and it improves the generalization of the main task.

Modularity and expansion

The model presented is highly complex, however; the goal was to allow further investigation and expansion. The comments contain notes to guide future changes, and the code contains debugging sections as well as snippets that fix reoccurring issues. There are functions that allow remapping for the classifier should the labels change. Code cell included for testing and debugging shape mismatches: The code has a lot of data processing and feature engineering. If changed at any point, it is very easy for shape mismatches to occur, a short and simple code block is included for cutting out duplicate columns and/or dropping features to quickly test the code (final implementation should NOT utilize this, but very useful to quickly test and mend an otherwise complex pipeline).
Additional skeleton implemented for adding other inputs into the model, can be freely renamed and added in, mostly an example for future use but currently negated with the arguments “extra_front_dim=0, extra_wind_dim=0” passed when creating the model.

Issues and optimizations

With a complex pipeline such as this, there were many issues and considerations. Starting with the most obvious, memory and performance. Clearly there is a LOT of data being processed, apart from the aforementioned sequence length being optimizied to strike a balance between metrics, performance and memory demands, the hardware demand was still large, therefore many optimizations were performed. Some examples include, using float32 explicitly as the default is float64 which straight away cut the memory demand in half. Sequence creation strikes a balance between model performance and memory demand, particularly the classifier. Clean feature engineering and adding into the dataframe ensures no fragmentation and faster dataset processing. Model depth and complexity optimized for balance (example, hyperparameter tuning, particularly of the number of heads in the model, doubling the heads results in doubling the time but only roughly 5% improvements across r2 scores, with last day going from 0.094 to 0.156).

As mentioned before, the code contains a lot of debugging code as modifying the data processing pipeline often leads to many issues down the code, particularly when fitting the model.
Since the model is a hybrid, keras does not allow weights during the fitting of the model. Therefore, the compilation stage implements a custom weighted loss function. The

dataset is highly imbalanced, and the classifier will falsely predict little to no rain if left with no weighting.
Data leaks is a huge aspect of such models, therefore manual splitting is implemented to ensure no leaks are present.
Calculating metrics also becomes a problem if using regular sklearn functions, the output of the models has to be first processed as the model will return predictions and classification for each of the days.