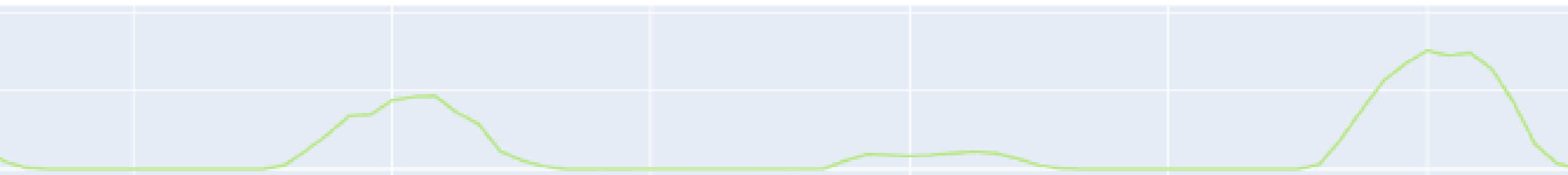
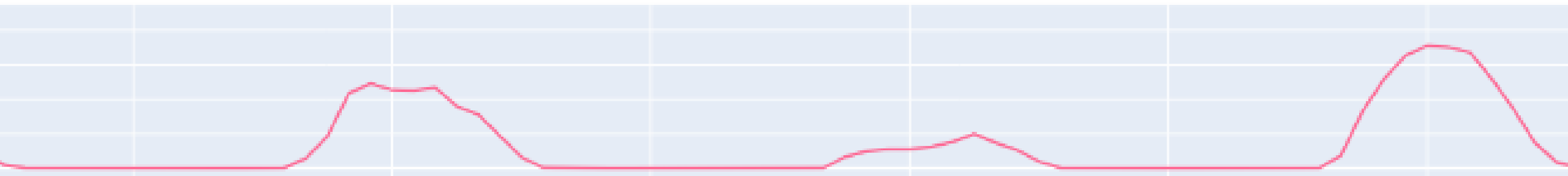
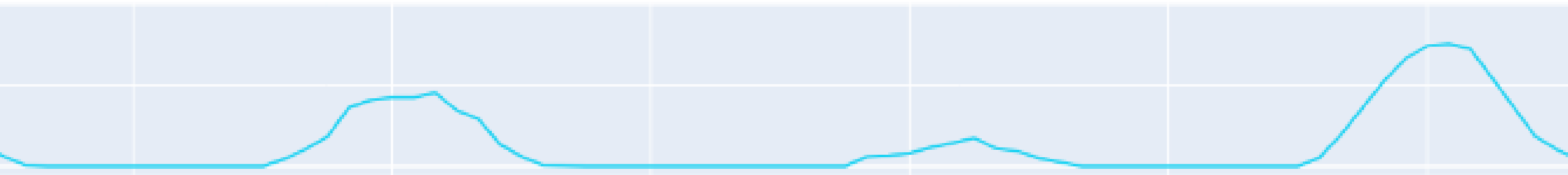


POSTECH

JBIG

OIBC



01

대회 설명



02

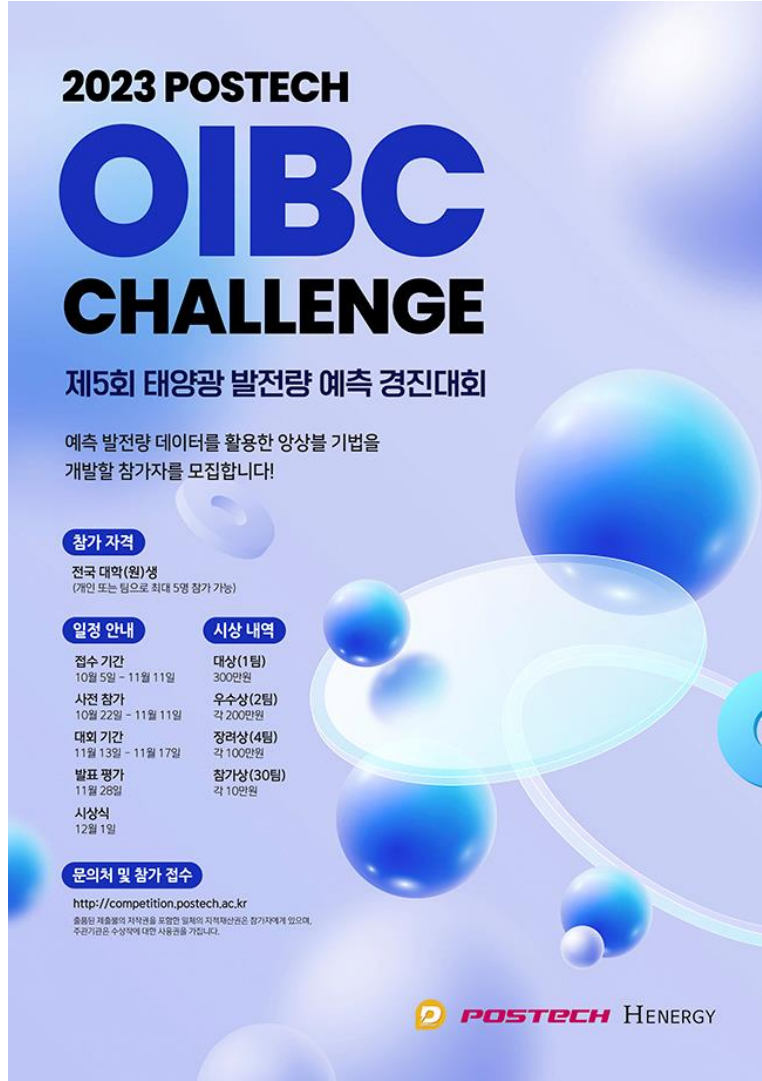
가설 설정 및 검증



03

결과 및 고찰



The poster for the 2023 POSTECH OIBC CHALLENGE features a light blue background with several large, translucent blue spheres of varying sizes. The text is in bold, dark blue and black fonts. At the top, it says '2023 POSTECH OIBC CHALLENGE' and '제5회 태양광 발전량 예측 경진대회'. Below this, it states '예측 발전량 데이터를 활용한 상상블 기법을 개발할 참가자를 모집합니다!'. There are three main sections: '참가 자격' (Eligibility), '일정 안내' (Schedule), and '시상 내역' (Prizes). The '참가 자격' section mentions '전국 대학(원)생' (Students nationwide). The '일정 안내' section lists dates for application, preliminary competition, main competition, and award ceremony. The '시상 내역' section lists prize categories and amounts. At the bottom, there is a URL 'http://competition.postech.ac.kr' and the POSTECH HENERGY logo.

2023 POSTECH
OIBC
CHALLENGE
제5회 태양광 발전량 예측 경진대회

예측 발전량 데이터를 활용한 상상블 기법을
개발할 참가자를 모집합니다!

참가 자격
전국 대학(원)생
(개인 또는 팀으로 최대 5명 참가 가능)

일정 안내
접수 기간
10월 5일 ~ 11월 11일
사전 참가
10월 22일 ~ 11월 11일
대회 기간
11월 13일 ~ 11월 17일
발표 평가
11월 28일
시상식
12월 1일

시상 내역
대상(1팀)
300만원
우수상(2팀)
각 200만원
장려상(4팀)
각 100만원
참가상(30팀)
각 10만원

문의처 및 참가 접수
<http://competition.postech.ac.kr>
출원한 제출물의 저작권을 포함한 일체의 지적재산권은 참가자에게 있으며,
주관기관은 수상작에 대한 사용권을 가집니다.

POSTECH HENERGY

대회 목적

13개의 기상변수와 5개의 모델 예측 발전량을 통하여 Amount를 예측하는 것

데이터 셋

(과거 데이터) 2022.06.19~2023.10.15

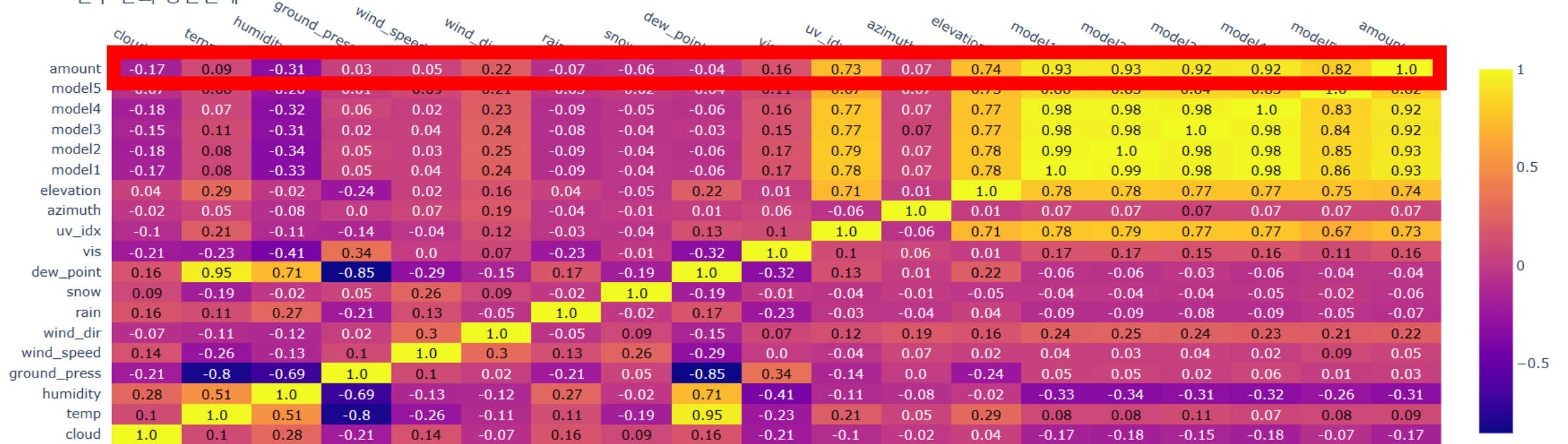
(사전 대회 기간 데이터) 2023.10.22~2023.11.11

특이사항

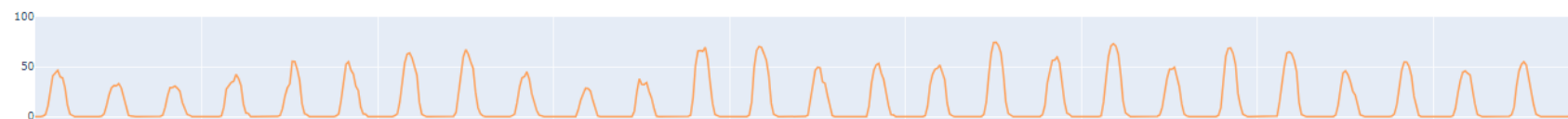
10시에 적합한 조건과 17시에 적합한 조건을 설정

데이터의 상관성 분석

변수 간의 상관관계



Model 1



Model 2



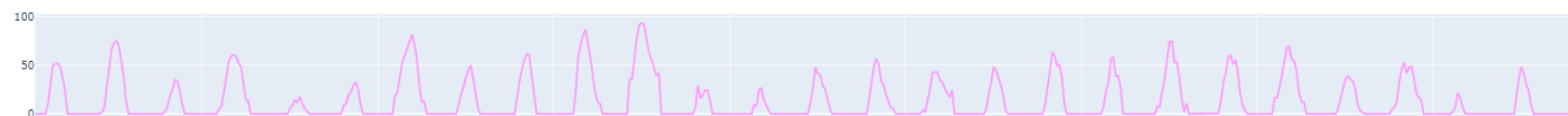
Model 3



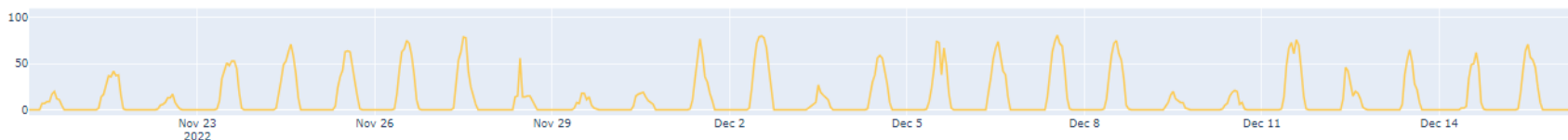
Model 4



Model 5

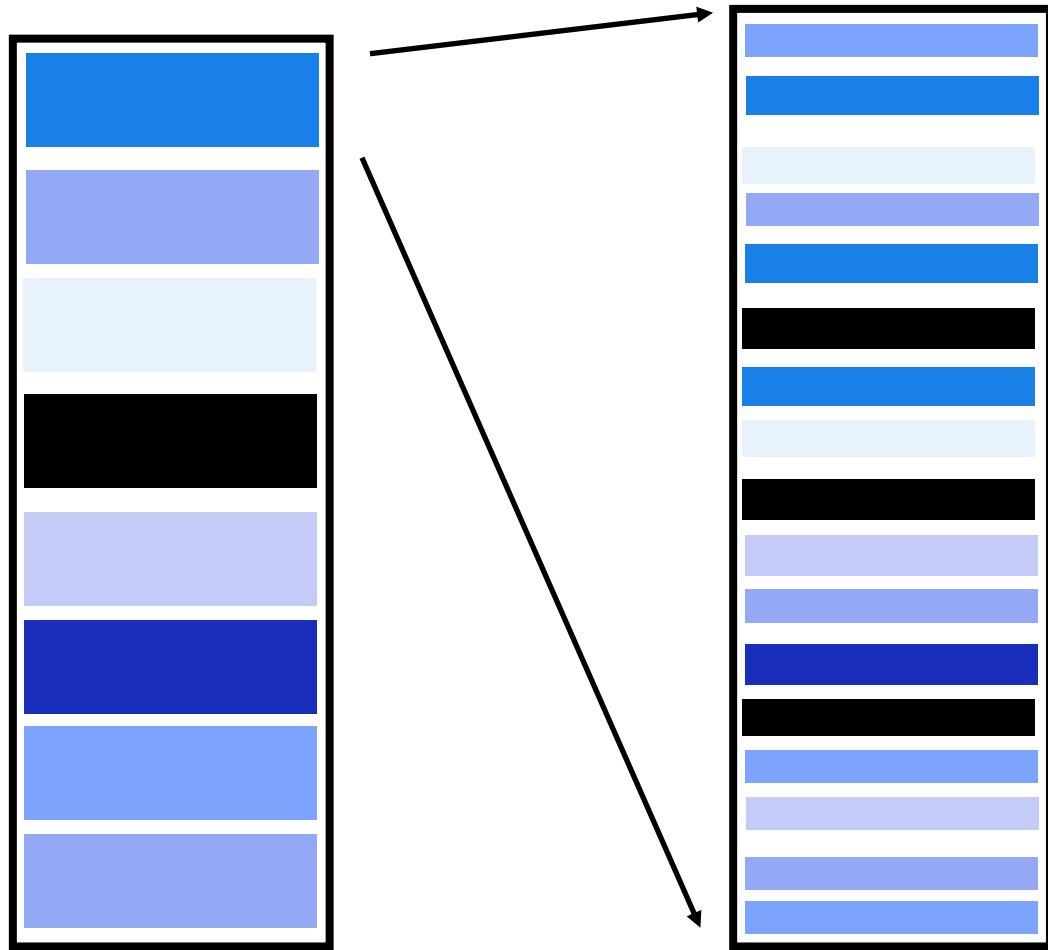


Amount



시간대별 모델의 예측값

각 시간대별 기상데이터와
유사한 N개 데이터 추출



N개의 데이터에서
최적 가중치 생성

최적 가중치를 각 모델별
예측값에 곱하여 더한 최
종 발전량 예측값

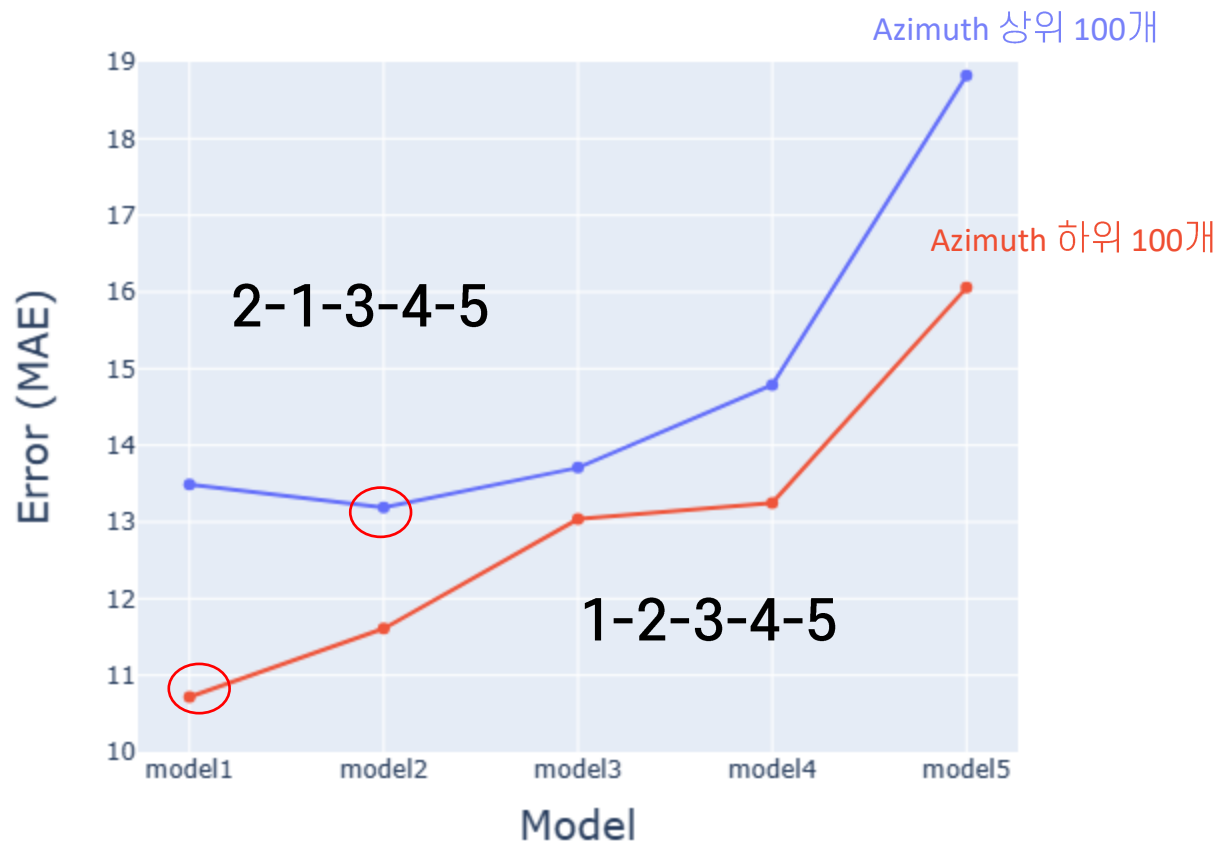
$$\text{pred} = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 = \sum_{i=1}^5 w_ix_i$$

Hypothesis

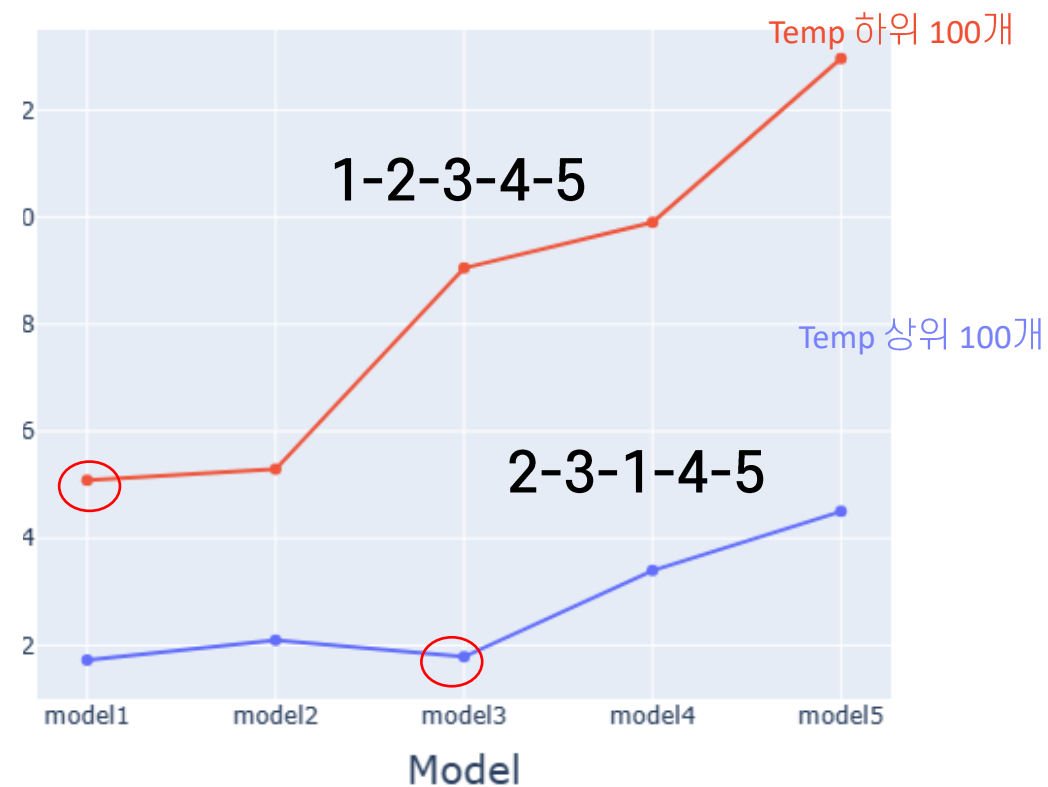
기상에 따라 최적 모델이 다르다?

1

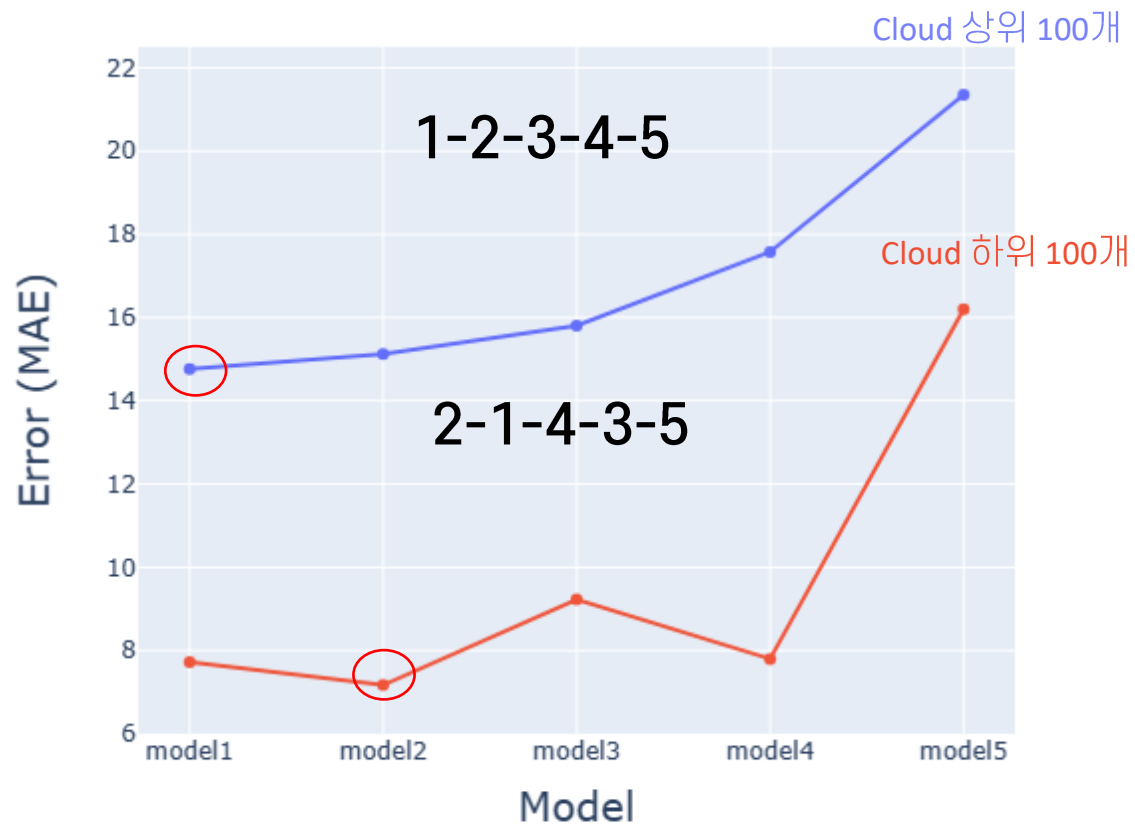
Azimuth Model별 평균 에러



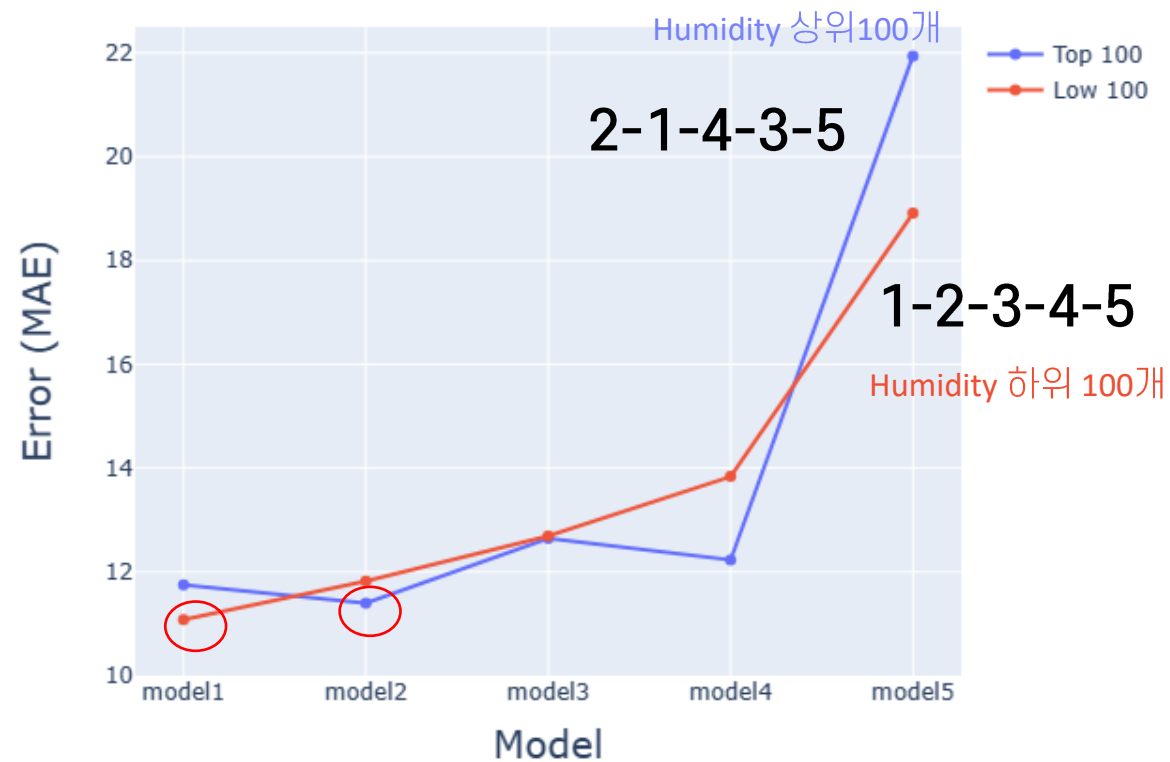
Temp Model별 평균 에러



Cloud Model별 평균 에러



Humidity Model별 평균 에러



Hypothesis

Minimize Function의 사용?

2



앙상블의 목적?

모델들을 적절히 조합하여 최고의 결과를 내는 것

||

각 모델에 대한 가중치를 어떻게 설정하느냐

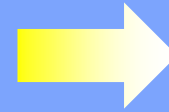
각각의 가중치는 0에서 1 사이이며, 가중치의 합은 1이다.(ex. 0.25 0.25 0.25 0.25)

가중치의 합이 1이어야 하는 이유 (Constraint)



모델들이 표현가능한 범위 내에서
예측을 하려고 했음(가중산술평균)

각각의 가중치가 0에서 1 사이여야 하는 이유 (Bound)



가중치의 절대값이 커지면
모델이 데이터에 과적합이 될 수 있음

scipy.optimize의 minimize

기상 지표

[과거와 현재의 차이?]

[과거와 현재의 차이?]

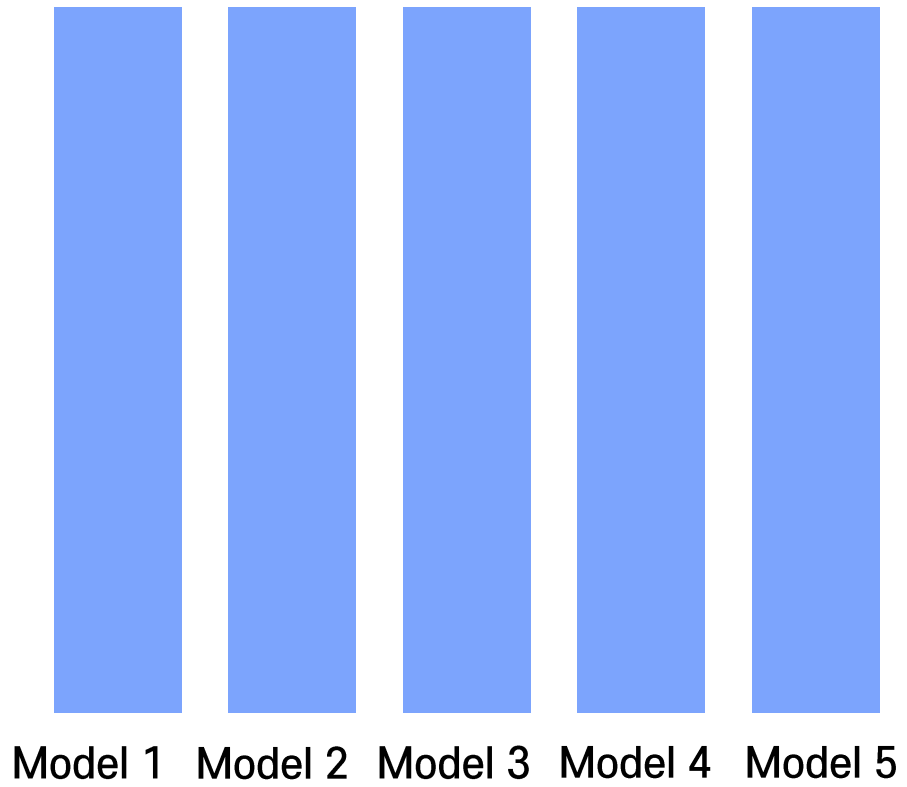


Hypothesis

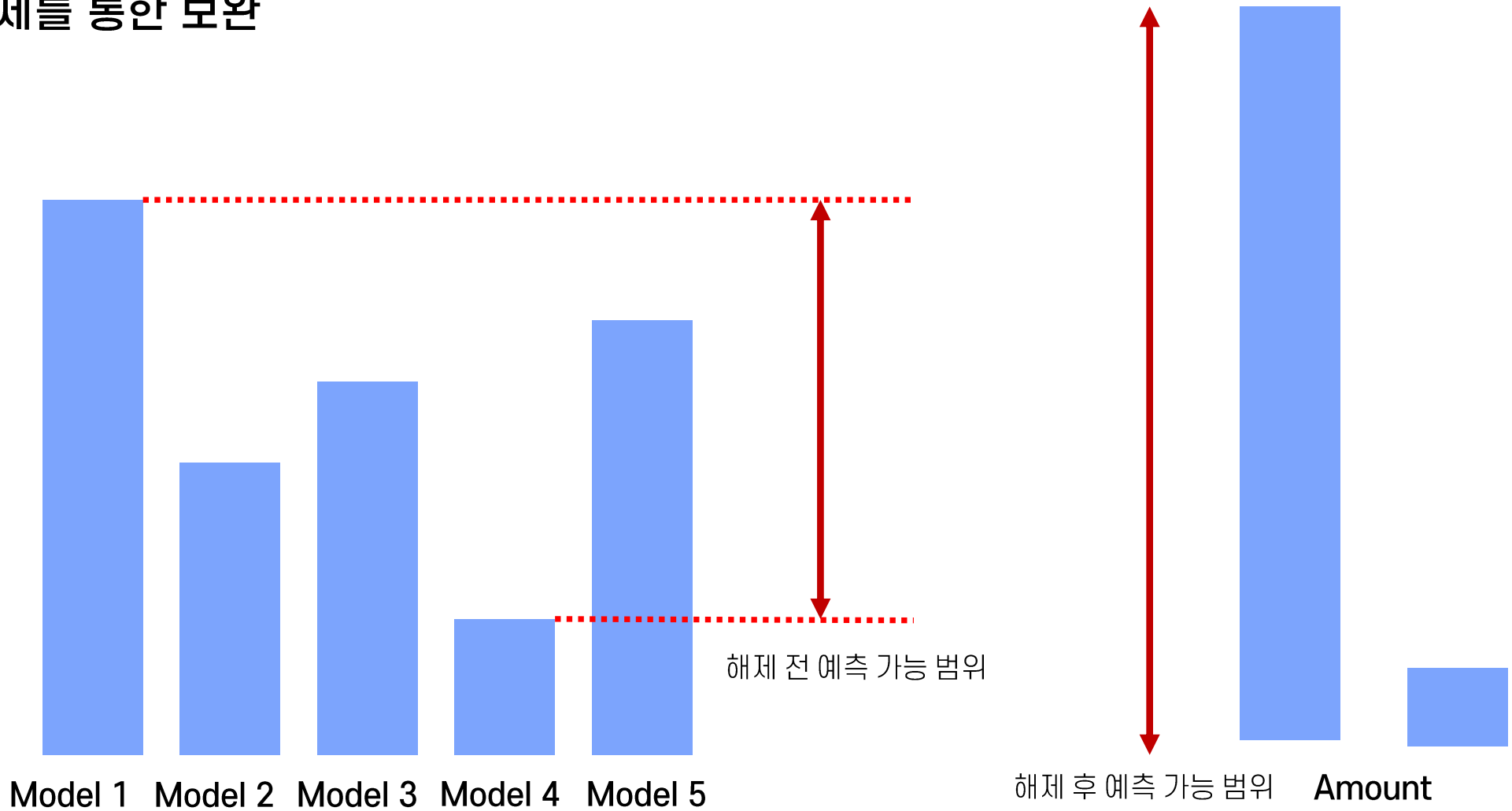
제약 조건의 효과성은?

3

Limit 해제를 통한 보완

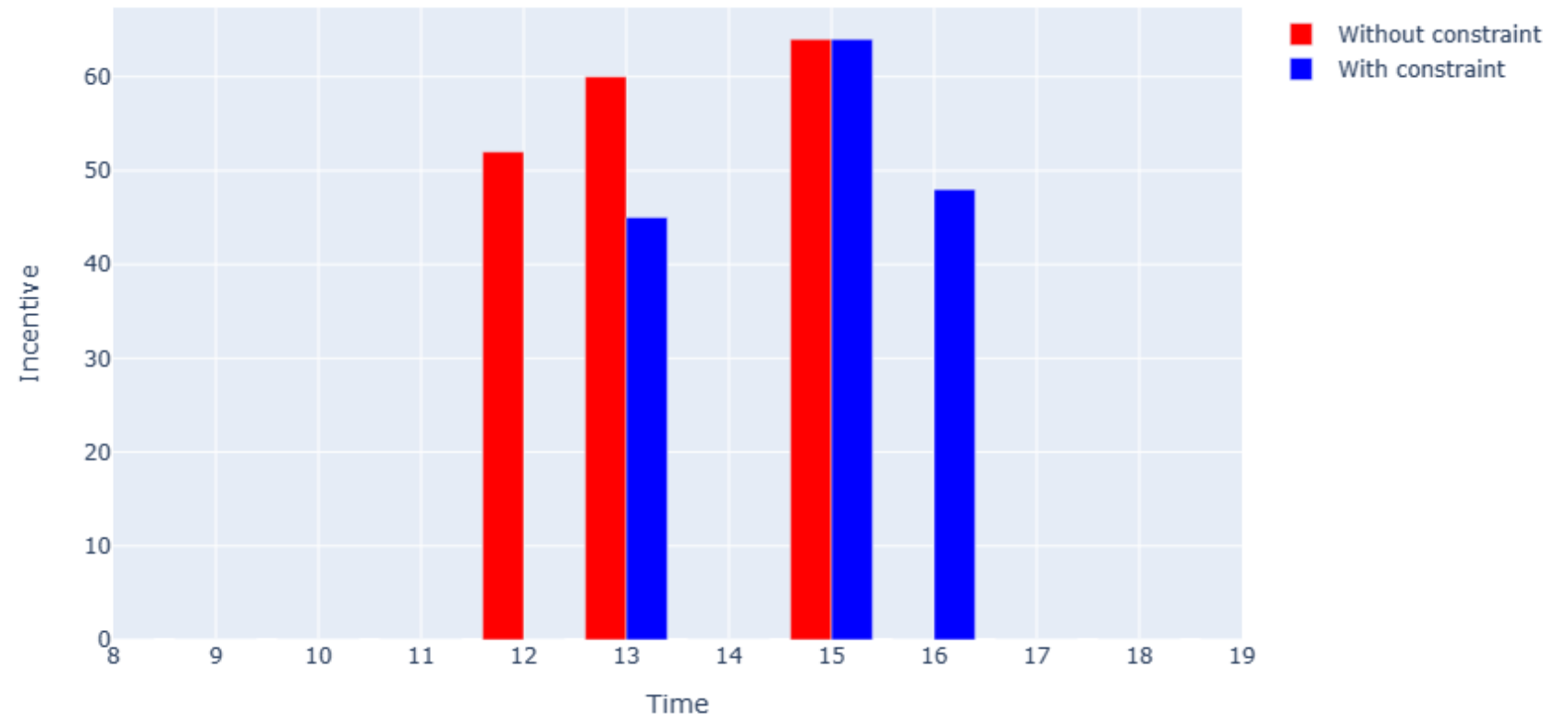


Limit 해제를 통한 보완

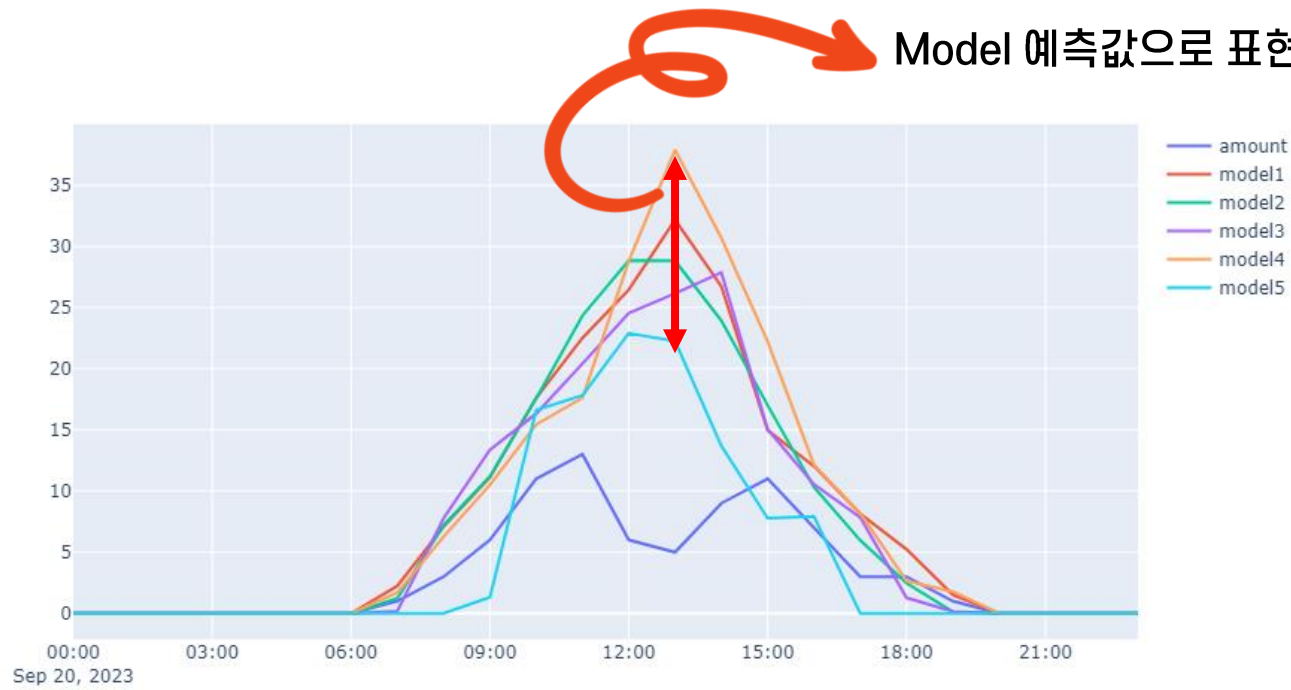


Limit 해제를 통한 보완

11월 17일 Constraint에 따른 Incentive 획득량



(예시) 2023.09.20 incentive 비교



	Bound 있음	Bound 없음
Constraint 있음	88	140
Constraint 없음	77	129

변수 중요도를 통하여 기상 변수 선별?

AutoGluon-Tabular(Auto ML)

AutoGluon v0.8 Cheat Sheet

Installation

AutoGluon (Gluon) requires pip > 1.4 (upgrade by `pip install -U pip`). More installation options: AutoGluon 0.7 supports Python 3.8 to 3.10. Installation is available for Linux, MacOS, and Windows.

```
pip install autogluon
```

Preparing Data

AutoGluon accepts DataFrames as inputs, where each row stores an example, while a column presents a feature. Here we use the `Simple-Titanic` dataset to demonstrate how to use AutoGluon.

```
import pandas as pd
train_data = pd.read_csv('titanic/train.csv')
```

From `autogluon.tabular` import `TabularDataset`

```
train_data = TabularDataset('titanic/train.csv')
```

It's also a Pandas DataFrame but with additional methods

Little data preprocessing, such as removing obvious non-predictive columns, is needed for AutoGluon.

```
train_data = train_data.drop(columns=['passengerid'])
```

Training

Train models to predict the values in the column "Survived". The training log will tell you how AutoGluon extracts features, selects, trains and ensembles models.

```
from autogluon.tabular import TabularPredictor
predictor = TabularPredictor(label='Survived', fit_kwargs={'data': train_data})
```

More options to construct a `TabularPredictor` instance

```
verbosity=1 # More training log
# The metric used to tune models. ALL available metrics, eval_metric='auc'.
```

More options for the fit method

```
# Limit the training time, in second time (default)
# Better model ensemble for a better accuracy, but longer training time. ALL available options.
presetter='best_quality'
# Use a separate dataset to tune models.
tuning_data=train_data
# Explore loss models. You can fully control the model search space. ALL available options.
hyperparameters='very_light'
# Ignore some models.
excluded_model_types=['NN', 'NN_TORCH']
```

Monitoring

Understand the contribution of each model

```
predictor.leaderboard()
```

Timing time

More options for `leaderboard()`

```
verbosity=1 # Recommended when using leaderboards
# Report metrics on a separate test dataset.
test_data=train_data
# Evaluate more metrics.
extra_metrics=['accuracy', 'log_loss']
```

Understand more about the trained models

```
predictor.fit_summary(show_plot=True)
```

You need to look to use `show_plot=True` (e.g. `pip install matplotlib`). It will generate performance vs. speed results in `SummaryOfModels.html`.

Understand the importance of each feature

```
predictor.feature_importance(test_data)
```

Importance score

Feature	Importance	Order	Value
Age	0.000000	0.000000	0.000000
Sex	0.000000	0.000000	0.000000
Survived	0.000000	0.000000	0.000000
Embarked	0.000000	0.000000	0.000000
Fare	0.000000	0.000000	0.000000
PassengerId	0.000000	0.000000	0.000000

Predicting

```
test_data = TabularDataset('test.csv')
# Predict for each row
predictor.predict(test_data)
# Return the class probabilities for class/floatation
predictor.predict_proba(test_data)
# Evaluate on test metrics. It needs test_data to have the label column.
predictor.evaluate(test_data)
```

AutoGluon predicts with the final ensemble model. You can also predict using an individual model.

```
# Get a list of existing models
models = predictor.get_model_names()
# Predict with the 1st model, both predict_proba and evaluate also accept the model argument.
predictor.predict(test_data, model=models[0])
```

Deploying

AutoGluon models are saved to disk automatically. You can check log to find where it saves, or get the path by `predictor.path`.

```
# Load saved model from disk.
predictor = TabularPredictor.load('autogluon/models/sg-20230228-004300')
```

If the inference speed matters, there are multiple ways to accelerate the speed. First, you can force all models in memory.

```
predictor.persist_models()
```

During training, you can use presets for the fit method optimized for fast inference (though may hurt model performance).

```
presetter='best_quality', 'lightest_size_for_deployment'
```

Alternatively, you can distill the ensemble into a single model.

```
# Get the list of names of the distilled models.
students = predictor.distill()
# Evaluate the 1st distilled model.
predictor.evaluate(test_data, model=students[0])
```

Results on Titanic: Accuracy 83.8% → 84.9%, evaluation time 60ms → 40ms. Now the distilled model even has a better accuracy.

- Click here for detailed Tabular tutorials.
- For data involving text and images, try out `AutoGluonCV`.
- Check the latest version of this cheat sheet at <https://autogluon.ai/tutorials/tabular/cheat-sheet.html>
- Any questions? Ask here
- Like what you see? Consider [donating](#) AutoGluon on GitHub and following us on Twitter to get notified of the latest updates!

Framework	Wins	Losses	Failures	Champion	Avg. Rank	Avg. Rescaled Loss	Avg. Time (min)
AutoGluon	-	-	1	23	1.8438	0.1385	201
H2O AutoML	4	26	8	2	3.1250	0.2447	220
TPOT	6	27	5	5	3.3750	0.2034	235
GCP-Tables	5	20	14	4	3.7500	0.3336	195
auto-sklearn	6	27	6	3	3.8125	0.3197	240
Auto-WEKA	4	28	6	1	5.0938	0.8001	244

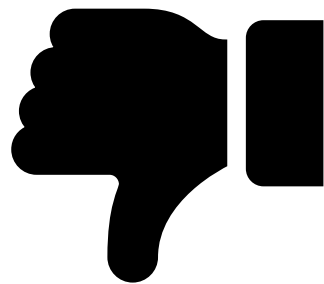
Framework	Wins	Losses	Failures	Champion	Avg. Rank	Avg. Percentile	Avg. Time (min)
AutoGluon	-	-	0	7	1.7143	0.7041	202
GCP-Tables	3	7	1	3	2.2857	0.6281	222
H2O AutoML	1	7	3	0	3.4286	0.5129	227
TPOT	1	9	1	0	3.7143	0.4711	380
auto-sklearn	3	8	0	1	3.8571	0.4819	240
Auto-WEKA	0	10	1	0	6.0000	0.2056	221

	importance	stddev	p_value	n	p99_high	p99_low
uv_idx	6.755734	0.266948	2.919400e-07	5	7.305383	6.206084
elevation	6.018258	0.194062	1.295570e-07	5	6.417834	5.618681
azimuth	1.799953	0.101760	1.220661e-06	5	2.009478	1.590428
cloud	1.403375	0.167020	2.362671e-05	5	1.747272	1.059479
humidity	1.402972	0.109507	4.418044e-06	5	1.628448	1.177496
rain	0.967076	0.119125	2.707791e-05	5	1.212356	0.721796
temp	0.458720	0.064058	4.447133e-05	5	0.590616	0.326823
dew_point	0.298858	0.077916	5.075197e-04	5	0.459287	0.138428
vis	0.195553	0.042976	2.627641e-04	5	0.284040	-0.107065
wind_dir	0.112965	0.099038	3.163521e-02	5	0.316886	-0.090955
wind_speed	0.043546	0.070728	1.203198e-01	5	0.189175	-0.102084
ground_press	-0.009937	0.072647	3.874832e-01	5	0.159517	- 0.139643
snow	-0.000000	0.000000	5.000000e-01	5	0.000000	-0.000000

Hypothesis 5

비슷한 날씨 조건에서의 비교?

구간을 나누지 않고 일괄 적용한 경우



9.137

EV/SQR



8.792

최대 발전량을 기준으로 구간을 나눈 경우

전체 데이터에서 최소의 MAE를 만족하는 조합 찾기

```
rank_columns = ['uv_idx_rank', 'elevation_rank', 'azimuth_rank', 'cloud_rank', 'humidity_rank']
```

```
...
100 개 데이터로 ('uv_idx_rank', 'elevation_rank', 'cloud_rank') 를 학습한 에러 = 11.415992762541132
-----
100 개 데이터로 ('uv_idx_rank', 'elevation_rank', 'humidity_rank') 를 학습한 에러 = 11.731137211639194
-----
100 개 데이터로 ('uv_idx_rank', 'cloud_rank', 'humidity_rank') 를 학습한 에러 = 9.13671450089481
...
-----
100 개 데이터로 ('uv_idx_rank', 'elevation_rank', 'cloud_rank', 'humidity_rank') 를 학습한 에러 = 9.38257010537417
-----
100 개 데이터로 ('azimuth_rank', 'elevation_rank', 'cloud_rank', 'humidity_rank') 를 학습한 에러 = 12.397549981707867
-----
100 개 데이터로 ('uv_idx_rank', 'azimuth_rank', 'elevation_rank', 'cloud_rank', 'humidity_rank') 를 학습한 에러 = 10.671785394651813
```

31개의 조합 중 최소의 MAE Error를 만족하는 조합

('100', " ('uv_idx_rank', 'elevation_rank', 'cloud_rank', 'humidity_rank') ")

구간을 나눈 데이터에서 최소의 MAE를 만족하는 조합 찾기(최대 발전량 부근)

```
rank_columns = ['uv_idx_rank', 'elevation_rank', 'azimuth_rank', 'cloud_rank', 'humidity_rank']
```

```
...
100 개 데이터로 ('azimuth_rank', 'cloud_rank') 를 학습한 에러 = 18.683330502128616
-----
100 개 데이터로 ('azimuth_rank', 'humidity_rank') 를 학습한 에러 = 20.58648330187275
-----
...
100 개 데이터로 ('uv_idx_rank', 'elevation_rank', 'cloud_rank') 를 학습한 에러 = 17.824633579876703
-----
100 개 데이터로 ('uv_idx_rank', 'elevation_rank', 'humidity_rank') 를 학습한 에러 = 19.57289507366416
-----
100 개 데이터로 ('uv_idx_rank', 'cloud_rank', 'humidity_rank') 를 학습한 에러 = 14.466763761527043
-----
100 개 데이터로 ('azimuth_rank', 'elevation_rank', 'cloud_rank') 를 학습한 에러 = 19.799828665829736
...
```

31개의 조합 중 최소의 MAE Error를 만족하는 조합
('100', " ('uv_idx_rank', 'cloud_rank', 'humidity_rank') ")

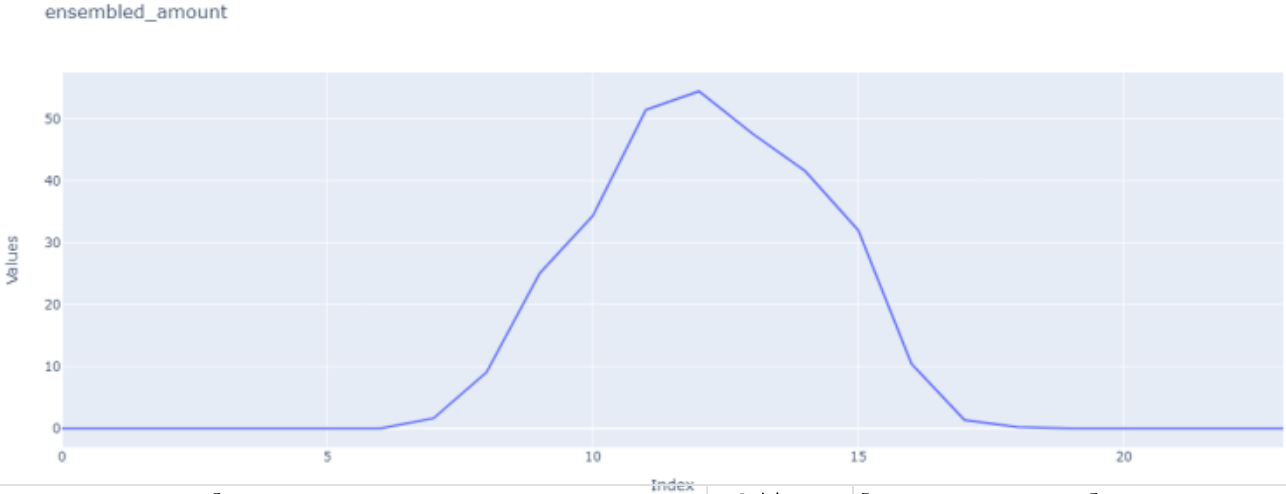
구간을 나눈 데이터에서 최소의 MAE를 만족하는 조합 찾기(최소 발전량 부근)

```
rank_columns = ['uv_idx_rank', 'elevation_rank', 'azimuth_rank', 'cloud_rank', 'humidity_rank']
```

```
...
-----
100 개 데이터로 ('uv_idx_rank', 'elevation_rank') 를 학습한 에러 = 3.426966984432953
-----
100 개 데이터로 ('uv_idx_rank', 'cloud_rank') 를 학습한 에러 = 3.1177611152377196
-----
100 개 데이터로 ('uv_idx_rank', 'humidity_rank') 를 학습한 에러 = 4.3638940552544
-----
100 개 데이터로 ('azimuth_rank', 'elevation_rank') 를 학습한 에러 = 3.6551060869654095
-----
...
-----
100 개 데이터로 ('azimuth_rank', 'elevation_rank', 'cloud_rank', 'humidity_rank') 를 학습한 에러 = 4.028714200725035
-----
100 개 데이터로 ('uv_idx_rank', 'azimuth_rank', 'elevation_rank', 'cloud_rank', 'humidity_rank') 를 학습한 에러 = 3.4992402333146178
```

31개의 조합 중 최소의 MAE Error를 만족하는 조합
('100', "('uv_idx_rank', 'cloud_rank') ")

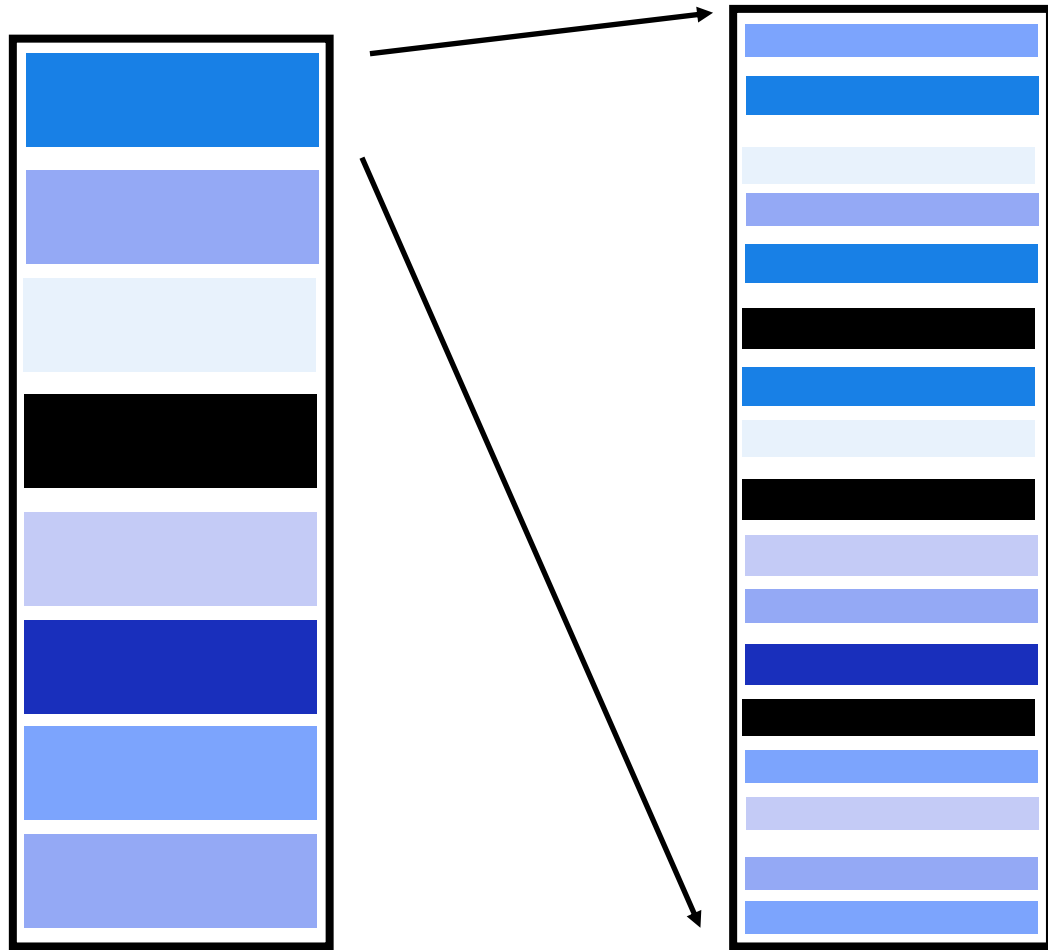
시간대별 가중치와 결과값



1시	[0.2 0.2 0.2 0.2 0.2]	13시	[0.99 0. 0. 0. 0.]
2시	[0.2 0.2 0.2 0.2 0.2]	14시	[0.99 0. 0. 0.0 0.01]
3시	[0.2 0.2 0.2 0.2 0.2]	15시	[0.54 0.28 0.06 0.08 0.03]
4시	[0.2 0.2 0.2 0.2 0.2]	16시	[0.62 0.08 0. 0.3 0.]
5시	[0.2 0.2 0.2 0.2 0.2]	17시	[0.24 0.25 0.23 0.28 0.002]
6시	[0.2 0.2 0.2 0.2 0.2]	18시	[0.23 0.18 0.57 0. 0.02]
7시	[0.29 0. 0. 0.714 0.]	19시	[0.31 0.06 0.50 0.09 0.03]
8시	[0.26 0.25 0.27 0.22 0.001]	20시	[0.27 0.001 0.73 0.0001 0.]
9시	[0.26 0.16 0.38 0.20 0.]	21시	[0.2 0.2 0.2 0.2 0.2]
10시	[0.39 0.42 0.01 0.18 0.0003]	22시	[0.2 0.2 0.2 0.2 0.2]
11시	[0.29 0.15 0.22 0.34 0.0005]	23시	[0.2 0.2 0.2 0.2 0.2]
12시	[0.99 0. 0.0034 0. 0.]	24시	[0.2 0.2 0.2 0.2 0.2]

시간대별 모델의 예측값

각 시간대별 기상데이터와
유사한 N개 데이터 추출



N개의 데이터에서
최적 가중치 생성



최적 가중치를 각 모델별
예측값에 곱하여 더한 최
종 발전량 예측값

$$\text{pred} = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 = \sum_{i=1}^5 w_i x_i$$

선형회귀, 랜덤포레스트를 사용한다면?

과거데이터 중 예측할 기상데이터와
가장 유사한 기상데이터를 추출하여 이에 최적화



급작스러운 기상변화에 민첩하고 정확하게 예측 가능
GPU자원이 필요하지 않고 적은 메모리로도 쉽게 연산이 가능

감사합니다.