KUBIG
DATA SCIENCE & AI

THIS IS B.A.A.M 양갱

Team | 분류예측 2팀
18기 신인수 19기 이동주 19기 이지운

KUBIG
DATA SCIENCE & AI

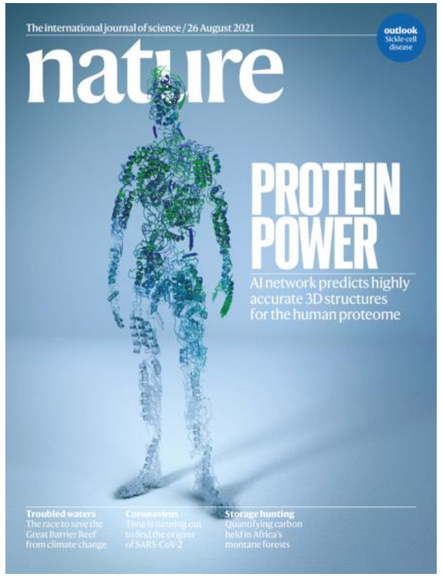
Beta
Alpha
Amino acid
Model

KUBIG
DATA SCIENCE & AI

01. 주제 선정

01. Motivation

딥러닝 기반 단백질 구조 예측 기술이 2021 Nature Method '올해의 기술'로 선정

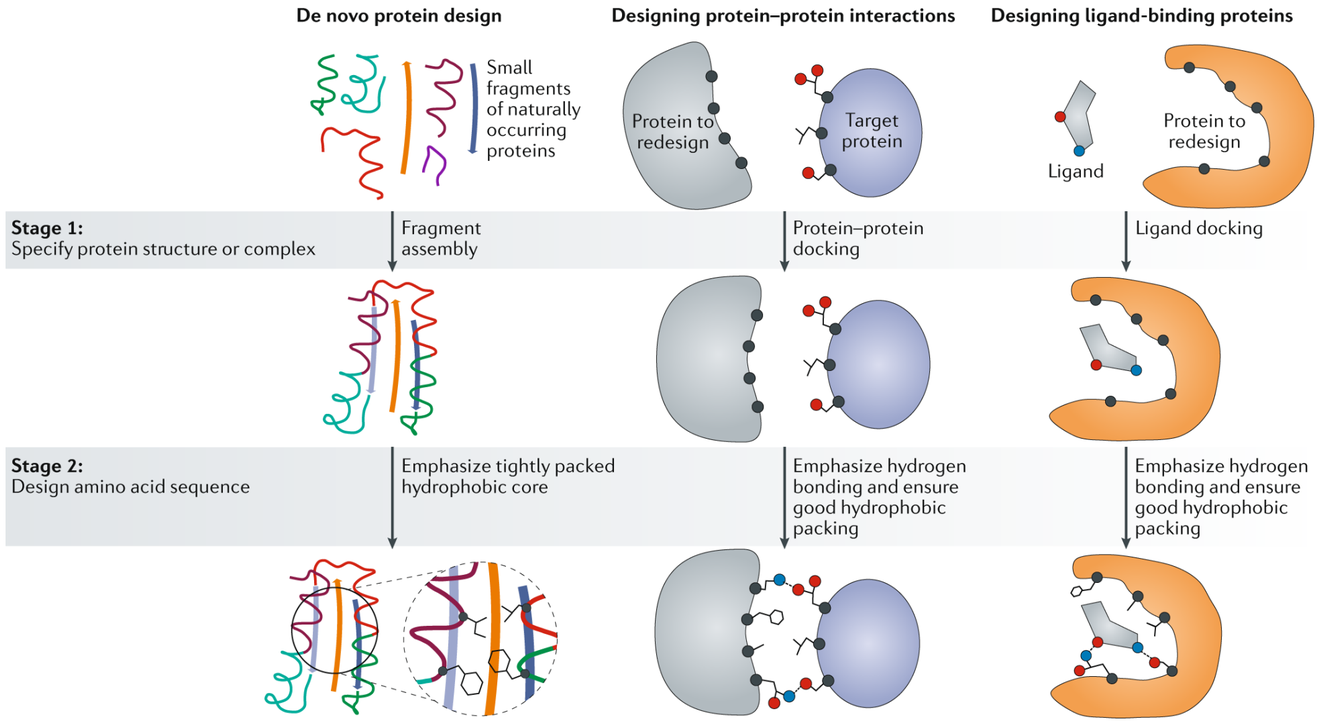


Google Deepmind:
AlphaFold

워싱턴대, 스탠포드대 등
RoseTTAFold
서울대: 백민경 Prof

서울대: Martin Steinneger
ColabFold

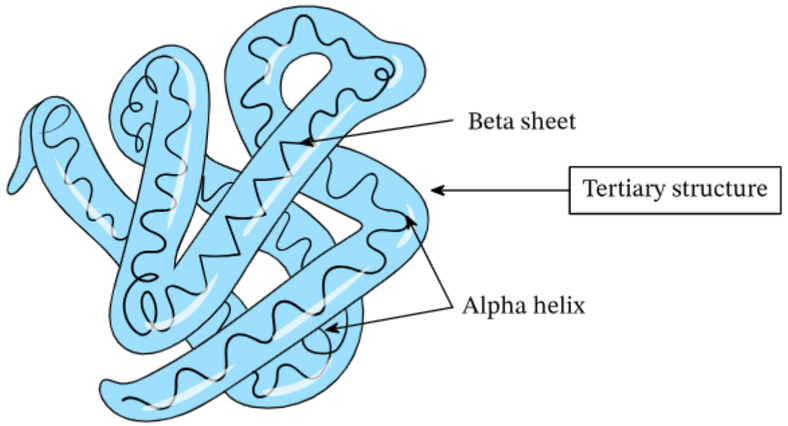
01. 단백질 구조 예측이란?

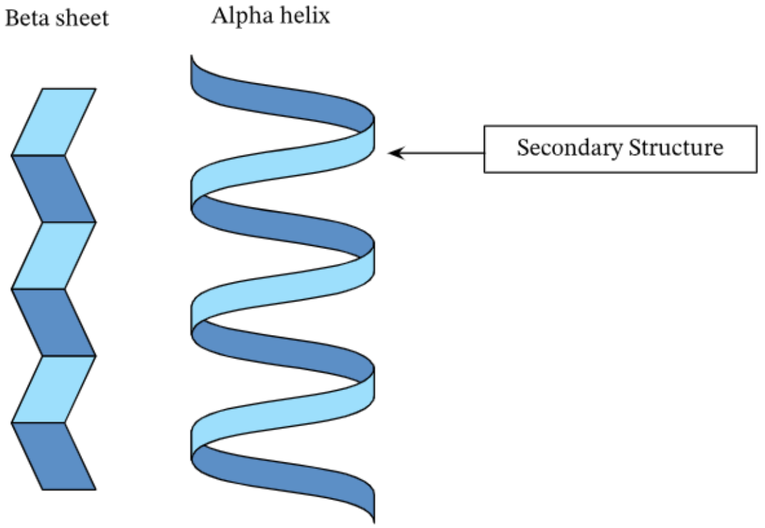


Protein-ligand interaction (PLI), protein-protein interaction (PPI), contact prediction, fold classification 등 다양 → 한정 필요

Kuhlman, B., & Bradley, P. (2019). Advances in protein structure prediction and design. *Nature reviews molecular cell biology*, *20*(11), 681-697.

01. 주제 탐색





Tertiary structure

단백질 3차 구조 예측을 위해서는 모든 원자에 대한 좌표가 필요.

Secondary structure

DSSP에 있는 C-a 좌표로 2차 구조 예측

01. 주제 탐색

RESEARCH ARTICLE

PROTEINS WILEY

NetSurfP-2.0: Improved prediction of protein structural features by integrated deep learning

nature methods | BRIEF COMMUNICATION
https://doi.org/10.1038/s41592-022-01488-1

Check for updates

OPEN

ColabFold: making protein folding accessible to all

Milot Mirdita[1,10] ✉, Konstantin Schütze[2], Yoshitaka Moriwaki[3,4], Lim Heo[5], Sergey Ovchinnikov[6,7,10] ✉ and Martin Steinegger[2,8,9,10] ✉

J. Vis. Commun. Image R. 71 (2020) 102844

Contents lists available at ScienceDirect

J. Vis. Commun. Image R.

journal homepage: www.elsevier.com/locate/jvci

Protein secondary structure prediction based on integration of CNN and LSTM model ☆

Jinyong Cheng, Yihui Liu *, Yuming Ma

School of Computer Science and Technology, Qilu University of Technology (Shandong Academy of Sciences), Jinan, China

Reports

BioTechniques

PS4: a next-generation dataset for protein single-sequence secondary structure prediction

Omar Peracha*[1]

[1]*Department for Continuing Education, University of Oxford, Rewley House, 1 Wellington Square, Oxford, OX1 2JA, UK; *Author for correspondence: omar.peracha@conted.ox.ac.uk*

ARTICLE

Check for updates

https://doi.org/10.1038/s41467-021-23303-9 OPEN

Structure-based protein function prediction using graph convolutional networks

Vladimir Gligorijević[1] ✉, P. Douglas Renfrew[1], Tomasz Kosciolek[2,3], Julia Koehler Leman[1], Daniel Berenberg[1,4], Tommi Vatanen[5,6], Chris Chandler[1], Bryn C. Taylor[7], Ian M. Fisk[8], Hera Vlamakis[5], Ramnik J. Xavier[5,9,10,11], Rob Knight[2,12,13], Kyunghyun Cho[14,15] & Richard Bonneau[1,4,14,16] ✉

Article

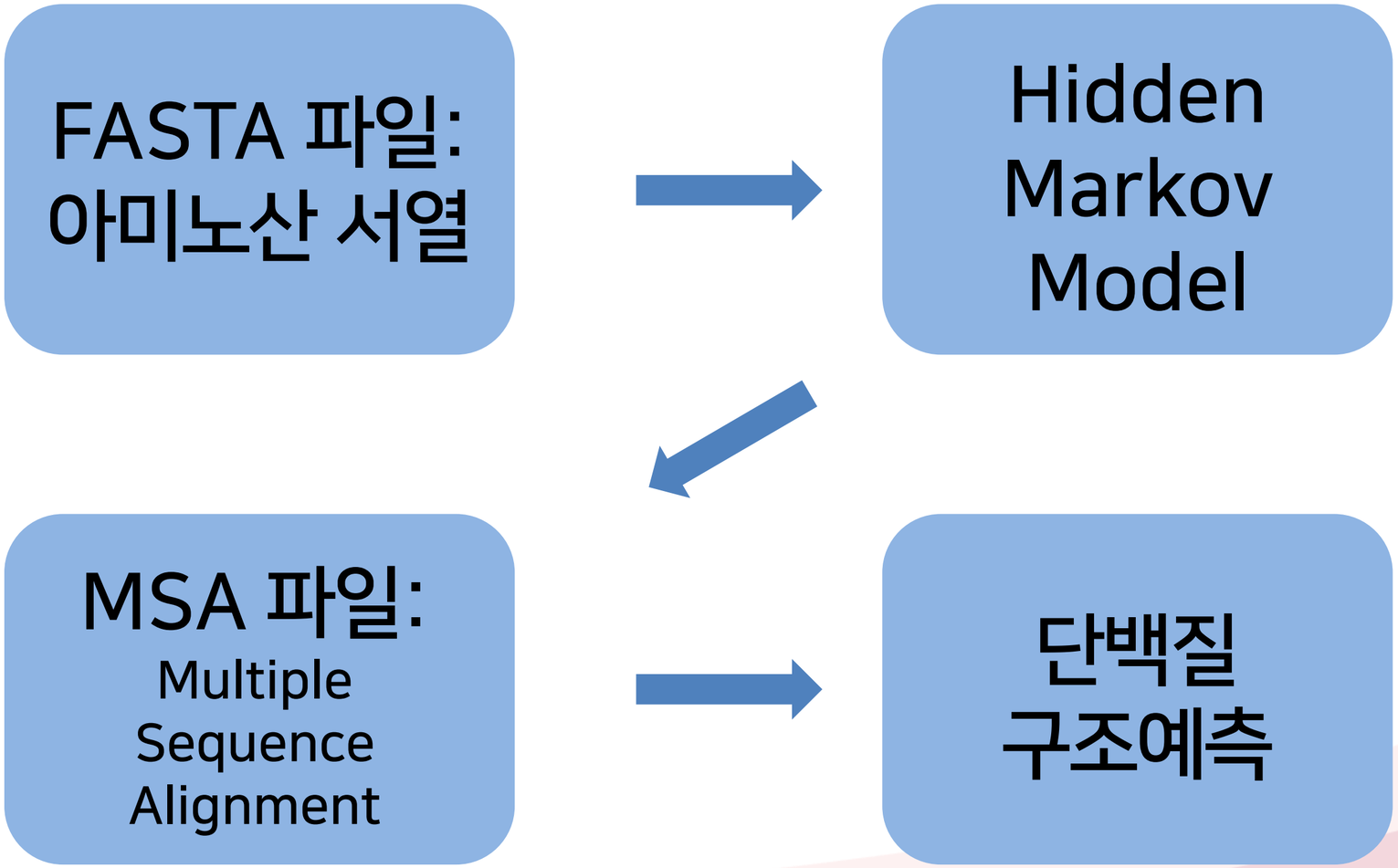
Highly accurate protein structure prediction with AlphaFold

ColabFold, AlphaFold 등의 선행 연구 논문 리서치
NetSurfP, CNN, LSTM, PS4 등 데이터 구조, 모델에 대한 리서치

KUBIG
DATA SCIENCE & AI

02. Data

02. 단백질 구조예측: 기존 과정



02. 전체적인 Workflow

02. 데이터 선정

Input data 선정

1. PDB → 데이터 하나씩 다운받아서 DSSP 돌려 feature extract (216606개로 최다)
2. PS4 → 18731개. DSSP 돌려 feature extract
3. Fasta → pytorch로 읽기 힘들다
4. NetSurfP → 10848개. DSSP 적용되어 있음
5. Sidechainnet → 디버깅 필요

02. 데이터 선정

Test dataset 선정

1. CASP → 새로운 단백질 데이터. 정확도 산출 방식 모름 → pass,,,,,
2. Hidden Markov model → MMseqs2로 전달하려 했는데 실패

→ PDB 데이터셋 split하여 진행

02. DSSP 형식

단백질 2차구조를 8가지로 구분

- H = α-helix
- B = residue in isolated β-bridge
- E = extended strand, participates in β ladder
- G = 3_{10}-helix
- I = π-helix
- P = κ-helix (poly-proline II helix)
- T = hydrogen-bonded turn
- S = bend

DSSP 파일 형식으로 총 216606개의 단백질 보유

02. DSSP 형식

```
....;....1....;....2....;....3....;....4....;....5....;....6....;....7..
  .-- sequential resnumber, including chain breaks as extra residues
  |    .-- original resname, not necessarily sequential, may contain letters for insertion codes
  |    | .-- one-letter chain ID
  |    | | .-- amino acid sequence in one letter code
  |    | | |  .-- secondary structure summary based on columns 19-38
  |    | | |  |.-- PPII (kappa) helix
  |    | | |  ||.-- 3-10 helix
  |    | | |  |||.-- alpha helix
  |    | | |  ||||.-- pi helix
  |    | | |  |||||.-- geometrical bend
  |    | | |  ||||||.-- chirality
  |    | | |  |||||||.-- beta bridge label
  |    | | |  ||||||||.-- beta bridge label
  |    | | |  |||||||||   .-- beta bridge partner resnum
  |    | | |  |||||||||   |   .-- beta bridge partner resnum
  |    | | |  |||||||||   |   |.-- beta sheet label
  |    | | |  |||||||||   |   ||   .-- solvent accessibility
  |    | | |  |||||||||   |   ||   |
#  RESIDUE AA STRUCTURE BP1 BP2  ACC     N-H-->O    O-->H-N    N-H-->O    O-->H-N    TCO  KAPPA ALPHA  PHI   PSI    X-CA   Y-CA   Z-CA
  1    1 A L              0   0  119      0, 0.0     2,-0.3     0, 0.0    33,-0.2   0.000 360.0 360.0 360.0 168.8    8.7    6.9   63.0
  2    2 A T  E     -a   34   0A  66     31,-2.0    33,-2.1     1,-0.1     2,-0.7  -0.456 360.0-169.6 -87.8 130.5    7.7    8.8   59.8
  3    3 A K  E >   -a   35   0A  66     -2,-0.3     3,-1.2    31,-0.2     4,-0.2  -0.850   8.5-179.0-111.3  94.7    7.6    7.5   56.2
  4    4 A L  G >> S+     0   0   23     31,-2.5     4,-2.9     -2,-0.7     3,-2.0   0.786  71.6  72.4 -65.6 -32.5    7.1   10.6   54.1
  5    5 A Y  G 34 S+     0   0    2     30,-0.8     -1,-0.3     1,-0.3    31,-0.1   0.709 101.0  46.1 -56.9 -26.7    7.0    8.7   50.7
  6    6 A Y  G <4 S+     0   0   39     -3,-1.2     -1,-0.3     2,-0.1     -2,-0.2   0.439 115.4  47.1 -93.5  -4.1    3.5    7.4   51.7
  7    7 A E  T <4 S-     0   0  138     -3,-2.0      2,-0.3     1,-0.2     -2,-0.2   0.825 135.2  -0.3 -99.6 -48.8    2.4   10.9   52.8
  8    8 A D    ><  -     0   0   57     -4,-2.9      3,-1.4     3,-0.1     -1,-0.2  -0.852  61.5-167.8-144.3 106.0    3.6   13.0   49.9
```

02. DSSP 형식

Scheme 1. Seven-Character Secondary Structure Information (7CSSI) Of DSSP[a]

```
[>,<,X,3] [>,<,X,4] [>,<,X,5] [S] [+,-] [ω,Ω] [ω,Ω]
|    1    |    2    |    3    | 4 | 5 | 6 | 7 |
```

[a]For each position the set of possible characters is shown in the rectangular brackets above. The space character "_" is set, if none of the conditions listed in the rectangular brackets apply for this position. In the original DSSP the position remains empty in this case. The Greek letters ω or Ω are place holders that for strand geometries identify the parallel or antiparallel partner strands, respectively. More details and the meaning of the other characters are given in the text. An example is shown in Table 1; a more detailed example can be found in Supporting Information Table S1.

Structure 열 한 글자씩 one-hot encoding

Zacharias, J., & Knapp, E. W. (2014). Protein secondary structure classification revisited: processing DSSP information with PSSC. *Journal of chemical information and modeling*, *54*(7), 2166-2179.

02. 데이터 전처리

PDB DSSP data에서 useful data 불러오기

Residue, AA, structure 등의 정보

```
def read_useful_data(file_path):
    obsolete_data = []
    header = []
    useful_data = []
    output_data = []
    with open(file_path, 'r') as f:
        # Skip lines until you find the line starting with '#'
        for line in f:
          if '#' in line:
            header = line.strip().split()
          elif '!' in line: # ! is missing residue
            pass
          elif not header:
            obsolete_data.append(line.strip())
          else:
            useful_data.append(line)


    return header, useful_data
```

```
[ ]  header, data = read_useful_data(f"{query}.dssp")
```

```
[ ]  print(header)
     print(len(header))
```

```
['#', 'RESIDUE', 'AA', 'STRUCTURE', 'BP1', 'BP2', 'ACC', 'N-H-->O', 'O-->H-N', 'N-H-->O', 'O-->H-N', 'TCO', 'KAPPA', 'ALPHA', 'PHI', 'PSI', 'X-CA', 'Y-CA', 'Z-CA']
19
```

02. 데이터 전처리

```
['#', 'RESIDUE', 'AA', 'STRUCTURE', 'BP1', 'BP2', 'ACC', 'N-H-->O', 'O-->H-N', 'N-H-->O', 'O-->H-N', 'TCO', 'KAPPA', 'ALPHA', 'PHI', 'PSI', 'X-CA', 'Y-CA', 'Z-CA']
```

```
for i in data[:10]:
  print(i)
```

```
    1    1 A V    >         0   0    1      0, 0.0     3,-0.6     0, 0.0     2,-0.4   0.000 360.0 360.0 360.0 121.4   11.2   -4.4   -1.5
    2    2 A V  B 3   +A  182   0A   5    180,-2.5   180,-1.9     1,-0.2   134,-0.2  -0.784 360.0   9.9 -93.0 133.4   14.7   -5.1   -2.7
    3    3 A G  T 3  S+    0   0   40    132,-0.6   146,-0.3     -2,-0.4    -1,-0.2   0.775  97.4 141.4  69.5  32.9   16.0   -8.6   -2.0
    4    4 A G    <  -    0   0   22     -3,-0.6     2,-0.3   144,-0.1   144,-0.2  -0.245  44.6-135.6 -97.3-179.4   12.6   -9.8   -0.9
    5    5 A T  E     -B  147   0B  85    142,-2.1   142,-2.5     -2,-0.1     2,-0.3  -0.919  39.5 -85.2-130.4 156.3   10.8  -13.1   -1.5
    6    6 A E  E>    -B  146   0B 111     -2,-0.3   140,-0.2   140,-0.2     2,-0.2  -0.497  46.2-130.0 -69.5 127.1    7.2  -13.6   -2.5
    7    7 A A  PP    -    0   0    1    138,-2.3   138,-0.1     -2,-0.3     2,-0.1  -0.478  23.0-106.8 -68.5 144.8    5.0  -13.6    0.6
    8    8 A Q  P<>   -    0   0  139     -2,-0.2     3,-1.6     1,-0.1     4,-0.5  -0.480  37.0-112.2 -68.1 147.4    2.4  -16.3    1.0
    9    9 A R  T 3  S+    0   0  105      1,-0.3     -1,-0.1     2,-0.1     3,-0.1   0.670 112.5  37.8 -55.2 -29.3   -1.0  -14.9    0.3
   10   10 A N  T 3  S+    0   0   85      1,-0.1     -1,-0.3     2,-0.0     -2,-0.0   0.303  89.8  92.2-113.5  17.6   -2.2  -15.1    3.9
```

02. 데이터 전처리

Split으로 열마다 분리.

```
[ ]  ex_1stresidue=[]
     ex_2ndresidue=[]
     a=[]
     aa=[]
     x_ca=[]
     y_ca=[]
     z_ca=[]


     for i in data[:10]:
       line = i.split()
       ex_1stresidue.append(line[0])
       ex_2ndresidue.append(line[1])
       a.append(line[2])
       aa.append(line[3])
       x_ca.append(line[-3])
       y_ca.append(line[-2])
       z_ca.append(line[-1])
```

```
residue_num 10
['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']

AA 10
['V', 'V', 'G', 'G', 'T', 'E', 'A', 'Q', 'R', 'N']

structure_x 10
[' ', 'B', 'T', ' ', 'E', 'E', 'P', 'P', 'T', 'T']

structure_detail 10
[' >      ', ' 3   +A ', ' 3  S+  ', ' <   -  ', '     -B ', '>    -B ', 'P    -  ', '<>   -  ', ' 3  S+  ', ' 3  S+  ']

BP1 10
['   0 ', ' 182 ', '   0 ', '   0 ', ' 147 ', ' 146 ', '   0 ', '   0 ', '   0 ', '   0 ']

BP2 10
[' 0  ', ' 0A ', ' 0  ', ' 0  ', ' 0B ', ' 0B ', ' 0  ', ' 0  ', ' 0  ', ' 0  ']

ACC 10
['   1 ', '   5 ', '  40 ', '  22 ', '  85 ', ' 111 ', '   1 ', ' 139 ', ' 105 ', '  85 ']

N_H__O_11 10
['    0', '  180', '  132', '   -3', '  142', '   -2', '  138', '   -2', '    1', '    1']

N_H__O_12 10
[' 0.0  ', '-2.5  ', '-0.6  ', '-0.6  ', '-2.1  ', '-0.3  ', '-2.3  ', '-0.2  ', '-0.3  ', '-0.1  ']

O__H_N_11 10
['  3', '180', '146', '  2', '142', '140', '138', '  3', ' -1', ' -1']

O__H_N_12 10
['-0.6  ', '-1.9  ', '-0.3  ', '-0.3  ', '-2.5  ', '-0.2  ', '-0.1  ', '-1.6  ', '-0.1  ', '-0.3  ']
```

02. 데이터 전처리

DataFrame으로 만들어서 공백 확인.

O(Others)로 처리

```
[ ] cleaned_data['structure_x'].unique()
```

```
array([' ', 'B', 'T', 'E', 'P', 'S', 'G', 'H'], dtype=object)
```

```
[ ] cleaned_data['structure_x_1']=cleaned_data['structure_x'].replace(' ', 'O')
```

```
[ ] cleaned_data[['structure_x', 'structure_x_1']].head()
```

	structure_x	structure_x_1
0		O
1	B	B
2	T	T
3		O
4	E	E

02. 데이터 전처리

AA, structure_x, structure_detail, BP2를 제외하고 numeric 데이터로 변환

```
num_variables = [var for var in variables_name if var not in ['AA', 'structure_x', 'structure_detail', 'BP2']]
num_variables
```

```
['residue_num',
 'BP1',
 'ACC',
 'N_H__O_11',
 'N_H__O_12',
 'O__H_N_11',
 'O__H_N_12',
 'N_H__O_21',
 'N_H__O_22',
 'O__H_N_21',
 'O__H_N_22',
 'TCO',
 'KAPPA',
 'ALPHA',
 'PHI',
 'PSI',
 'X_CA',
 'Y_CA',
 'Z_CA']
```

```
for var in num_variables:
    cleaned_data[var] = pd.to_numeric(cleaned_data[var], errors='coerce')
```

KUBIG
DATA SCIENCE & AI

03. Model

03. 모델 구성

앞선 논문 스터디에서 언어 모델, Transformer (BERT) 가 자주 사용되었음

LSTM, BERT 모델 시도

```python
import torch.nn as nn

class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(num_layers, x.size(0), hidden_size).to(x.device)
        c0 = torch.zeros(num_layers, x.size(0), hidden_size).to(x.device)

        out, _ = self.lstm(x, (h0, c0))
        out = self.fc(out)
        return out
```

LSTM

```python
[ ]  import torch.nn as nn
     from transformers import BertModel, AdamW

     class CustomBERTModel(nn.Module):
         def __init__(self, num_features, num_labels):
             super(CustomBERTModel, self).__init__()
             self.embedding = nn.Linear(num_features, 768)  # BERT's hidden size
             self.bert = BertModel.from_pretrained('bert-base-uncased')
             self.classifier = nn.Linear(768, num_labels)

         def forward(self, inputs):
             embeddings = self.embedding(inputs)
             outputs = self.bert(inputs_embeds=embeddings)[0]
             logits = self.classifier(outputs)
             return logits

     num_features = len(dataframes[0].columns) - 8  # Adjust to your input feature count
     num_labels = 8  # Number of secondary structure classes
     model = CustomBERTModel(num_features=num_features, num_labels=num_labels)
```

BERT

03. 모델 구성

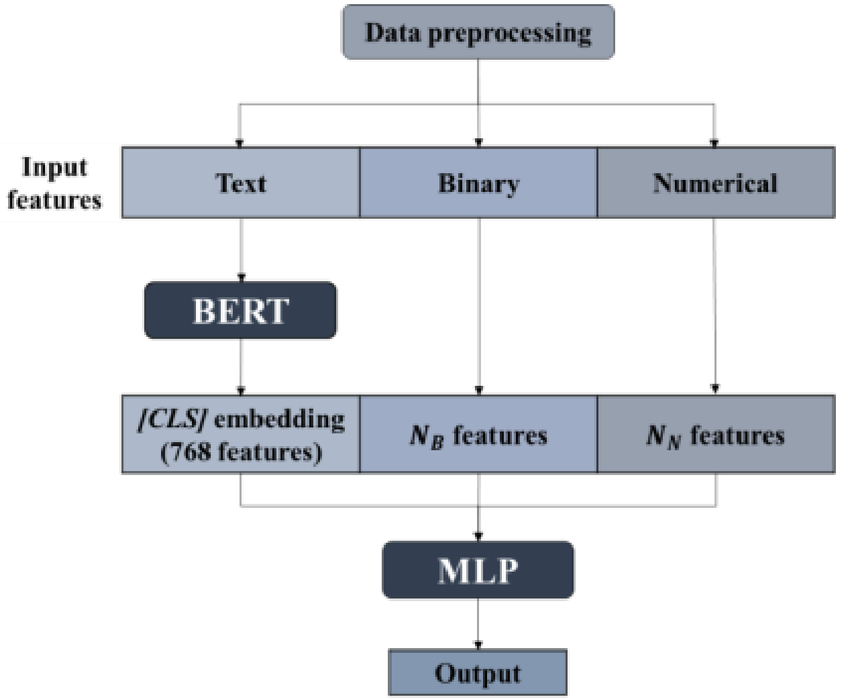
BERT 모델은 input이 sequence or sentence

≠ 우리 모델 input은 numeric + categorical data

→ BERT는 우리 데이터에 적합하지 않음 !

LSTM 모델로 진행

Figure 1. Overview of the proposed methodology.



KUBIG
DATA SCIENCE & AI

04. Train / Test

04. Train

아미노산 one-hot encoding 함수

총 20개 열

```
[ ]  amino_acids = list('ACDEFGHIKLMNPQRSTVWY')  # 20 standard amino acids

     def one_hot_encode(df, column, amino_acids):
         one_hot_df = pd.DataFrame(0, index=df.index, columns=amino_acids)
         for aa in amino_acids:
             one_hot_df[aa] = (df[column] == aa).astype(int)
         return pd.concat([df, one_hot_df], axis=1).drop(columns=['AA']) # 원래 'AA' 열 드랍
```

```
structures = list('HBEGITSO')

def one_hot_encode_structure(df, column, structures):
  one_hot_df = pd.DataFrame(0, index = df.index, columns = structures)
  for st in structures:
    one_hot_df[st] = (df[column] == st).astype(int) # True = 1, False = 0

  one_hot_df.columns = ['ss_'+i for i in structures]

  return pd.concat([df, one_hot_df], axis=1).drop(columns=['structure_x']) # 원래 'structure_x' 열 드랍
```

AA, structure_x, structure_detail 열 one-hot encoding 진행

04. Train

Save and Load Checkpoint

지금 우리가 다루고 있는 것이 200,000개 단백질이기 때문에 중간에 튕길 가능성이 매우 큼. 그래서 중간중간 checkpoint을 만들어서 모델을 저장하도록 하자.

```
[ ]  # checkpoint 저장 함수
     def save_checkpoint(model, optimizer, epoch, loss, filename='checkpoint.pth.tar'):
         state = {
             'epoch': epoch,
             'state_dict': model.state_dict(),
             'optimizer': optimizer.state_dict(),
             'loss': loss,
         }
         torch.save(state, filename)
         print(f'Checkpoint saved: {filename}')
```

```
def load_checkpoint(model, optimizer, filename='checkpoint.pth.tar'):
    checkpoint = torch.load(filename)
    model.load_state_dict(checkpoint['state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer'])
    epoch = checkpoint['epoch']
    loss = checkpoint['loss']
    print(f'Checkpoint loaded: {filename} (Epoch {epoch}, Loss {loss})')
    return epoch, loss
```

2만개의 단백질 데이터를 다루기 때문에 checkpoint로 모델 저장

04. Train

```
# Model Parameters
input_size = 60  # Total columns (68) - target columns (8)
hidden_size = 128
num_layers = 2
output_size = 8  # Number of secondary structure categories

# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Initialize the model, criterion, and optimizer
lstm_model = LSTMModel(input_size, hidden_size, num_layers, output_size).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(lstm_model.parameters(), lr=0.001)

# Training over each chunk
num_epochs = 10 ##########  epoch 2 -> 10, 50 .... ##########
checkpoint_path = '/content/drive/MyDrive/Model/checkpoint.pth.tar'

start_epoch = 0
# Load checkpoint if it exists
if os.path.exists(checkpoint_path):
    start_epoch, _ = load_checkpoint(lstm_model, optimizer, checkpoint_path)

for epoch in range(start_epoch, num_epochs):
    for i in range(1, 70):
        chunk_file = f'/content/drive/MyDrive/Model/checkpoints/checkpoint_data_chunk_{i}.pkl'
        train_model_on_chunk(chunk_file, lstm_model, criterion, optimizer, device, epoch, checkpoint_path)
```

LSTM model

Loss : CrossEntropy

Optimizer : Adam 으로 설정

Epoch는 10, 50으로 진행

100개 chunk 파일 중

70개를 trian에 사용

04. Test

```
# Assuming predictions is your numpy array of shape (n, 8)
predictions = np.array(predictions)

# Column names
column_names = ["B", "E", "G", "H", "I", "O", "S", "T"]

# Convert to DataFrame
df = pd.DataFrame(np.array(predictions).squeeze(axis=0), columns=column_names)

# Add a 'predict' column with the highest probability label
df['predict'] = df.idxmax(axis=1)

print(df)
```

```
                B         E             G             H             I  ₩
0    1.435113e-05  0.000024  3.106356e-05  3.905097e-07  1.347806e-07
1    3.369720e-04  0.000682  9.186351e-05  3.629135e-07  3.896270e-08
2    6.629902e-04  0.000215  9.174770e-05  1.599649e-06  6.459278e-07
3    1.477594e-06  0.000124  3.168064e-05  1.032419e-06  5.733825e-08
4    5.379265e-02  0.923216  2.259929e-07  9.669353e-08  3.067538e-07
..            ...       ...           ...           ...           ...
641  1.597045e-04  0.005808  5.796922e-06  1.659968e-07  4.709688e-08
642  8.671289e-03  0.982444  8.359780e-08  2.591750e-08  1.934312e-08
643  4.453050e-05  0.999896  1.544655e-10  3.878809e-08  2.373051e-11
644  6.080267e-05  0.991900  1.450951e-08  1.315002e-08  3.314085e-11
645  4.605211e-08  0.000050  7.931241e-08  8.490879e-10  1.360064e-09

                O             S             T predict
0    9.998690e-01  5.679093e-05  4.509064e-06       O
1    9.987556e-01  1.302315e-04  2.759167e-06       O
2    8.185791e-05  1.156003e-06  9.989454e-01       T
3    4.805398e-07  2.954897e-04  9.995455e-01       T
4    2.291579e-02  7.293502e-06  6.788142e-05       E
..            ...           ...           ...     ...
641  9.939949e-01  1.081930e-05  2.047901e-05       O
642  8.881188e-03  6.427015e-07  2.264809e-06       E
643  5.989440e-05  1.056192e-09  4.785475e-10       E
644  8.036416e-03  2.298449e-06  1.606639e-08       E
645  9.999499e-01  2.963181e-07  4.332114e-09       O

[646 rows x 9 columns]
```

나머지 30개의 chunk 파일로 test.

다음과 같이 각 이차구조 (B, E, G, H, I, O, S, T)

에 대한 확률 예측.

04. Test

```
[ ]  # Calculate average accuracy
     total_accuracy = 0
     total_dataframes = 0
     accuracy_list = []

     for result in results:
         for res in result['results']:
             total_accuracy += res['accuracy']
             total_dataframes += 1
             accuracy_list.append(res['accuracy'])

     average_accuracy = total_accuracy / total_dataframes
     print(f"Average Accuracy: {average_accuracy:.2f}% for {num_epochs} epochs")
```

```
Average Accuracy: 96.63% for 10 epochs
```

10 epochs test → 96.63% accuracy

```
[ ]  len(under_threshold)/total_dataframes
```

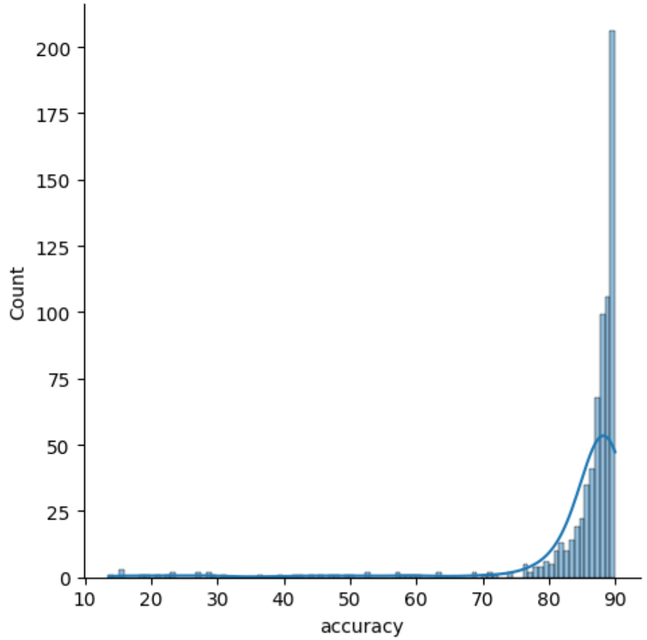
```
0.012484460086498223
```

```
[ ]  min(under_threshold['accuracy'])
```

```
13.636363636363635
```

```
import seaborn as sns
sns.displot(under_threshold['accuracy'], kde=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7c61ad6cc490>
```



90%보다 낮은 정확도는 threshold에 기록.

04. Test

```
# Calculate average accuracy
total_accuracy = 0
total_dataframes = 0
accuracy_list = []

for result in results:
    for res in result['results']:
        total_accuracy += res['accuracy']
        total_dataframes += 1
        accuracy_list.append(res['accuracy'])

average_accuracy = total_accuracy / total_dataframes
print(f"Average Accuracy: {average_accuracy:.2f}% for {num_epochs} epochs")
```

```
Average Accuracy: 96.80% for 50 epochs
```

50 epochs test → 96.80% 로 약간 상승

```
[ ]  len(under_threshold)/total_dataframes
```

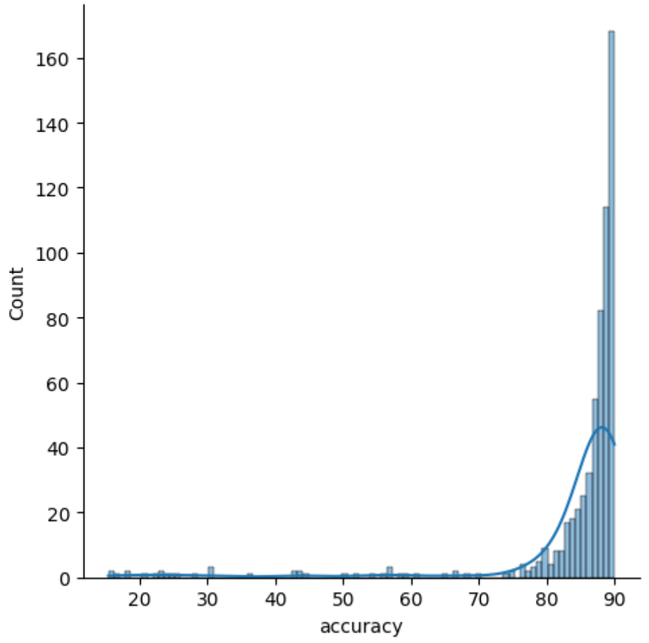
```
0.010768503440668173
```

```
[ ]  min(under_threshold['accuracy'])
```

```
15.384615384615385
```

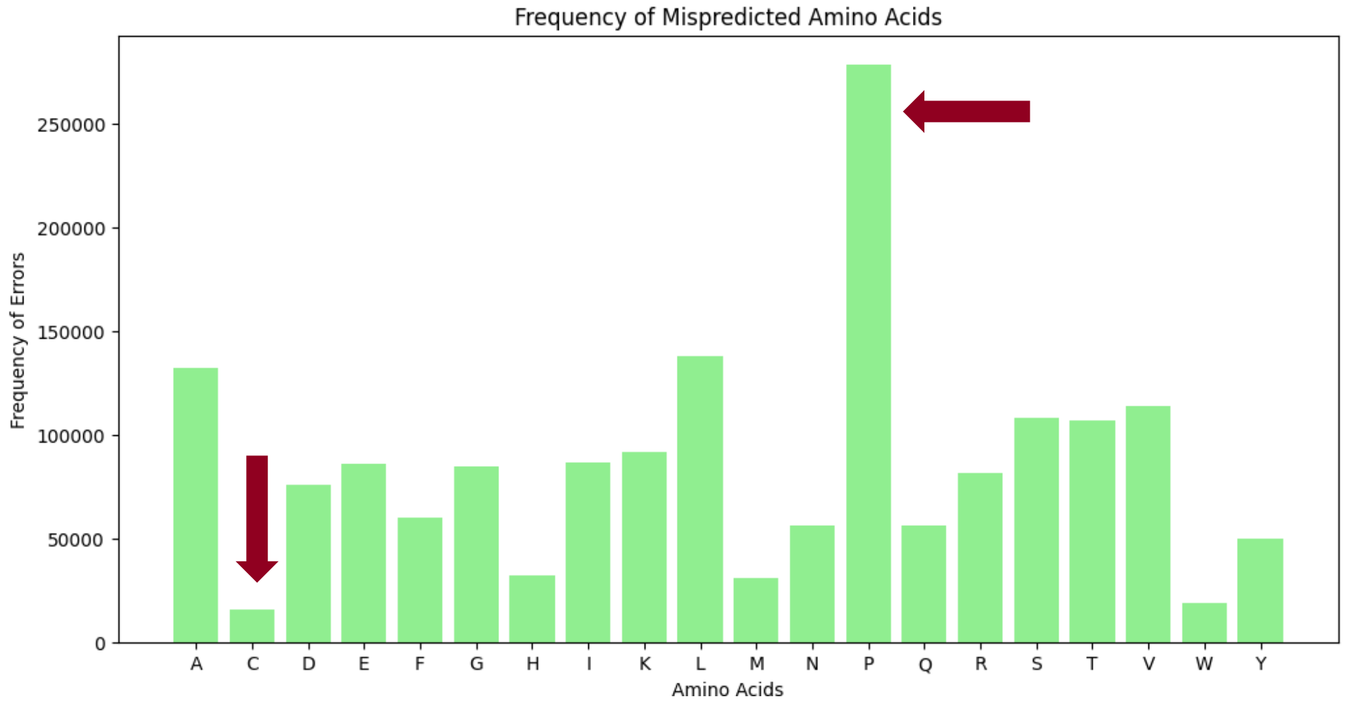
```
[ ]  import seaborn as sns
     sns.displot(under_threshold['accuracy'], kde=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7a15467389d0>
```



10 epochs보다 조금 개선된 모습

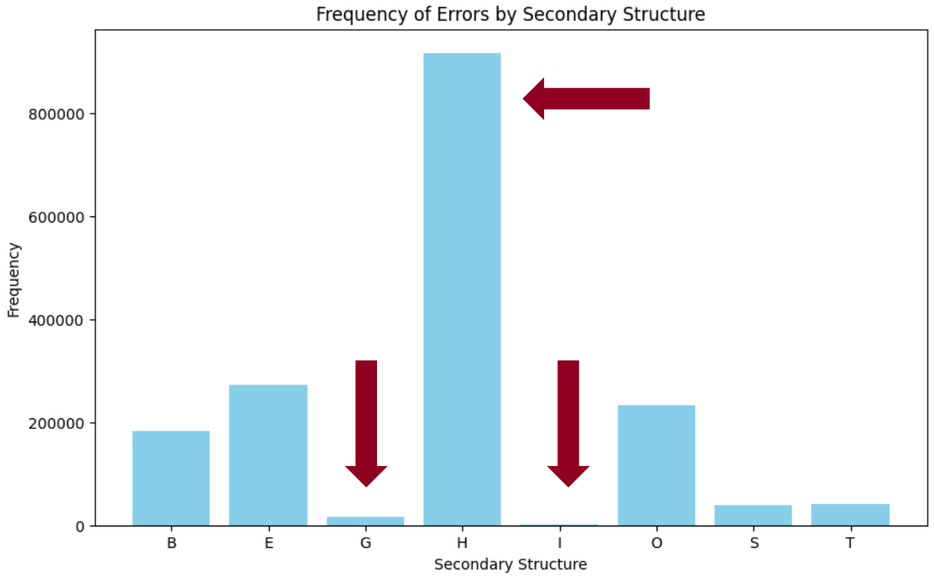
04. Results

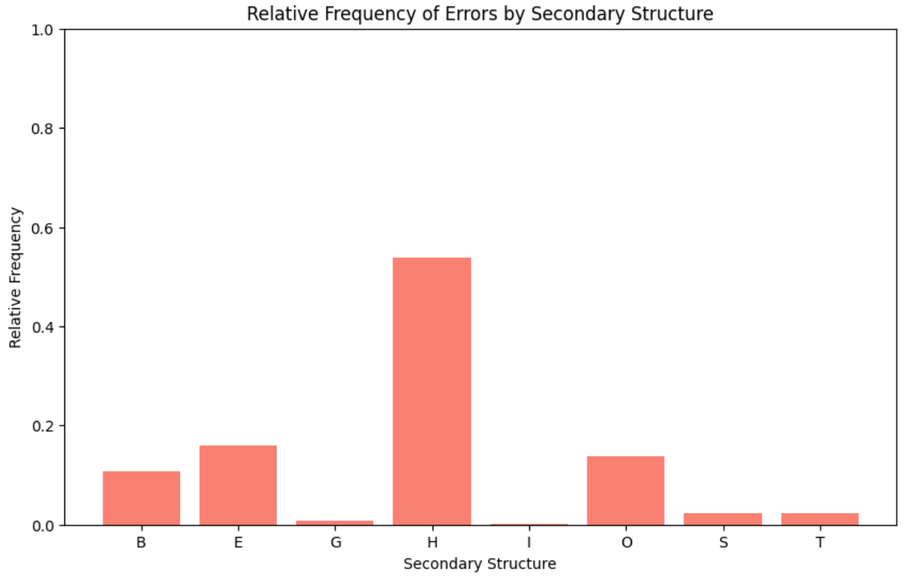


Proline 오류 최대: alpha-helix를 불안정하게 만드는 특징

Cysteine 오류 최소: beta-sheet을 안정하게 만드는 특징

04. Results





H 오류가 가장 많음: alpha-helix 오류가 가장 많음

G, I 오류가 가장 적음: 310-helix와 π-helix (alpha와 beta의 중간 형태)

→ Alpha helix보다 beta-sheet 형태 예측에 강세

04. Discussion:Limitation

- AlphaFold가 CASP14에 우승했던 CASP16에 observer 자격으로 참여하는 것이 프로젝트 목표 → output 형식이 달라 중도 포기
- FASTA → mmcif 파일의 형식으로 단백질 구조 예측
- Mmcif 파일의 경우 모든 원자의 좌표가 있음
- DSSP의 경우 Ca 만 있어 형식이 맞지 않음

04. Discussion:Limitation

- DSSP 데이터가 있는 단백질의 구조 예측만 가능하다. → 개선하려면 mmcif 파일로 training
- testing set이 따로 없어서 전체 데이터를 다운로드 받고, train-test를 나눴다. → CASP 등 대회를 통해 testing 하기
- CASP 대회의 경우 형식이 FASTA input, mmcif output이기 때문에 DSSP를 사용하면 해당 task는 수행 불가 → DSSP외의 최신 데이터베이스 사용
- DSSP 데이터베이스 자체의 한계 (1983년 개발, 2010년 update)
- mmcif의 경우 모든 원자의 X, Y, Z 좌표가 명시되어, 더 많은 정보 보유
- DSSP의 경우 Cα의 X, Y, Z 좌표만 있다 →mmcif output으로 예측 어려움

04. Discussion:Contribution

- KUBIG 최초의 bioinformatics 프로젝트
- 2년 뒤 CASP17에 도전할 경우 참고할 시행착오
  - Biopython DSSP 모듈: structure information 생략
  - HMM: MMseq2 불러오는데 어려움 → Google Colab 문제로 추정
  - Training 형색: mmcif 형식으로 해야
- 대용량 데이터 분석 (총 86GB) 경험

KUBIG
DATA SCIENCE & AI

Thank You