Dokumentation zu WeatherPredicter

Autor: Shazil Khan

GitHub Link: <https://github.com/Bro-tec/WeatherPredicter.git>

Table of contents

Introduction

Main Section

* *Data collection*
* *Data extraction*
* *Machine learning*
* *Visualization*

Closing

Introduction

This project is about collecting weather data and predicting selected parameters using a Long Short-Term Memory (LSTM) model.

Data is collected by the German Weather Service and fed into the LSTM model.

The aim is to generate forecasts for various values such as temperature, wind speed and weather conditions for the coming days.

Several models were trained, which are divided into two variants, each with different data inputs and outputs.

These model variants are referred to as "Hourly" and "Daily".

Hourly is a less computationally intensive and therefore more resource-efficient variant that makes predictions based on a limited number of input values.

It generates six outputs, including temperature, wind direction, wind speed, visibility and weather conditions 1 and 2.

Daily, on the other hand, is a more computationally intensive and complex variant that uses a large number of input values, which slows down the training process.

In turn, it offers six output values, including minimum and maximum temperature, minimum and maximum wind speed and weather conditions 1 and 2.

The motivation behind using both methods is that Hourly requires fewer resources to generate trained models.

However, these are less accurate in their predictions.

Daily, on the other hand, requires more resources, which slows down the training process, but enables more accurate predictions.

Main Section

The idea is to use machine learning to create a weather forecast using Python as the main language.

First, weather data was searched for and converted into data sets (features).

The data for which forecasts were to be made were also converted into data sets (labels).

These were passed on in two procedures called Hourly and Daily.

The next step was to create our own LSTM models, whereby several training files were created.

The data sets were then used to train the models.

During training, the loss value was evaluated and displayed in a diagram.

Furthermore, the accuracy of the predictions was determined.

If labels were used, these were also displayed in the form of a confusion matrix.

Finally, the predictions were visualized together with the labels to see how accurate the models are.

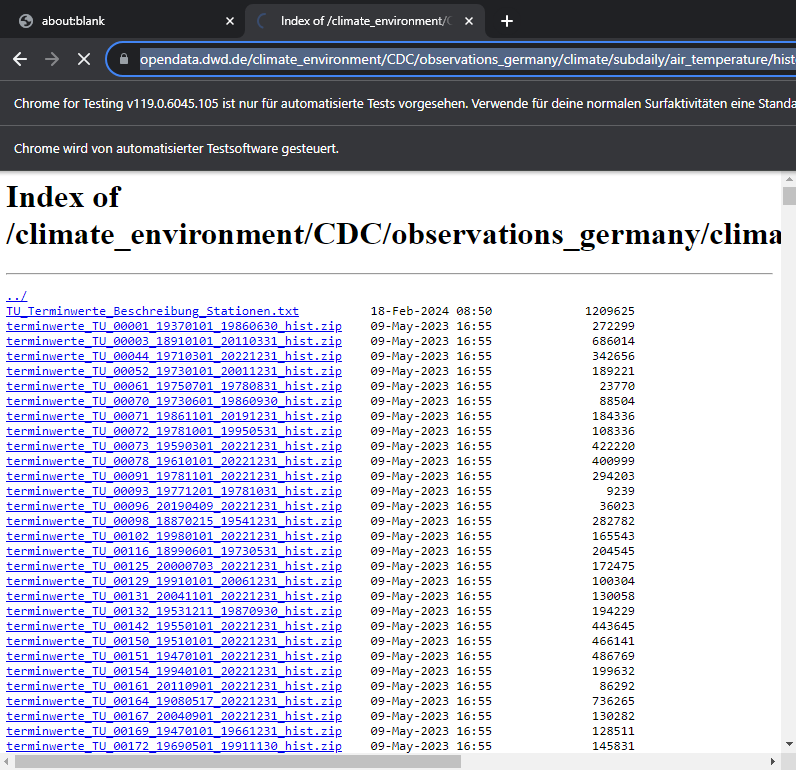
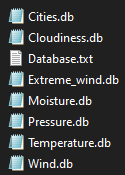
Future predictions without labels are also possible.

Data collection

Although there were few delays in collecting data, there were significant delays.

Various data collection methods were tried out.

First, data was collected from the website of the German Weather Service (DWD) using a web crawler in Node.js.

The collected data had to be merged into databases as it was too cumbersome to extract them individually.

While working with the data, it was found that a lot of data was missing and there were many zero values that were not filled in for various reasons.

In the search for higher quality data, the OpenWeather application programming interface (API) was tried.

However, it turned out that this was chargeable and therefore not suitable for the project.

After numerous further evaluations, Bright Sky, an API from the DWD, was finally used.

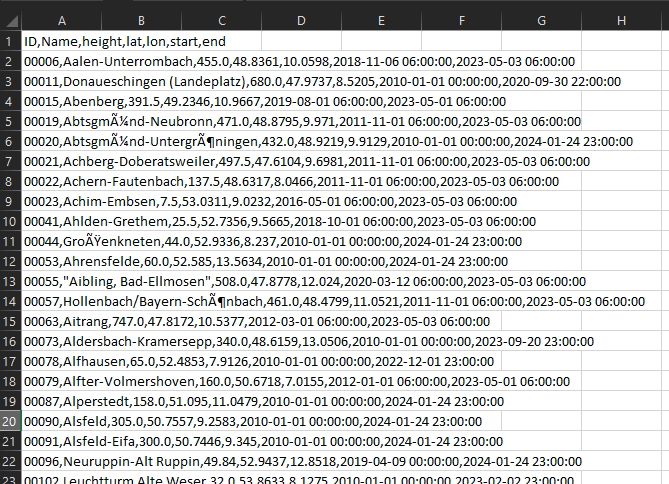
Although it provided many zero values, accessing the data sets was less cumbersome.

They did not have to be cached and a lot of data was directly available.

Data extraction

To generate the data, the ID of the city and the date are required first.

The IDs of the cities are collected by starting the file get\_DWD\_data in the CollectData folder and saved in an Excel table with the city data. This file also contains the start date of the collected data for each city, which can be used to set the start date.



The data obtained was converted into data sets and trained with a ready-made Keras program.

This process took around 30 minutes or even longer for an epoch with 200 data sets.

For this reason, Asyncio was used to parallelize the data collection.

This step reduced the time to collect all 1561 dates on a given date and convert them into usable datasets to between one and five minutes, depending on the quality of the internet connection at the time of training.

The data collection was realized with two different approaches: the Hourly method, which is supposed to predict the values of the next hour and, if possible, also those of the next day, and the Daily method, which is supposed to predict the values of the next seven days.



The Hourly method is a less computationally intensive model collection that is based on fewer features and can be trained more quickly. The features for each city amount to 17 values, which are combined with the values of the next four cities. Since the day has 24 hours, 24 features are output for each city that has been successfully issued. In addition to these 85 data, the hour is added to each feature, so that the features amount to 86 values. The labels, on the other hand, are made up of the six values: temperature, wind direction, wind speed, visibility and weather conditions 1 and 2.

The daily method, on the other hand, is a more computationally intensive model method, as it is based on more features, which leads to longer training times. As with hourly, the features for each city amount to 17 values, which this time are combined with the values of a total of eight cities in four cardinal directions at both shorter and more distant distances. Since the day has 24 hours, all 24 features are combined for each successfully spent city. The number of these features per city is therefore 7752 values. The labels, on the other hand, are made up of the six values: minimum temperature, maximum temperature, minimum wind speed, maximum wind speed, and weather conditions 1 and 2.

Machine learning

Since too much data was used at once, the training of the data was to be carried out on an Nvidia graphics card, whereby this process was to be realized with CUDA.

Since the training algorithm was realized in Keras, we tried for two weeks to get CUDA running with TensorFlow.

This rendered several environments unusable, and both the driver and Anaconda had to be reinstalled several times.

Due to the loss of time, PyTorch was used as an alternative.



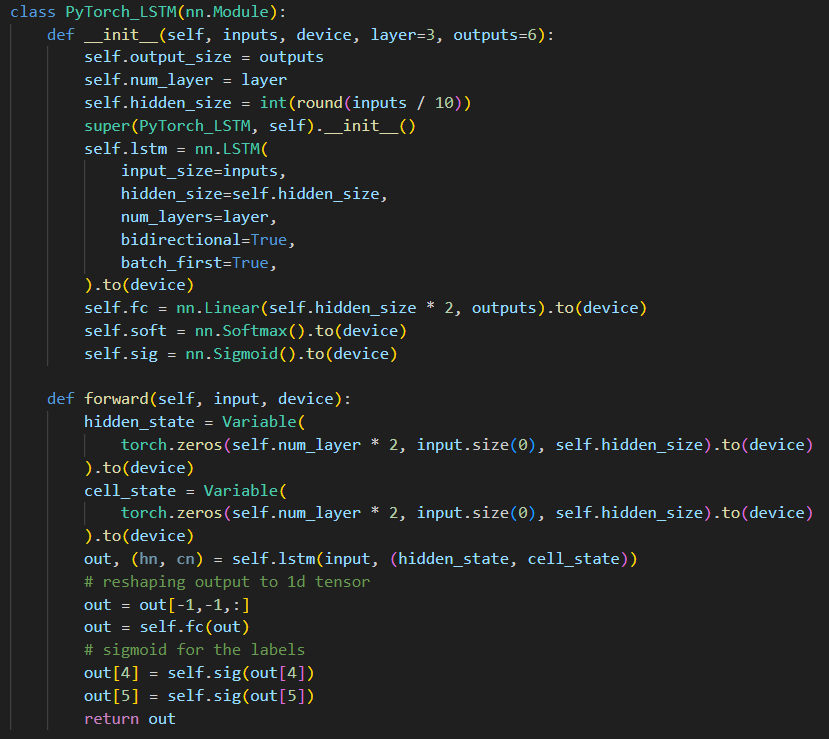
As with Keras, an LSTM was used, which receives the data sets in the form of tensors and trains them on the graphics card.

Initially, a Kullback-Leibler divergence loss was used, but after several attempts it was finally switched to a cross-entropy loss.

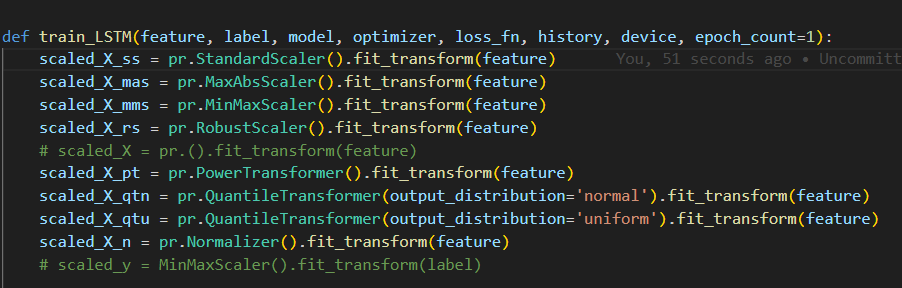
Accuracy is calculated as a percentage of the sum of the correctly predicted values.

For the labels, the mistake was made for a long time during training to use Softmax, which maximizes one value and minimizes all others. The mistake is that among the six classes, not the one with the highest probability is selected, but that scaled solution values are output from the values, which is not what the Softmax is intended for.

The values within the last two tensors of the data set are labels, but these should output positive numbers, which was realized with Sigmoid.



The data sets also had to be scaled. While various attempts were made to scale the input data, it was decided to leave them unchanged as they were different values.



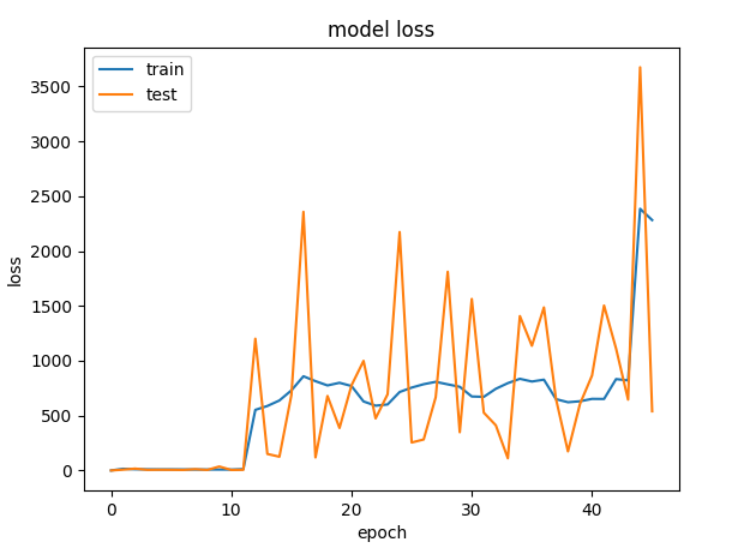
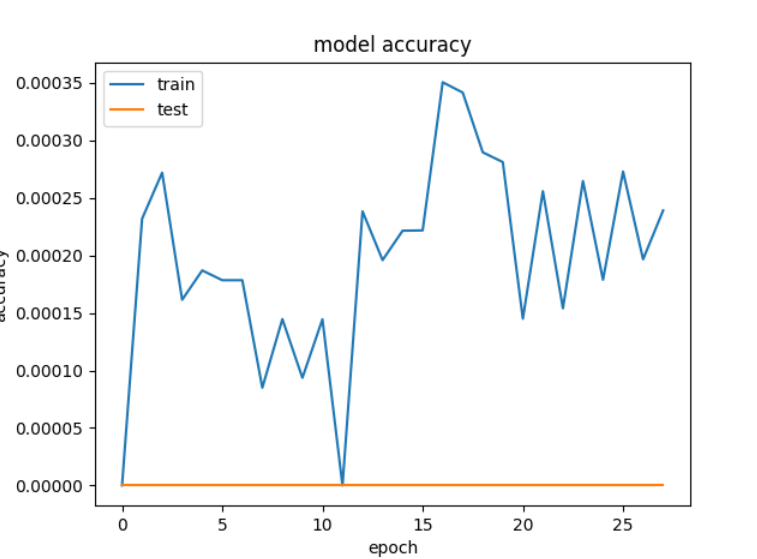
The labels, on the other hand, were scaled using a separate function.

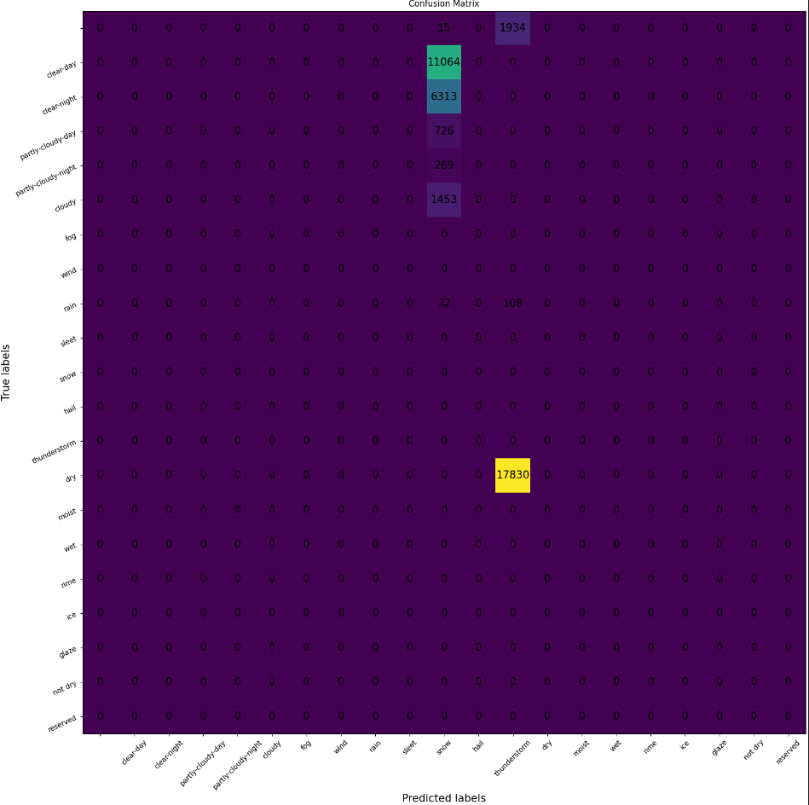
The predicted outputs were also converted into usable values using a separate function.

Visualization

To visualize the evaluation, the accuracy and loss values were displayed in diagrams.

The classifiable values in the output were displayed as a multiclass classification.

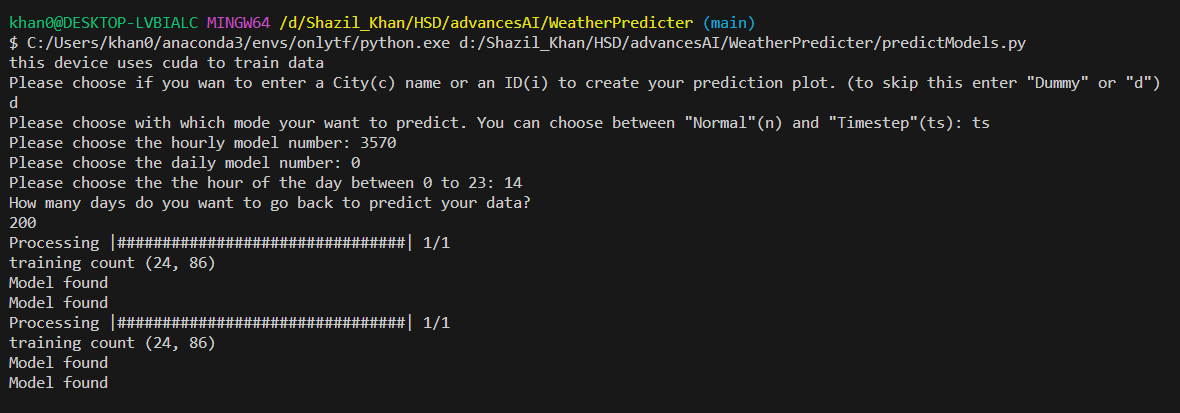
 



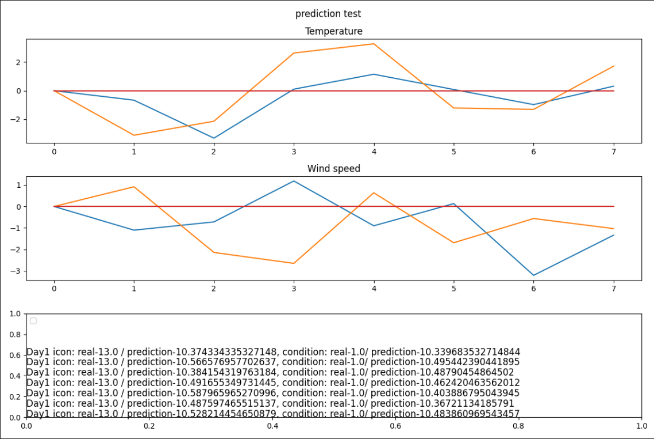
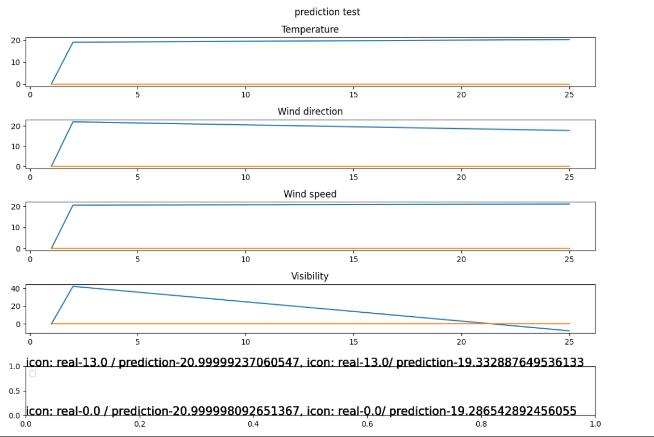
*These are bad issues and will be improved as soon as the training is completed*

*Finally, the outputs were displayed separately with a selected model in diagrams.*

In order to meet the requirements that a user can train these models, the files were edited in such a way that the user can train the models and make their own predictions by interacting with the console.



All diagrams are saved in the "Plots" folder and can be viewed there.

*Hourly Daily*

*These are bad issues and will be improved as soon as the training is completed*

*Finally, the outputs were displayed separately with a selected model in diagrams.*

If sufficient memory is available on the end device, all models in timeseries mode can be saved separately numbered after each epoch to counteract the loss of accuracy caused by overfitting.

Closing

The project was successfully completed and fulfills all the requirements for training models that make good predictions.

The files can also be executed by people without basic knowledge in the field of machine learning, which means that even non-experts are able to train these models.

However, the models were not fully trained, so only unfinished models could be evaluated and tested.

For future projects, it is recommended to use APIs, as they can provide a lot of data without much effort.

Furthermore, the finished models should be made available for use. If large amounts of data need to be trained, it is advisable to use PyTorch from the outset, as it can be used to train successfully and without many problems on CUDA-capable devices.

Translated Using DeepL

Source:

https://www.deepl.com/de/translator