**우주전파재난 AI 경진대회 개발모델 개요서**

**□ 참가팀**

|  |  |
| --- | --- |
| **팀명** | AJAI |
| **팀원 성명** | 남기현, 전준서(총 2명) |
| **소속** | 아주대학교 전자공학과 |

**□ 모델 개발 개요서**

**1. 데이터 셋 구성**

1) 주어진 데이터 셋

- ACE, DSCOVR 위성 데이터의 태양풍 데이터

- 국내 지자기 관측소 3곳의 지구자기장 관측 데이터

“Kp는 미국 NOAA SWPC(Space Weather Prediction Center)에서 위도 44 – 60도 사이의 ***8개 지자기 관측소***에서의 구한 K 지수를 통합하여 산출하고 있다.”[1]

우주전파센터 홈페이지의 Kp지수 관련 내용에서 “8개 지자기 관측소” 를 가지고 Kp지수를 계산한다고 명시되어 있으므로 ACE, DSCOVR 위성 데이터의 태양풍 데이터는 사용하지 않고 국내 지자기 관측소 3곳의 지구자기장 관측 데이터만을 사용

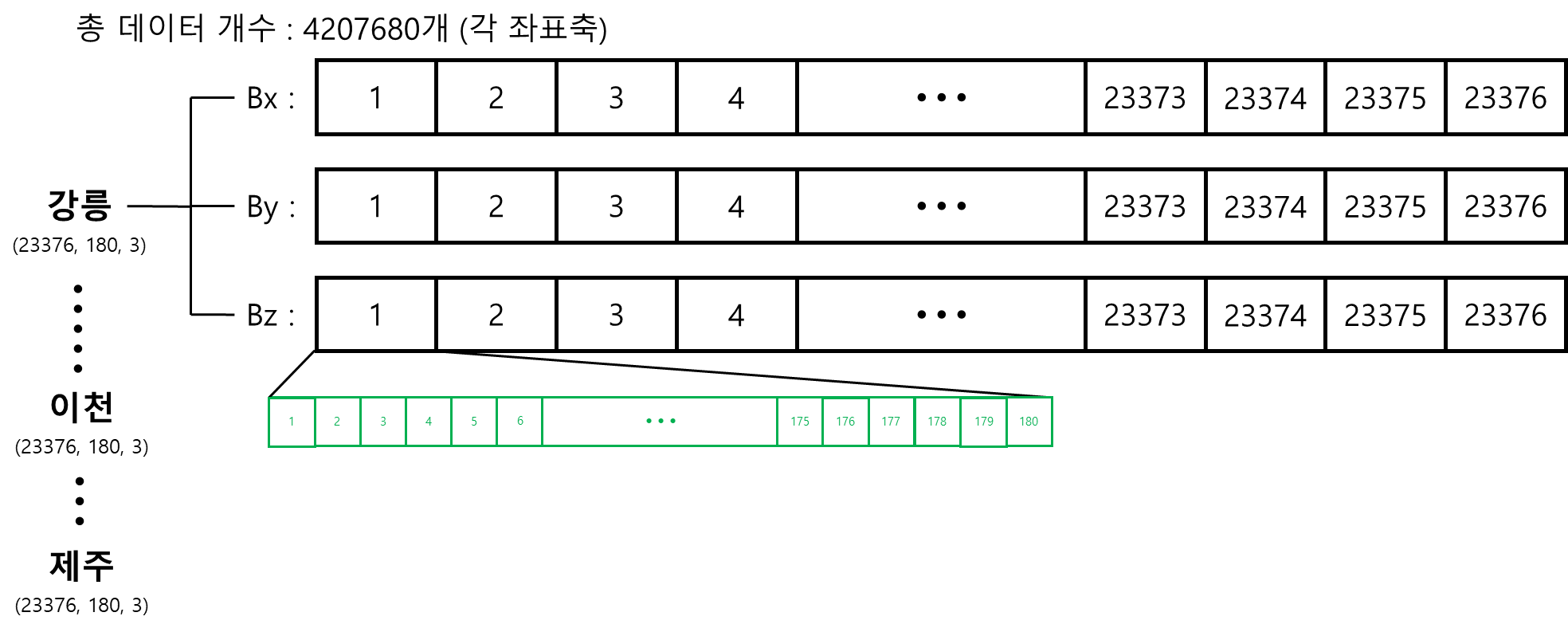
2) 데이터 셋 전처리

2) - 1. 데이터 보간

|  |
| --- |
| GGG\_df[['X','Y','Z']] = GGG\_df[['X','Y','Z']].interpolate(method='linear', limit=5)  GGI\_df[['X','Y','Z']] = GGI\_df[['X','Y','Z']].interpolate(method='linear', limit=5)  GGJ\_df[['X','Y','Z']] = GGJ\_df[['X','Y','Z']].interpolate(method='linear', limit=5) |

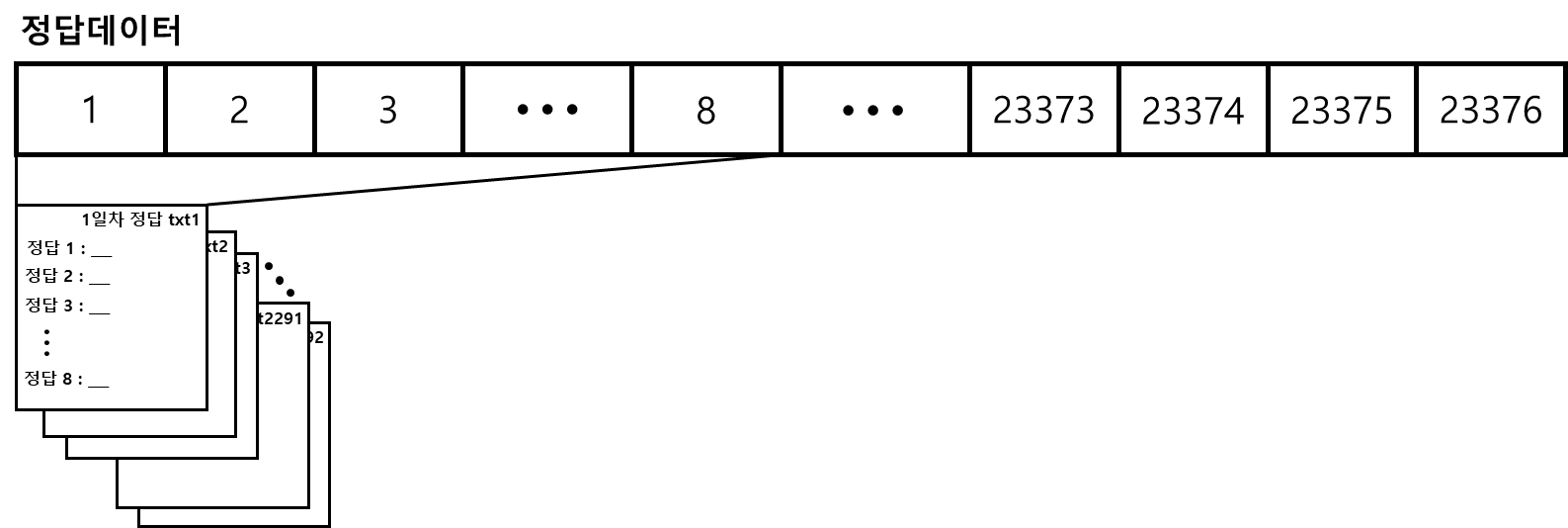
결측치가 있는 부분에 선형보간을 적용함.

(단, 연속적으로 매우 길게 결측치가 형성되어 있는 경우 선형 보간을 할 시 오차가 커지므로 threshold를 5로 지정함.)

2) - 2. 입력데이터 셋

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 지역 | 데이터 | 데이터형 | 변환 | 데이터형 |
| 강릉 | Bx | 4207680 | (23376, 180, 3) |
| By | 4207680 |
| Bz | 4207680 |
| 이천 | Bx | 4207680 | (23376, 180, 3) |
| By | 4207680 |
| Bz | 4207680 |
| 제주 | Bx | 4207680 | (23376, 180, 3) |
| By | 4207680 |
| Bz | 4207680 |

2) - 3. 출력 데이터 셋



|  |  |  |  |
| --- | --- | --- | --- |
| 데이터 | .txt | 변환 | 데이터형 |
| 국내 지자기  교란지수 | 8개/day씩 총 2922개 | (23376, 1) |

2) - 4. 결측치 처리

보간을 threshold=5로 진행하였으므로 여전히 결측치가 존재함. 따라서 22376개의 데이터 셋을 모두 검사하여 결측치가 존재하는 데이터는 모두 삭제함. (단, 삭제시 정답 데이터와 일대일 매핑하여 함께 삭제해야 함.)

|  |
| --- |
| def delete\_nan(X, y):  delete\_idx = []  for idx, X\_ in enumerate(X):  if np.isnan(X\_).sum() != 0:  delete\_idx.append(idx)    return np.delete(X,delete\_idx, axis = 0), to\_categorical(np.delete(y,delete\_idx, axis = 0))  GI\_X = np.concatenate([GGG\_X, GGI\_X], axis=2)  GJ\_X = np.concatenate([GGG\_X, GGJ\_X], axis=2)  IJ\_X = np.concatenate([GGI\_X, GGJ\_X], axis=2)  GIJ\_X = np.concatenate([GGG\_X, GGI\_X, GGJ\_X], axis=2)  # 조합 X  GGG\_X, GGG\_y = delete\_nan(GGG\_X, y)  GGI\_X, GGI\_y = delete\_nan(GGI\_X, y)  GGJ\_X, GGJ\_y = delete\_nan(GGJ\_X, y)  # 조합 O  GI\_X, GI\_y = delete\_nan(GI\_X, y)  GJ\_X, GJ\_y = delete\_nan(GJ\_X, y)  IJ\_X, IJ\_y = delete\_nan(IJ\_X, y)  GIJ\_X, GIJ\_y = delete\_nan(GIJ\_X, y) |

입력 데이터 셋의 변화 : (22376, 180, 3) -> (22376-α, 180, 3)

2) - 5. 데이터 feature 추출

A.

**위키피디아의 Kp 지수에 계산에 관한 설명**

***“The maximum positive and negative deviations during the 3 hour period are added together to determine the total maximum fluctuation.”*[2]**

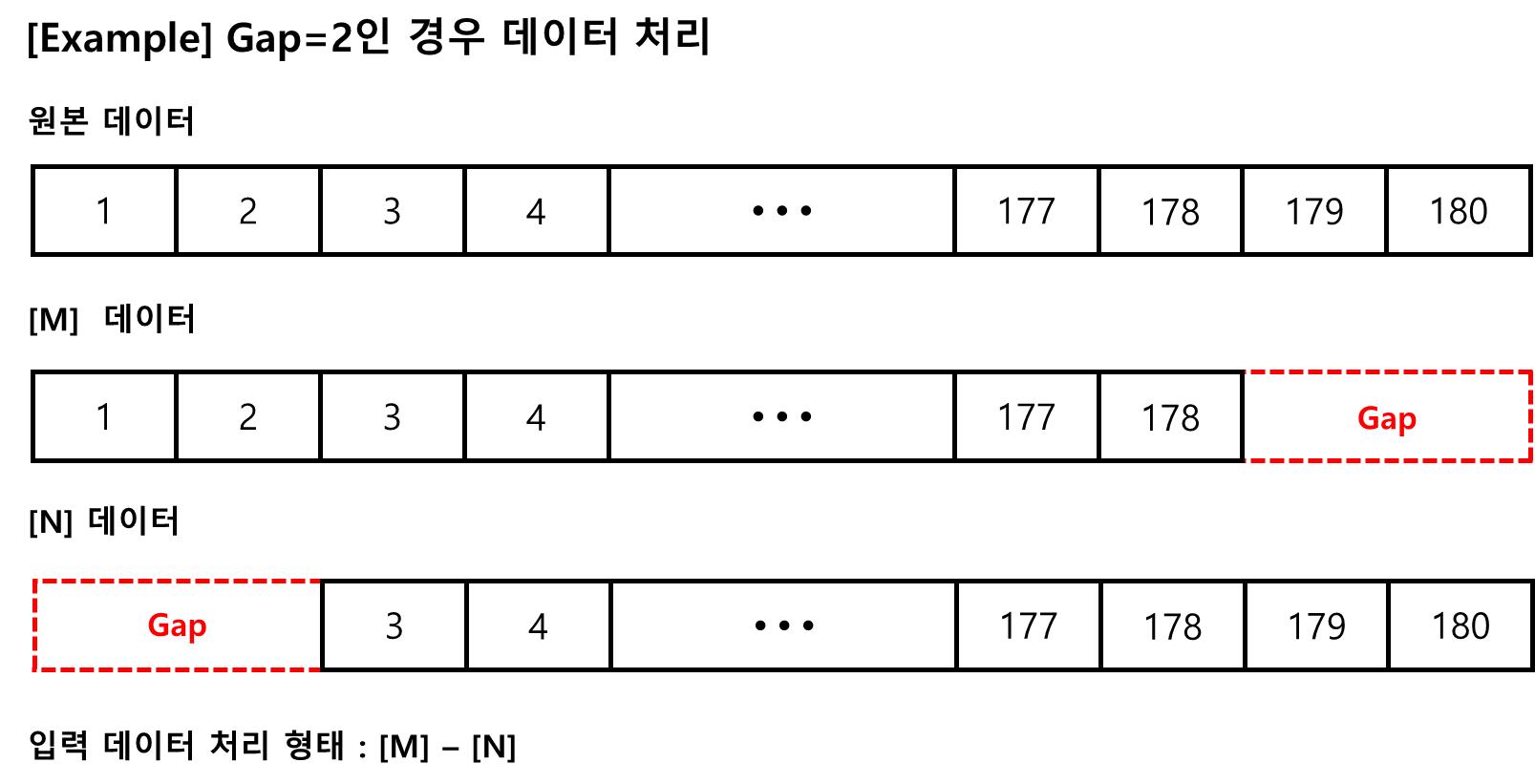
**구글 번역**

**총 최대 변동을 결정하기 위해 3시간 동안의 최대 양수 및 음수 편차를 합산합니다.**

“변동” 이라는 단어에 초점을 맞춰 변동은 곧 변화율을 의미 180개의 시계열 데이터를 미분하기로 결정.

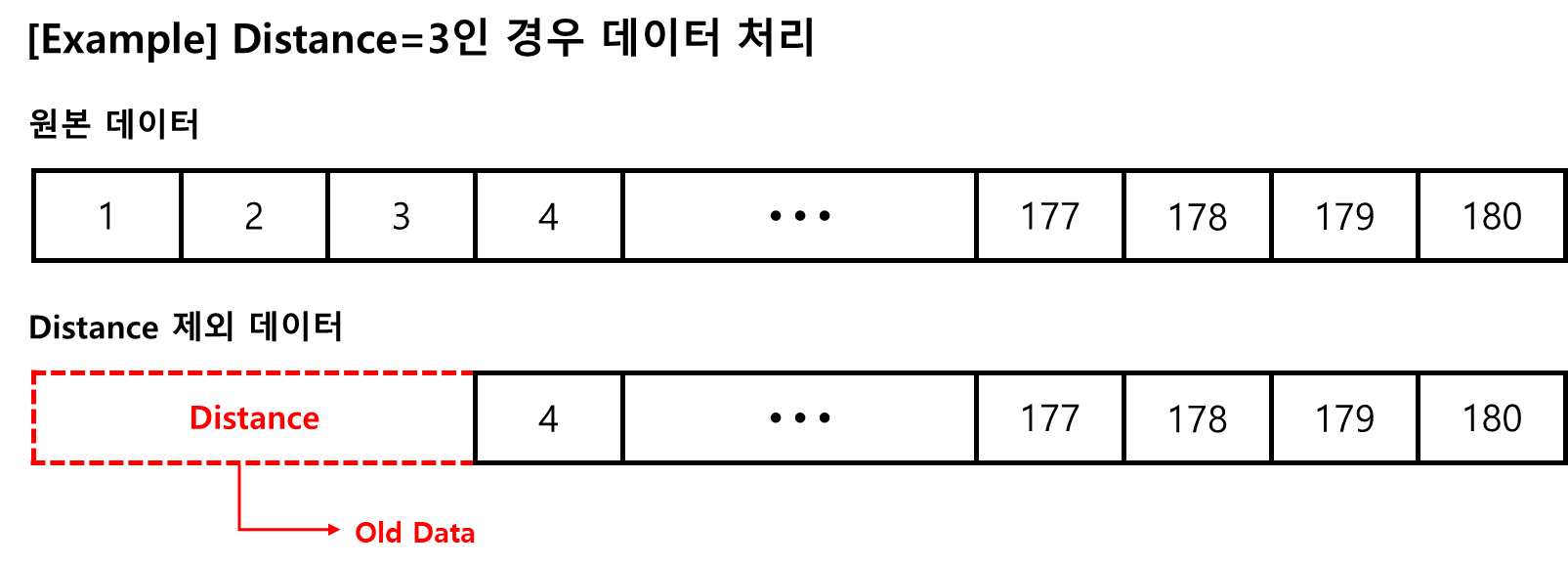
현재의 시점과 n분 뒤의 시점을 뺌

(미분에 정의에 의하면 간격인 h를 나눠줘야 하지만 모두 동일한 간격으로 설정되어 있으므로 나누지 않아도 무방함.) **=> gap 변수**



B.

180분의 데이터를 모두 사용하는 것과 시점이 많이 흐른 데이터를 없애는 것에 대해 어떤 것이 더 효율적인지 확인. **=> distance 변수**



A, B를 고려

|  |
| --- |
| G\_diff\_X = GGG\_X[:,0:180-gap-distance,:] - GGG\_X[:,gap:180-distance,:]  I\_diff\_X = GGI\_X[:,0:180-gap-distance,:] - GGI\_X[:,gap:180-distance,:]  J\_diff\_X = GGJ\_X[:,0:180-gap-distance,:] - GGJ\_X[:,gap:180-distance,:]  GI\_diff\_X = GI\_X[:,0:180-gap-distance,:] - GI\_X[:,gap:180-distance,:]  GJ\_diff\_X = GJ\_X[:,0:180-gap-distance,:] - GJ\_X[:,gap:180-distance,:]  IJ\_diff\_X = IJ\_X[:,0:180-gap-distance,:] - IJ\_X[:,gap:180-distance,:]  GIJ\_diff\_X = GIJ\_X[:,0:180-gap-distance,:] - GIJ\_X[:,gap:180-distance,:] |

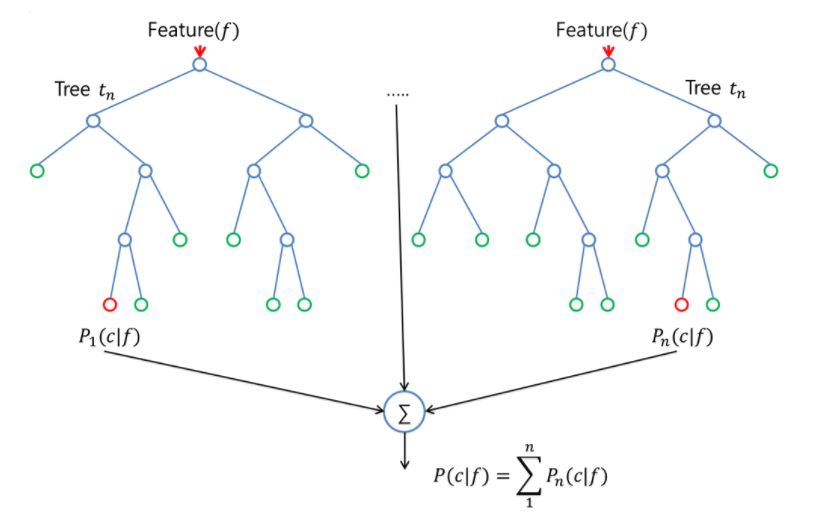
C. 평균, 최대, 최소, 표준편차 추출

distance와 gap을 고려하여 전처리한 이후 180개의 시계열 데이터에 대해 평균, 최대, 최소, 표준편차를 추출

|  |
| --- |
| def get\_feature(X):  feature = []  for sample in X:  tmp = sample[:,:].max(axis=0).tolist() + sample[:,:].min(axis=0).tolist() + sample[:,:].mean(axis=0).tolist() + sample[:,:].std(axis=0).tolist()  feature.append(tmp)  return np.array(feature)  G\_diff\_feature\_X = get\_feature(G\_diff\_X)  I\_diff\_feature\_X = get\_feature(I\_diff\_X)  J\_diff\_feature\_X = get\_feature(J\_diff\_X)  GI\_diff\_feature\_X = get\_feature(GI\_diff\_X)  GJ\_diff\_feature\_X = get\_feature(GJ\_diff\_X)  IJ\_diff\_feature\_X = get\_feature(IJ\_diff\_X)  GIJ\_diff\_feature\_X = get\_feature(GIJ\_diff\_X) |

**2. 적용 알고리즘**

2.1 적용 머신러닝 기법(Random Forest)



케라스에서 제공하는 Random Forest를 사용하였으며 파라미터는 default값을 사용함

2.2 입력 조합 설정

① 강릉(Bx, By, Bz)

② 이천(Bx, By, Bz)

③ 제주(Bx, By, Bz)

④ 강릉(Bx, By, Bz) + 이천(Bx, By, Bz)

⑤ 강릉(Bx, By, Bz) + 제주(Bx, By, Bz)

⑥ 이천(Bx, By, Bz) + 제주(Bx, By, Bz)

⑦ 강릉(Bx, By, Bz) + 이천(Bx, By, Bz) + 제주(Bx, By, Bz)

**총 7개의 조합이 나올 수 있으며 해당 조합에 대해 모두 테스트**

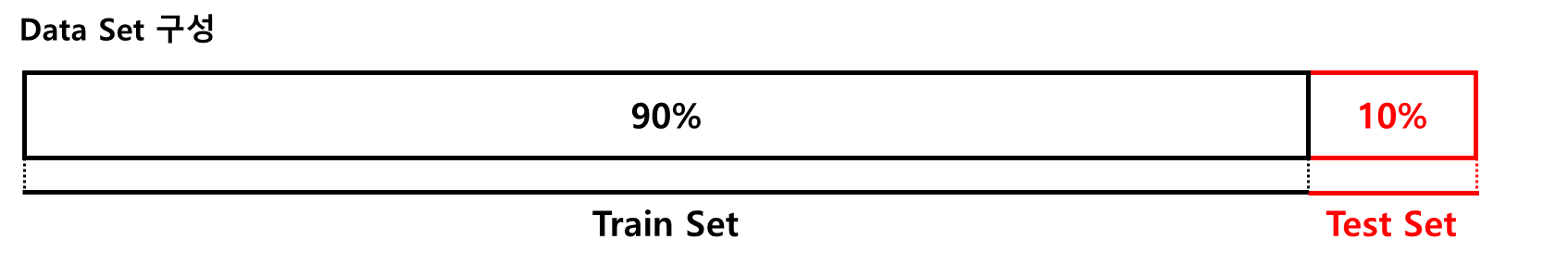
**3. 모델 구성**

케라스에서 제공하는 Random Forest 적용

|  |
| --- |
| def test\_rf(X, y):  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1, random\_state=42)    rf = RandomForestClassifier(random\_state=0)  rf.fit(X\_train, y\_train.argmax(axis=1))    pred = rf.predict(X\_test)    return rf, accuracy\_score(y\_test.argmax(axis=1), pred)  rf = []  acc = []  for idx, name in enumerate(total\_data.keys()):  rf\_, acc\_ = test\_rf(total\_data[name][0], total\_data[name][1])  rf.append(rf\_)  acc.append(acc\_)  print("({}) 조합 {} Accuracy : {}%".format(idx, name, round(acc\_, 4)) |

**4. 모델 정확도(accuracy)**

데이터 셋을 train 셋과 test 셋으로 나누어 train 셋으로 학습한 뒤 test 셋에 대한 Accuracy 측정



gap = 0 부터 48까지 증가시키며 7가지 입력 데이터 조합에 대해 test set Accuracy 측정

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Gap=1 | | Gap=2 | | Gap=3 | | Gap=4 | | Gap=5 | |
| ACC1 | 0.67167 | ACC1 | 0.69031 | ACC1 | 0.70168 | ACC1 | 0.71669 | ACC1 | 0.73169 |
| ACC2 | 0.66424 | ACC2 | 0.70068 | ACC2 | 0.71526 | ACC2 | 0.72692 | ACC2 | 0.74344 |
| ACC3 | 0.68938 | ACC3 | 0.71778 | ACC3 | 0.72564 | ACC3 | 0.73744 | ACC3 | 0.73875 |
| ACC4 | 0.66477 | ACC4 | 0.69684 | ACC4 | 0.69891 | ACC4 | 0.72944 | ACC4 | 0.73616 |
| ACC5 | 0.69531 | ACC5 | 0.71575 | ACC5 | 0.72782 | ACC5 | 0.74454 | ACC5 | 0.75151 |
| ACC6 | 0.68983 | ACC6 | 0.7067 | ACC6 | 0.71662 | ACC6 | 0.72705 | ACC6 | 0.74491 |
| ACC7 | 0.69873 | ACC7 | 0.73044 | ACC7 | 0.73414 | ACC7 | 0.74683 | ACC7 | 0.75581 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Gap=39 | | Gap=40 | | Gap=41 | | Gap=42 | | Gap=43 | |
| ACC1 | 0.84129 | ACC1 | 0.83629 | ACC1 | 0.8372 | ACC1 | 0.83947 | ACC1 | 0.83174 |
| ACC2 | 0.8518 | ACC2 | 0.85569 | ACC2 | 0.84742 | ACC2 | 0.85326 | ACC2 | 0.84548 |
| ACC3 | 0.82001 | ACC3 | 0.82481 | ACC3 | 0.81826 | ACC3 | 0.8187 | ACC3 | 0.82132 |
| ACC4 | 0.8412 | ACC4 | 0.84273 | ACC4 | 0.83808 | ACC4 | 0.84325 | ACC4 | 0.84428 |
| ACC5 | 0.83883 | ACC5 | 0.83744 | ACC5 | 0.83883 | ACC5 | 0.84255 | ACC5 | 0.83976 |
| ACC6 | 0.84367 | ACC6 | 0.84119 | ACC6 | 0.8402 | ACC6 | 0.84913 | ACC6 | 0.84913 |
| ACC7 | 0.85677 | ACC7 | 0.86364 | ACC7 | 0.86416 | ACC7 | 0.86892 | ACC7 | 0.86205 |

**gap이 42**이고 **⑦번 데이터 조합**일 때 ACC가 가장 높음.

distance = 0 부터 4까지 증가시키며 test set Accuracy 측정

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| distance=0 | | distance=2 | | distance=4 | |
| ACC1 | 0.83947 | ACC1 | 0.83129 | ACC1 | 0.82901 |
| ACC2 | 0.85326 | ACC2 | 0.84402 | ACC2 | 0.84014 |
| ACC3 | 0.8187 | ACC3 | 0.81346 | ACC3 | 0.80952 |
| ACC4 | 0.84325 | ACC4 | 0.83549 | ACC4 | 0.83394 |
| ACC5 | 0.84255 | ACC5 | 0.84069 | ACC5 | 0.83744 |
| ACC6 | 0.84913 | ACC6 | 0.84764 | ACC6 | 0.84169 |
| ACC7 | 0.86892 | ACC7 | 0.85579 | ACC7 | 0.84672 |

distance가 올라갈 수록 정확도가 낮아짐. 즉, **180분 데이터 모두 사용**하는 것이 ACC가 가장 높음.**최종모델(Gap=42, distance=0, 데이터조합 ⑦번)**

|  |  |  |
| --- | --- | --- |
| 구분 | 정확도(Accuracy) | 적용 |
| train | 100.000% | model=RandomForest  metrics=Accuracy |
| test | 86.892% |

**5. 참고 문헌**

[1] https://spaceweather.rra.go.kr/obsenv3.do

[2] https://en.wikipedia.org/wiki/K-index