Weather Prediction

By Yash Patel, Vinod Raman, Seamus Somerstep, and Unique Subedi

Introduction

In this project, we are tasked with predicting the minimum, average, and maximum temperature for the next five days for 20 different weather stations. This report summarizes our data pipeline and the models we used to tackle this task.

Data Sources and Preparation

To leverage different strenghts of publicly available datasets, we used NOAA and Wunderground as our data sources in the following manners:

- NOAA: used to acquire historical data going back to 1960
- Wunderground: used to acquired recent data, not restricted in timespan in theory but only used for the final year of analysis in the models presented

Acquiring data from NOAA was straightforward after doing a conversion of the airport codes to the codes in the NOAA database, using the code below. By default, results are saved into `data/raw_noaa`:

We simply had to parse the DLY files acquired from the NOAA database, which could easily be done as follows. By default, results are saved into `data/processed_noaa`:

Acquisition from Wunderground, however, was complicated by several issues. These, however, were worth navigating, since Wunderground was ultimately used as the ground truth data source in the final evaluation. The source of the complication was that Wunderground fetches its content dynamically, making a naive web scraper useless to get the data, as seen initially in loading a Wunderground page:

To circumvent this, we found the resources that were being fetched by the page by looking at the resources through Google Chrome's network analysis:

From this, we were able to find that the contents of the hourly data, shown at the bottom of the Wunderground page, is fetched from a weather.com API endpoint. From the network analysis, we found the corresponding cURL command to fetch this content directly:

With this cURL command in hand, we could then fetch these programmatically in Python by directly querying the endpoint:

Some care had to be taken in making the requests, since sending one cURL request per day would be very inefficient: instead, we are able to batch requests and send them in parallel. Note that this requires multiple cores, meaning the final version run through Docker (if restricted to a single CPU) does not leverage this. However, this part of the data fetching processing is all done offline, meaning we can take advantage of multiple cores in this case. By default, results are saved into `data/raw_wunderground`:

Using this fetches a list of the *hourly* measurements, i.e. temperature, wind speed, wind direction, etc… However, the final task was specificallly interested in measuring attributes at the daily time scale, meaning such granular measurements would not be useful. Unlike the hourly contents itself, the Wunderground site does *not* retrieve its daily summaries from an endpoint: after all, doing so would be redundant given that the latter can be directly computed from the former. Therefore, the Wunderground site both dynamically fetches the hourly content and then computes the daily summaries that are shown. We, therefore, had to replicate this ourselves to get the min, max, and average temperatures for training.

This naively could be done simply by aggregating the contents to each unique day in the dataset and taking the min, max, and average across such datapoints. However, naively doing so produces incorrect answers, since the data are *returned in a single time zone* while the summative values are *computed based on the time zone of the station*, i.e. Alaska's min, max, and average will be taken in the Alaska/Anchroage time zone. Similar to the hourly data, the timezone could be queried using a separate endpoint, as follows:

With this, we can finally do the aggregation as follows. By default, results are saved into `data/processed_wunderground`:

Evaluation Pipeline

With the processed data, we wished to produce datasets for cross validation that parallel the final test set, which was to be Nov 30-Dec 10. The most natural evaluation test, therefore, would be looking at the same timeframe across multiple years. We start by fetching *all* the data necessary to construct this evaluation task. In particular, for a window of Nov 30-Dec 10 2021, we would need Wunderground data from Nov 30 **2020** - Dec 15 **2021**. Note that how far we extend back is arbitrary (training could have extended for more than one year) and that we also need data 5 days beyond the end of the evaluation period, since for *each day* in the evaluation window, we predict the following 5 days min, max, and average temperatures. Therefore, we start by fetching this overall dataset:

From there, for each day in the evaluation window, we artifically chop off the dataset to simulate as if we were predicting from that day of interest. For example, if we are doing predictions on Dec 1 2021, we return the dataset view from Nov 30 2020 - noon EST Dec 1 2021. NOAA is returned up to 3 days from the prediction day, since we NOAA is known to lag by about 3 days from the current day. For each day, the "target" was then compiled to be the `[min, mean, max]` temperatures for the five days looking forward from the prediction day for each of the stations, as follows:

With this, we had a unified framework for evaluating models: given the evaluation period, we take the chopped dataset, feed that to the model being evaluated, and compute the MSE as our evaluation metric of the predicted against the target min, mean, and max daily temperatures, which was repeated for multiple years to assess the robustness of findings:

Prediction Models

Baselines

In this section, we discuss simple, computationally-efficient baseline models for predicting the minimum, average, and maximum temperatures for the next five days. These simple models enable us to better evaluate the performance of the more sophisticated models we tried.

Previous Day Predictor Our first baseline model, termed the Previous Day Predictor, exploits the idea that for every station, today's minimum, average, and maximum tempertures are good estimates for the minimum, average, and maximum temperates for *each* of the next five days. To this end, the Previous Day Predictor predicts for each station, for each of the next five days, today's minimum, average, and maximum temperture as the respective minimum, average, and maximum temperature for that day. That is, for each station, the Previous Day Predictor outputs a vector of 15 values consisting of today's minimum, average, and maximum temperatures repeated five times in a row. The model class below formalizes this idea in code.

Note that Wunderground data is used to compute "today's" minimum, average, and maximum temperature since the NOAA dataset lags by a couple days.

Historical Average Predictor Suppose we want to make a prediction for Dec 1. The rationale for this model is that the temperature of this year's Dec 1 will be similar to that of past Dec 1's. In particular, let y_i be the maximum temperature of Dec 1 for year i. In NOAA dataset, we have $1941 \leq i \leq 2021$. Then, the prediction of maximum temperature of Dec 1, 2022 is

$$\frac{1}{2021 - 1941 + 1} \sum_{i=1941}^{2021} y_i.$$

So, our baseline is to predict historic average of minimum, average, and maximum temperatures for each station on that particular day. Unfortunately, this baseline did not end up performing that well for us.

Weighted Average Predictor The Weighted Average baseline predictor interpolates between the Previous Day Predictor and the Historical Average Predictor. At its core, it exploits the idea that tomorrow's temperature will be similar to today's temperature but the temperate 5 days from now will be more similar to the historical average temperature for that day. To this end, the Weighted Average Predictor computes a weighted average between today's temperature and the historical average temperature for each day and station. In other words, for each station, for each day, for each measurement (i.e., min, avg, max), the Weighted Average Predictor computes a weighted average between today's temperature and the historical average for the corresponding day. The model class below formalizes this idea in code. Note that it takes in as input a fixed set of mixing weights and uses the PreviousDayPredictor and the Historical Average Predictor as black-boxes.

Note that the Weighted Average Predictor uses the same fixed set of 5 mixing weights between today's temperature and the historical average temperature across all stations and measurements. That is, there is a single mixing weight associated to each day out. The table below provides the mixing weights used in the instantiation WA_pred.

Days out	Weight on Today's temperature	Weight on Historical Average temperature
1	1	0
2	0.80	0.20
3	0.60	0.40
4	0.40	0.60
5	0.20	0.80

Observe that the weiht on Today's temperature decreases as the number of days out increases and vice versa for the weight on the historical average temperature. This coincides with the idea that tomorrow's temperature is most similar to today's temperature, but the temperature five days from now will be more similar to the historical average temperature for that day.

Time Series Models

ARIMA, which stands for Auto-regressive Integrated Moving Average, are general models for time series forecasting. The ARIMA model for stationary time series are linear regression models and one set of its predictors are lags of the variables to be predicted. A key concept in time series modeling is stationarity, which roughly says that the statistical properties of the time series is constant over time. The time series may not be stationary as it is, so we have to apply some linear/non-linear transformation to the data to bring stationarity. For example, differencing is a popular technique to bring stationarity. Suppose Y_t be the maximum temperature at time stamp t. Then, the time series $\{Y_t\}_{t\geq 1}$ may not be stationary, but the differenced variable $y_t = Y_t - Y_{t-1}$ perhaps is. Generally, we may have to use higher order of differencing. For example a second order differencing involves transforming time series to $y_t = (Y_t - Y_{t-1}) - (Y_{t-1} - Y_{t-2})$. On top of lagged difference of the variable, this model can also use the forecast errors in the past time steps, which are called moving average components. These errors denoted by $\{\epsilon_t\}_{t\geq 1}$ are assumped to to iid Normal$(0, 1)$. Therefore, an ARIMA model with p auto-regressive component, d order of differencing, and q moving average components is

$$y_t = \mu + \beta_1 y_{t-1} + \beta_2 y_{t-2} + ... + \beta_p y_{t-p} + \theta_1 \epsilon_{t-1} + ... + \theta_q \epsilon_{t-q},$$

where $y_t = (Y_t - Y_{t-1}) - ... - (Y_{t-d+1} - Y_{t-d})$. We use ARIMA$(p, d, q)$ to denote this model.

We trained three ARIMA models per station each for minimum, average, and maximum temperature. Thus, we have 60 ARIMA models all together for 20 stations. Each model forecast respective temperature for next five days. The model class below formalizes this idea in code.

We obtained decent results with ARIMA model, but not in par with our regression based models. Moreover, we tried ARIMA models that takes into account various seasonal trends (monthly, yearly, weekly), but the model fitting took much longer with only marginal improvements in the results.

Regression-Based Models

In this section, we present prediction models based on Regression.

Features for Regression-Models For our regression-based models, we exclusively used data from Wunderground. Data from this source is returned hourly so for feature preperation we aggregated the into daily data using the following calculations. Average temperature is calculated as the average of the hourly temperatures while max/min temperature are calculated as the highest and lowest hourly temperature over the day respectively. Heat index, dew point, pressure and wind speed are aggregated in a similar manner. Note that heat index is a human percieved temperature feature which combines heat with humidity. The dew point is the temperature at which the relative humidity becomes 100%. Finally wind direction was computed as the mode hourly wind direction from the possible categories of (CALM, S, SSE,SEE, E, NEE, NNE, N, NNW, NWW, W, SWW, SSW, VAR). Below is an aggregated table of the features used.

Features
Daily Min/Max/Avg Temp
Daily Min/Max/Avg Pressure
Daily Min/Max/Avg Heat Index
Daily Min/Max/Avg Dew Point
Daily Min/Max/Avg Wind Speed
Daily Mode Wind Direction

The code below gives a snapshot of the Wunderground dataframe for a particular station.

temp_min

wspd_min

pressure_min

heat_index_min

dewPt_min

temp_mean

wspd_mean

pressure_mean

heat_index_mean

dewPt_mean

temp_max

wspd_max

pressure_max

heat_index_max

dewPt_max

wdir_mode

2021-10-05

35

0.0

29.76

35

27

39.652174

5.043478

29.810435

39.652174

30.869565

44

12.0

29.88

44

35

NE

2021-10-06

42

7.0

29.35

42

32

43.875000

17.625000

29.519167

43.875000

34.000000

46

25.0

29.76

46

36

SE

2021-10-07

41

0.0

29.36

41

34

44.107143

8.607143

29.472857

44.107143

36.964286

49

21.0

29.55

49

39

SSE

2021-10-08

41

0.0

28.95

41

35

42.066667

6.733333

29.062333

42.066667

36.433333

44

17.0

29.31

44

38

SSE

2021-10-09

42

0.0

28.96

42

36

43.511628

6.534884

29.249302

43.511628

38.697674

47

10.0

29.51

47

41

S

Creating a Regression Dataset Based on the features above, we created a Regression data set (X, y). The ith row of (X, y) is associated with a particular date i such that $X[i]$ is a 48-dimensional vector containing the aforementioned features for the previous 3 days ($3*16 = 48$ covariates in total) and $y[i]$ is a 15-dimensional vector containing the daily minimum, average, and maximum temperatures for the next 5 days. As an example, if today's date was 10/11/2020, then the corresponding rows in X and y will contain the features for the days 10/8/2020 -10/10/2020 and the temperatures for the days 10/12/2020 - 10/16/2020. The function below uses Wunderground data for a given station to construct and output a regression dataset for that particular station.

The input `data` contains the Wunderground data for a specific station. The input `window_size` controls how many days of features should be included in each row of X, but this was fixed as 3 (as mentioend above). The input `features` controls which features (i.e. columns of X) should be kept when constructing the regression dataset.

A Meta-Predictor In order to expediate the process of model exploration, we additionally implemented a MetaPredictor class which takes in as input a blackbox implementation of a regression model, and uses it to make predictions. That is, instead of having to create a a new model class for every type of Regression model we want to try, we can now simply instantiate and pass any regression model as input into the constructor of the MetaPredictor. The MetaPredictor will then train 20 copies of the specified regression model on the regression dataset created by calling `create_regression_data` for each station, and then use each of the trained models to make a prediction of the temperatures for the next 5 days. The model class below formalizes this idea in code.

Note that each time the function predict is called, the MetaPredictor, makes 20 regression datasets, one for each station, trains the specified regression model on each of the datasets, and uses the trained models to make predictions for the next 5 days for each station. This process is done sequentially, one station at a time. Note that each time `self.reg.fit` is called, it erases its previous memory, so that a fresh new model is trained for each station.

In the next few sub-sections, we show how the MetaPredictor model class can be used to create a variety of different regression models.

OLS, Lasso, and Ridge Given the Meta-Predictor, it is easy to construct predictors that use OLS, Lasso, and Ridge Regression models for predicting the temperatures for each station. In particular, the code below shows how to construct each of these regression models using the Meta-Predictor class.

Note that by construction, the Meta-Predictor will train one regression model per station. Each regression model will take in as input the covariates previously specified, and output a 15-dimension vector corresponding to its predictions of the minimum, maximum, and average temperatures for the next 5 days. That is, for each station, the Meta-Predictor class will train a multi-output regression model. By using RidgeCV and LassoCV and passing in a list of candidate values for α, these sklearn model classes (RidgeCV, LassoCV) will automatically hyperparameter tune the regression model for each station. Once again, by construction, the Meta-Predictor will create a new dataset and train a new regression model for each station every time the predict function is called.

Multilayer Perceptron (MLP) A multilayer perceptron is a fully connected neural network. In particular, we used a two layer perceptron

$$y = W_2\,\sigma(\,\sigma(W_1 x + b_1) + b_2),$$

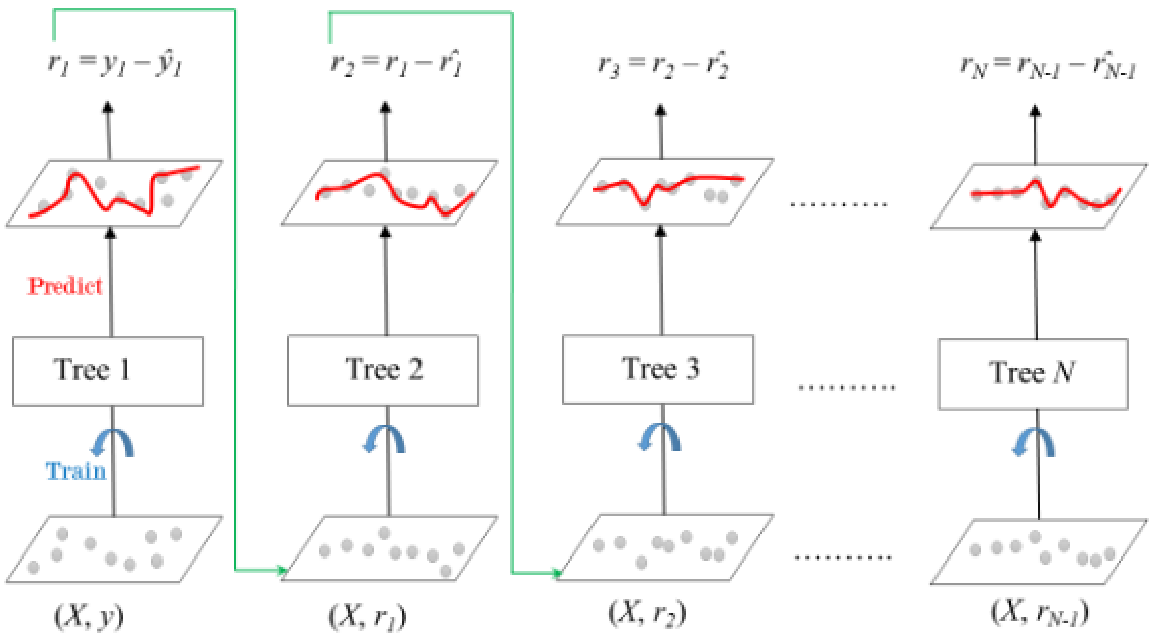
where the activation is relu. Each hidden layer is of dimension 20 and the output layer has 15 dimensions for min, average, and max of next five days. Since we trained one MLP model for one station, we trained 20 MLP regressors alltogether. The code below formalizes this idea by using the MLPRegressor class from sklearn and the aforementioned MetaPredictor class.

One main attraction of MLPs are that it allows for multitask regressions based on shared representations. It is particularly relevant here because the assumption of shared representation seems reasonable if we are predictive the response of same qualitative nature, temperature. We obtained pretty good results with this model, however, similar performance was obtained with ensemble methods with much less training time. So, we ended up choosing ensemble based models for deployment.

Ensemble Methods

Ensemble methods, like Boosting and Bagging, are powerful prediction models for tabular data that are known to generalize well in practice. These types of models contain sub-models of their own (like Regression trees) and make predictions by cleverly training and aggregating the predictions of these sub-models. In this section, we build weather prediction models based on Gradient Boosting and Random Forests - two popular Ensemble methods.

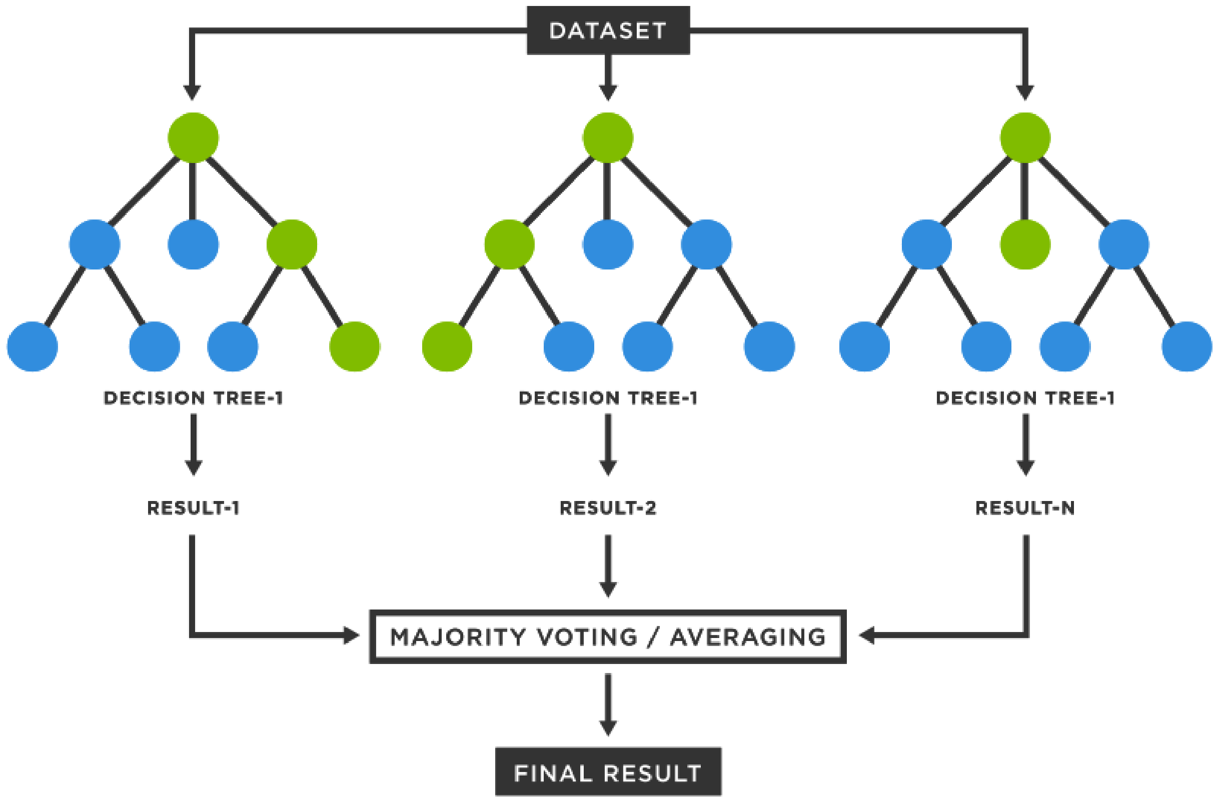
Gradient Boosted Trees Gradient Boosted Trees are an important class of Ensemble methods that trains a sequence of dependent Regression Trees. More specifically, Gradient Boosting trains Regression Trees sequentially, where the next Regression Tree is trained to correct the mistakes of the previously trained Tree. The Figure below captures this idea visually.



One important note is that Gradient Boosting can only be performed when the response variable is a scalar. Since we need to predict a vector of length 300 (i.e the min, avg, and max temperatures for each of the next 5 days for each of the 20 stations), we will need to train one GradientBoosted Tree for each entry in this vector. That is, we trained 300 different Gradient Boosted Trees one corresponding to each measurement, station, and day combination. The model class below implements this idea by using the MultiOutputRegression model from sklearn.

Note that we let each Gradient Boosting model use 20 regression Trees. Each Gradient Boosted tree was trained on the same covariates as the Regression-based Models. Like for the Regression-based models, we can again reuse the MetaPredictor model class.

Random Forest In addition to Gradient Boosted Trees, we also implemented a prediction model using Random Forest. Similar to Gradient Boosting, Random Forest is a Tree-based Ensemble model. However, unlike Gradient Boosting, Random Forests use Bagging to aggregate the prediction of its Regression Trees. The figure below visualizes a Random Forest model.



In addition, while Gradient Boosting can be used to make predictions when the response variable is a scalar, Random Forests can be trained and used for tasks when the response variable is a vector. To this end, we trained one Random Forest for each station, each of which outputs 15 predictions corresponding to the minimum, average, and maximum temperatures for the next 5 days. Thus, only a total of 20 Random Forest were trained. Each Random Forest used 100 Regression Trees of depth 5. In addition, the same covariates as in the Regression models and Gradient Boosting were also used for each of the Random Forest Models. The model class below formalizes this idea in code. Note that like the Regression-based models and Gradient Boosting, we can also use the Meta-Predictor here by passing in a Random Forest regression model into the constructor of the Meta-Predictor model class.

Model Selection and Evaluation

In order to select a model for deployment, we evaluated each of the models specified previously on a ten day evaluation window from Nov 30 - Dec 10 for the years of 2017, 2018, 2019, 2020, and 2021. More specifically, we simulated the exact same prediction process for 10 days between Nov 30 - Dec 10th across 5 different years. For each year and each model, we compute the average Mean Squared Error of that model's predictions for each of the 10 days. That is, for a given model, year, and day, the mean squared error was computed between the 300 temperature measurements predicted by the model and the true 300 temperature measurements corresponding to the min, avg, and max temperatures across all 20 stations for the next 5 days. This means, that for one particular year, each model had 10 MSE values, one per day in that year. The code below computes these MSEs for each model across all the years. The MSE values for the Previous Day Predictor are printed as an example of what the output looks like.

```
LASSO complete!
{2017: [45.402877891586385, 70.99900595688898, 114.64673000120425, 146.46203679559758, 154.808921679990
```

```
PrevDay complete!
Historic complete!
WA complete!
ARIMA complete!
OLS complete!
Ridge complete!
MLP complete!
```

Next, for each model and each year, we compute an average over the 10 MSEs to produce a single average MSE representing the performance of that model in that year. The code below is function that takes as input the dictionary of MSE values for a particular model, and outputs the average MSE per year for that model.

We apply the function above across the MSE dictionaries for each model to compute an average MSE for each model in each of the 5 years. The results have been summarized in the table below.

```
{2017: 113.1581879640379, 2018: 81.39315497797512, 2019: 97.44291151472791, 2020: 99.51453825096834, 20
```

Model	2017	2018	2019	2020	2021
Prev. Day	113.16	81.39	97.44	99.51	115.49
Historic	248.80	191.88	179	123	176
Weighted Avg.	133.27	87.54	104.96	80.22	113.01
ARIMA	168.76	92.86	108.16	124.9	146.57
OLS	108	65.7	66.9	56.6	85.2
LASSO	100.1	67.6	65.8	52.17	84.69
Ridge	104.5	65.2	67.1	54.1	84.3
MLP	111.15	80.32	67.14	61.17	105.18
Gradient Boosting	101.08	77.38	54.30	60.84	85.81
Random Forest	92.36	64.87	55.34	52.71	83.97

Based on the Table above, we find that the Random Forest model consistently outperfoms other models across all 5 years. This led us to deploy it in practice.

Conclusion

Although weather is caused by the atmospheric change that can, in principle, be described by laws of physics, we found that statistically modeling of the weather can be rather challenging. One key challenge is including all the atmospheric factors that determines the weather in our statistical model. Nevertheless, we found that the weather can be predicted with a reasonable accuracy even with a few macro factors using regression-based and ensemble models. With right set of features and large training data, it may be possible to predict temperature fairly accurately by just using highly expressive statistical models. Finally, combining statistical models with physics-based constraints is an exciting future direction for weather prediction.