Predicting Daily High Temperatures with Long Short-Term Memory Models

Tristan Lee
March 25, 2024

1 Introduction

I detail the usage of Long Short-Term Memory (LSTM) models to predict the daily high temperatures of a location in each of Austin, Chicago, Miami, and New York City. The models are trained on historical weather data of various sources, with some dating back to the year 2000. Somewhat expectedly, I find that the models are able to predict gradual changes and patterns in temperatures well, but perform poorly when the the temperature changes more dramatically. I also briefly describe the usage of the Kalshi platform to log predictions, and the contents/file structure of this repository.

2 Data Collection and Aggregation

2.1 Data Sources

I used 7 different data sources throughout this project. 5 of them were used for collecting historic data before March 4th to train the models, then 2 different ones where used to collect data from March 4th on to feed to the model to make live predictions. For the historic data:

- NOAA/NWS NCEI Climate Data Online (CDO): NOAA's tool for geting historic data. This had data from January 1st, 2000 to March 3rd, 2024 for all 4 locations that I bulk ordered as a CSV file. The locations here corresponded to the weather stations asked for in the common task description.
- Meteostat: A Python API that had data from the same date range and location coverage as NOAA. The API gives weather data given a certain latitude and longitude, and the documentation states that it itself takes the data from a combination of OpenWeatherMap and certain European sources.

- OpenMeteo: An online tool that had the same data range and location coverage as NOAA, and also allowed bulk CSV download. It's data is accurate up to a 3km-by-3km square around the location in question.
- VisualCrossing: An online tool that allowed for bulk download, but not the same date range or location coverage because of the API call limit. I downloaded 1 year of the most recent data for Chicago and New York from this source. The data is accurate up to the latitude/longitude I provided as input.
- World Weather Online (WWO): An API endpoint from which, similar to VisualCrossing, I got 1 year of data, though from Austin and Miami, because of the API call limits. This API also took latitude/longitude as input.

And for the current data:

- NOAA/NWS Daily Climate Report (CLI): The NWS's daily weather reports from their various weather stations. From my understanding, the data here should be the same as the CDO, but the web interface for getting just one day's data from this CLI is easier than ordering the CDO data, so I used this slightly different source for adding current data day-to-day.
- Wunderground: A site similar to Accuweather that has a web interface for viewing weather data for various days. This source had data that the CLIs did not, which I'll explain next.

2.2 Features

From these sources I collected 5 features for each day: maximum temperature, precipitation, pressure, cloud cover, and relative humidity. My reasoning for using more features than just the temperature was that a model may be able to learn the relationships between all these data points; for example, that low pressure is usually followed by cloud cover and storms, thus affecting the temperature.

2.3 Aggregation

For each location, I aggregated all the historic data into a single table, which had these 5 features for each date in the range January 1st, 2000 to March 3rd, 2024. For each date, I took the average of all available data for that date to be the table's value. For example, NOAA's CDO, Meteostat, and OpenMeteo all had data for the entire date range in question. The CDO had precipitation and temperature data, but did not have relative humidity, pressure, or cloud cover data, while Meteostat and OpenMeteo had all 5 of these. So for dates in the range January 2000 to March 2023, the table's temperature and precipitation data was the average of these 3 sources, while the table's humidity, pressure, and cloud cover data was the average of 2 sources: Meteostat and OpenMeteo. Also, the most recent year's data counted VisualCrossing's and WWO's data in the average, for Chicago and New York, or Austin and Miami, respectively. So the final result is, for each location, a single table of historic data containing the 5 features for each day considering data from 4 different sources. For adding the current data live, the CLI had all features except the pressure, which I got from the Wunderground source.

3 The Long Short-Term Memory Model

3.1 Model Choice

I figured an LSTM was an appropriate choice for predicting weather conditions of the next day, because they heavily depend on both the conditions of days immediately before the day in question, and the general seasonal trend of temperatures around that day (for example, general warming going from winter to spring during March, or cooling going from summer to fall during September). An LSTM is suitable for learning both the short term day-to-day and long term seasonal affects on temperature.

3.2 Input/Output and Training

I trained a separate LSTM on each location's data. A single instance of input is a sequence of 30 days' worth of weather conditions, and the output is a prediction for the next day's conditions. I used 4 total layers for each LSTM: an input layer, 2 LSTM layers each with 50 units, and a final dense layer. Honestly, I wasn't sure of what parameters to use for the model, so besides the number of units (I read that 50 was a decent default value to use), I just used TensorFlow's default values for everything else (see the source code for more details). The 2000-to-present historic data that each location has corresponds to 8830 days, giving $8830 - 30 = 8800$ contiguous sequences of 30 days followed by another day (alternatively, contiguous sequences of 31 days). Thus the training data consists of 8830 instances: a sequence of 30 days' weather conditions as input, and the following day's conditions as the ground truth label, where each day is a vector of 5 features. Before training, the data was scaled to fit in the interval $[0, 1]$. Finally, I used Adam for optimizing, and mean squared loss as the loss function.

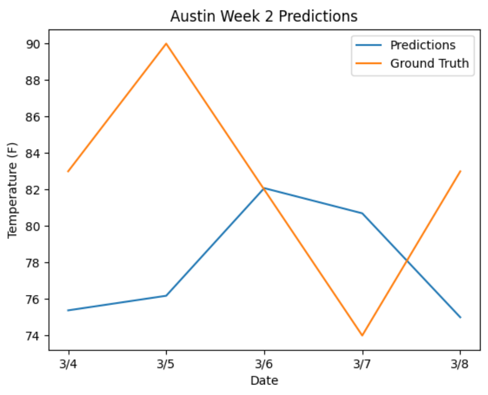
3.3 Predicting

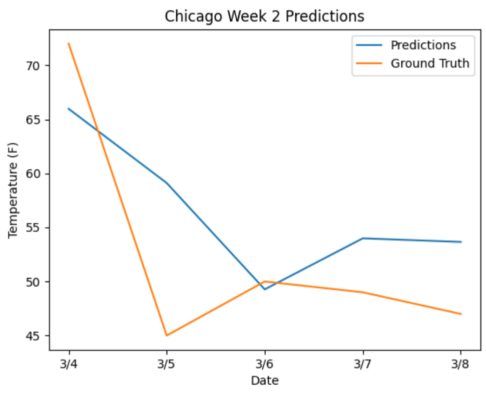
Since I had to continuously predict values throughout the week, I manually added the ground truth values to the CSV file being read from as the days went on. So, to predict the max temperature for March 4th, 2024, the LSTM took as input the sequence of 30 days' of weather conditions before March 4th (February 3rd to March 3rd), and outputted its vector of predicted weather conditions for March 4th. Then I just took the temperature component of that vector as the prediction. Then, to predict March 5th, I added March 4th's true observed conditions from the CLI and Wunderground to the data, so now the LSTM takes as input the sequence of days from February 4th to March 4th to predict March 5th (see the source code for more details).

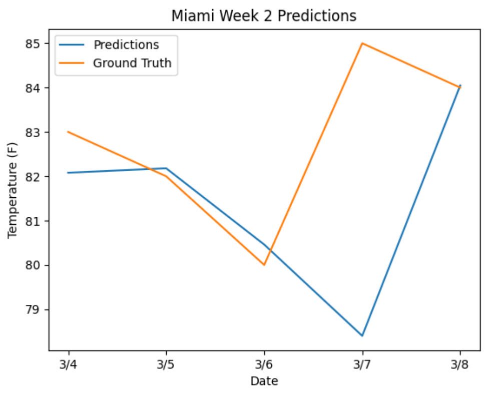
4 Results

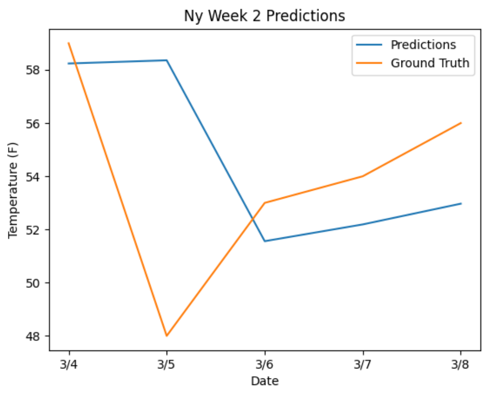
4.1 Predictions

Here are plots for Week 2's predictions and ground truth:







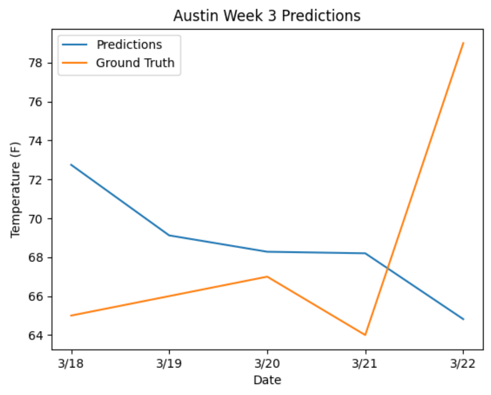


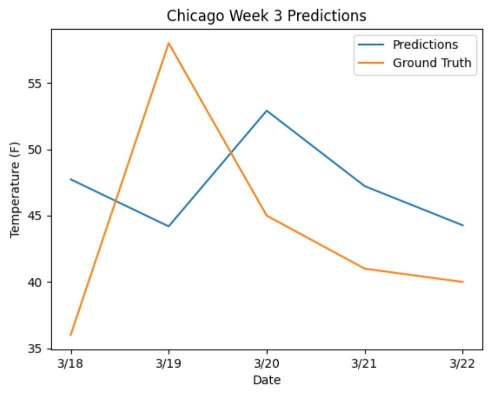
The model appears to "correct" its prediction based heavily on the recent day-to-day trend. For instance, consider March 4th through March 6th for Austin. The model seems to take the drastic increase in ground truth temperature from the 4th to the 5th into account for its prediction for the 6th by "increasing" its March 5th prediction a similarly drastic amount. Consider also the decrease from the 5th to the 6th, where the model "decreases" its March 6th prediction to get its March 7th one, then again from the 6th to the 7th. Also, the fact that these temperature drops were consecutive seems to affect the model's prediction as well, as the magnitude of decrease in prediction from the 7th to 8th is greater than that of the 6th to 7th. Similar "corrective" behavior can be observed in the other 3 locations: March 5th in Chicago and New York, and March 7th in Miami.

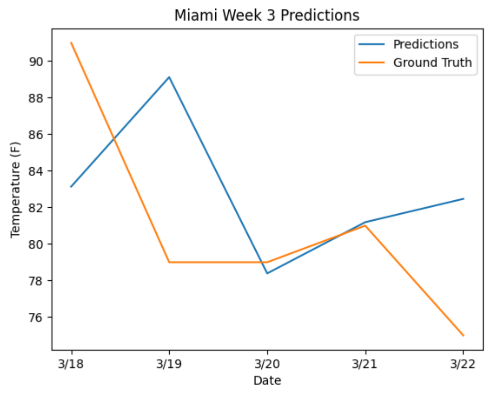
These large errors occurred when the temperature fluctuated drastically from the previous day. However, when the temperature maintains a steady trend, the model performed generally better. Notable areas include March 6th through 8th for both Chicago and New York, and

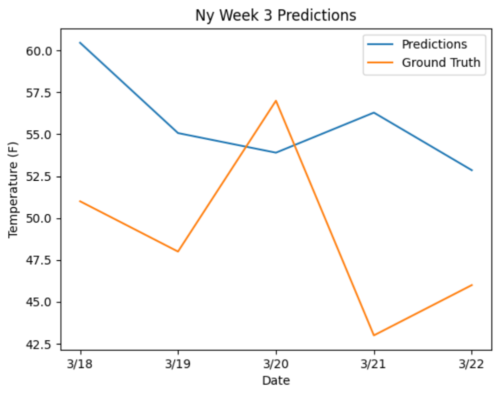
March 4th through 6th for Miami. With steady temperatures, the model can leverage its knowledge of the previous 30 days combined with smaller corrections to account for the immediately previous day, leading to more "steady" behavior.

Week 3 demonstrates more of this "corrective" behavior:









Major corrections include March 19th for both Chicago and Miami, and 18th through 20th for New York. Interestingly, despite Austin having consecutive increases in temperature from the 18th through 20th, the model still "decreased" its guess going from 20th to 21st, though barely by any amount. It seems here the magnitude of the increases weren't enough to outweigh the "momentum" of the long term prediction.

For reference, here are the raw prediction and ground truth values from which these plots are based. Note that the predictions were rounded to 2 decimal places, while the ground truths only take on whole values because the CLI only reports whole numbers.

Location	3/4	3/5	3/6	3/7	3/8	3/18	3/19	3/20	3/21	3/22
Austin Prediction	75.38	76.18	82.08	80.70	75.00	72.75	69.12	68.28	68.2	64.82
Austin Ground Truth	83	90	82	74	83	65	66	67	64	79
Chicago Prediction	65.97	59.11	49.27	53.99	53.66	47.74	44.19	52.91	47.22	44.27
Chicago Ground Truth	72	45	50	49	47	36	58	45	41	40
Miami Prediction	82.08	82.18	80.46	78.4	84.05	83.13	89.12	78.39	81.19	82.46
Miami Ground Truth	83	82	80	85	84	91	79	79	81	75
New York Prediction	58.24	58.36	51.56	52.19	52.97	60.46	55.07	53.9	56.29	52.85
New York Ground Truth	59	48	53	54	56	51	48	57	43	46

The average error across all days and locations was 5.87 degrees Fahrenheit. This error was higher where and when there were more noisy fluctuations in the temperature, like Week 2 for Austin or Week 3 for New York.

4.2 Possible conclusions

Somewhat expectedly, the model performed well on steady trends, but worse on highly fluctuating ones. The model seems to heavily weigh drastic changes in temperature and consecutive changes in its next day prediction, which can be highly erroneous if those fluctuations were just noise expected of the weather, but can be correct if the temperature enters a steady period afterwards. In general, the performance of these LSTMs seems to match their theoretical performance of tracking both long term and short term trends in their input sequences, as we saw both at play in the predictions.

5 Kalshi

Kalshi was used to log the predictions. For basically all trades, I placed limit orders for 10 contracts at 50 cents, which meant the order went through most of the time, but also meant I broke even only if I was right 50% of the time. As you can expect, using the model to make these trades resulted in a very high net loss, as the error was often well over the 2 degree windows Kalshi uses.

6 Repository contents

This repo contains a few things:

1. `report.pdf`, this report.
2. `data`: a folder containing the Python files used to collect and aggregate the data. Note that most of the data used in the collection/aggregation phase wasn't included in the repo for sake of space, but I did include the final aggregated tables.
   a) `clean_and_split_noaa.py`: this filters out most columns of, and separated into different locations, the single file I ordered with the CDO tool.
   b) `init_meteostat_data.py`: this uses the Meteostat Python API to fetch weather data for each location, using longitude and latitude, and stores the results in 4 separate CSV files (not submitted).
   c) `aggregate_open_meteo.py`: this aggregates the hourly data given by the Open-Meteo site into daily data, aggregating the features I want appropriately. Once again, this is saved to CSV files (not submitted).
   d) `init_world_weather_online.py`: this uses an API endpoint to get 1 year's worth of data for Austin and Miami, and saves it to a CSV file (not submitted). Note that it uses an API key that I also haven't submitted.
   e) `aggregate_all_data.py`: this aggregates all data from the various CSV files into a single file for each location, taking the average of all available values for some location, date, and feature.
   f) `austin`, `chicago`, `miami`, and `ny`: in my local copy, these folders hold all the data used to create the aggregated tables. In the repo, they only contain the aggregated tables gernerated by `aggregate_all_data.py`. They also contain the live updates I made during Weeks 2 and 3 using the CLI and Wunderground, so they have data up to March 21st, 2024.
3. `model_and_trade`: a folder containing the sources files used to train and predict with the model, as well as the files using the Kalshi API to automate trades.
   a) `train_lstm.py`: this trains a model for each location based on that location's aggregated table. The actual `.keras` files for the models and scalers used for normalization haven't been submitted, once again to space's sake.
   b) `predict.py`: this reads from the most recent 30 days of the aggregated table, which includes my current weather updates, to predict next day's weather. It uses the saved scalars to normalize the current data in the same way as the test data. All of the functionality is placed in a function to be called by the trading file.
   c) `KalshiClientBaseV2.py`: this file was provided by the Kalshi API documentation site, and contains helper functions for the API's various endpoints.
   d) `kalshi_trade.py`: my file that calls the prediction functionality and places orders based on that, using the functionality in `KalshiClientBaseV2.py`. The only

nontrivial point left to mention is that the guess for high temperature is contained within Kalshi's ticker, so the file checks for the ticker with the closest temperature to the prediction, rather than place a bet for a temperature interval for a single "Austin March 22nd, 2024" ticker, for example.

4. `plots`: a folder containing just the script `model_eval.py` used to generate the plots in this report. It generates actual PNG files, which, for space's sake, I haven't included.