신약 개발 후보물질 추천을 위한 머신러닝 모델 설계

201724512 유경민
201724617 허수민

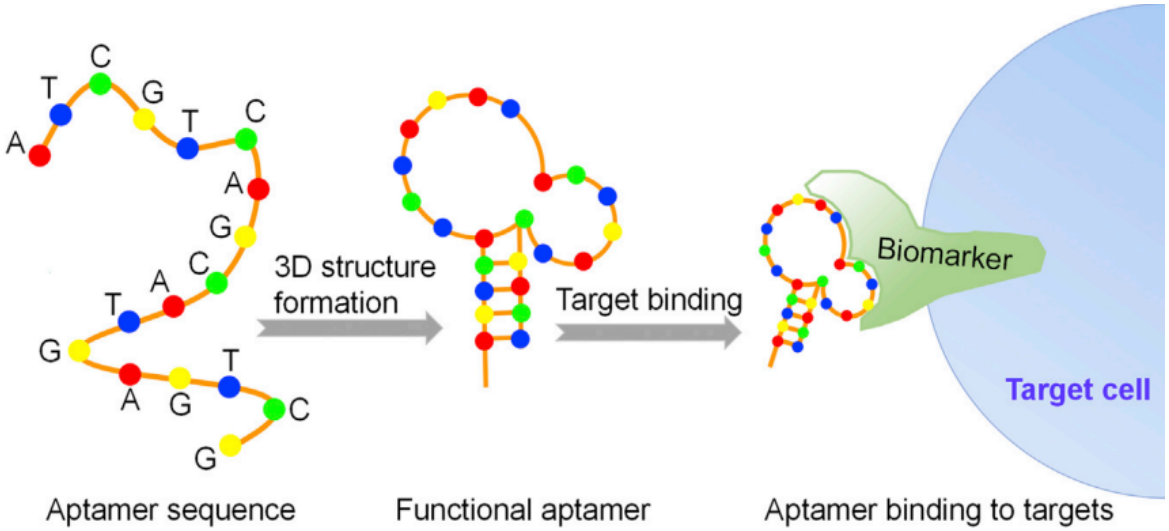
지도교수 송길태

Contents

1. 연구 배경

압타머(Aptamer)

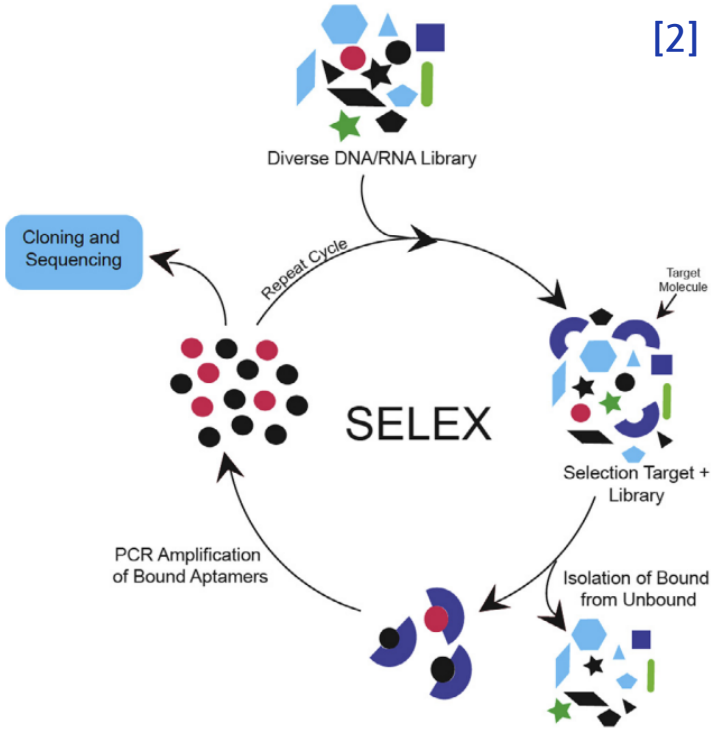
[1]



- RNA/DNA와 유사한 구조를 가지는 핵산 물질
- 항체와 비교될 수 있을 정도의 높은 결합 친화도
- 표적 단백질에 다양한 형태로 결합 가능

➡ 암 치료제 분야에서 사용

SELEX(Systematic evolution of ligands by exponential enrichment)



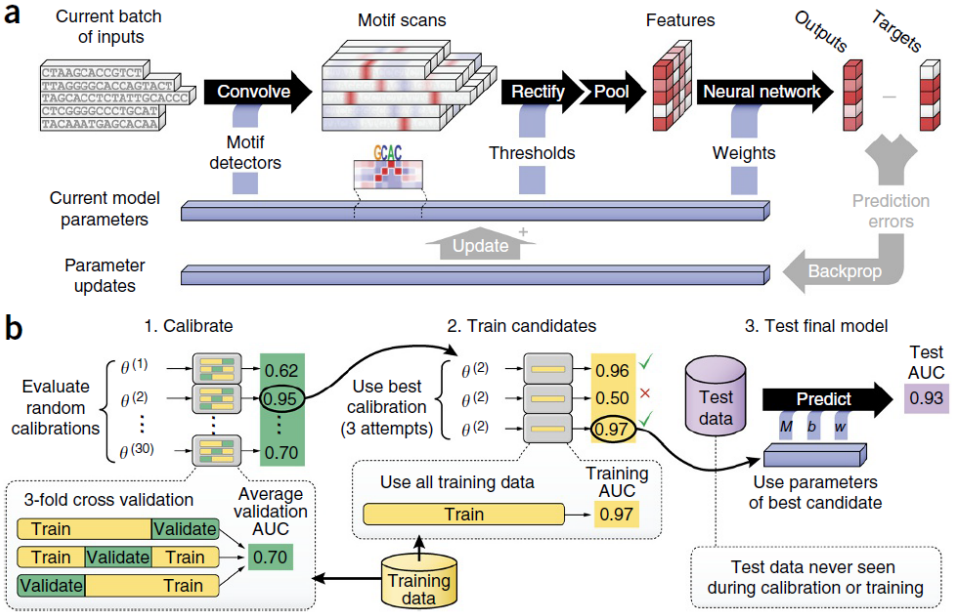
[2]

- 다양한 후보의 압타머 중 결합 친화도가 높은 물질을 선별하는 대표적인 과정
- 유전 물질을 합성 & 분리하는 작업을 반복적으로 수행
- 시간이 오래 소요

➡ **알고리즘 기반 예측 모델 등장**

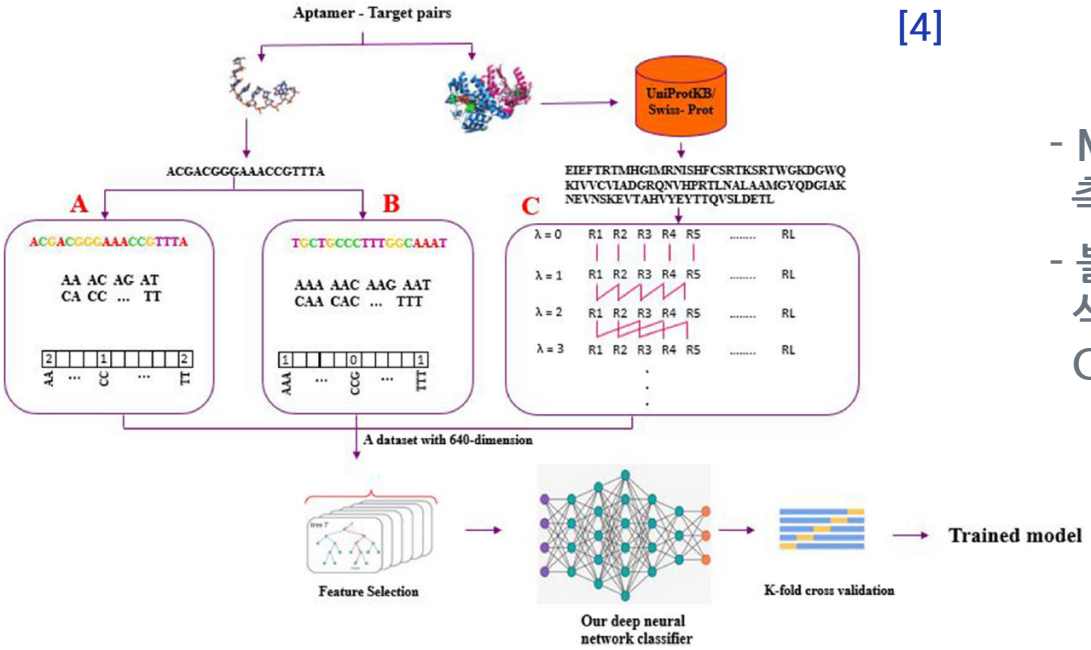
MLP 기반 결합 예측 모델 - DeepBind

[3]



- CNN에 기반을 두고 패턴 추출 및 예측을 수행하는 모델
- 노이즈가 포함된 데이터에도 학습 가능

MLP 기반 결합 예측 모델 - AptaNet



[4]

- MLP를 기반으로 단백질 결합 여부를 예측
- 불균형 데이터를 고려, 언더샘플링의 방식 중 하나인 NCL(Neighbourhood Cleaning Rule) 기반의 모델학습 수행

연구 목표

FoldID	EventID	seq	Bound
B	seq_000000_peak	AACTACGTAACTCTCCAGTA	1
B	seq_000000_shuf	AACTCTACCTAAGTACGTCA	0
B	seq_000003_peak	ATGACGCAATACTGCATCAG	1
B	seq_000003_shuf	ACACATGATGCTATCAAGCG	0
B	seq_000004_peak	GGTAAACCCACCACCTCTTC	1
B	seq_000004_shuf	GGTTCTCTACCAAACCACCC	0
B	seq_000008_peak	CGGGGATATGACGCAATAAT	1
B	seq_000008_shuf	CATATAACGGGGATGCGAAT	0

원본 데이터 (약 500종류의 API 데이터)

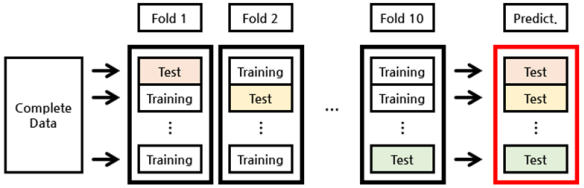
생물학적 정보 추가

bound	A	C	G	T	AA	...	Xc2.lambc	Xc2.lambc	Xc2.lambda.30
1	0.5	0.1	0.05	0.35	0.315789	...	0.022982	0.02506	0.023447
1	0.25	0.3	0.15	0.3	0.052632	...	0.022982	0.02506	0.023447
0	0.5	0.1	0.05	0.35	0.157895	...	0.022982	0.02506	0.023447
1	0.55	0.1	0	0.35	0.210526	...	0.022982	0.02506	0.023447

전처리된 데이터

```
opt=RMSprop(lr=0.0001, rho=0.9, epsilon=None, decay=0.0)
model = Sequential()
model.add(Dense(1024, input_dim=x.shape[1], activation='relu'))
model.add(Dense(512, input_dim=x.shape[1], activation='relu'))
model.add(Dense(256, activation='relu'))
```

예측모델 학습

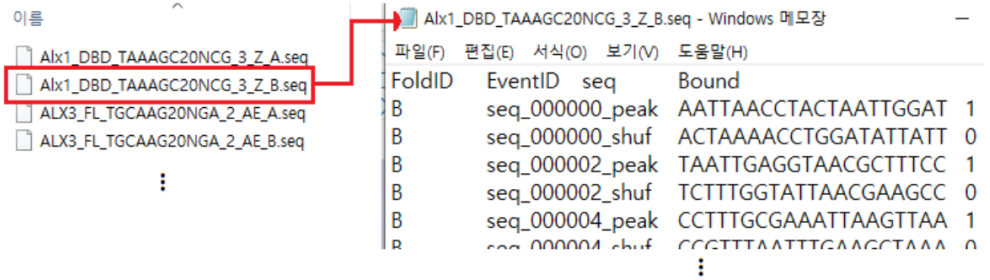


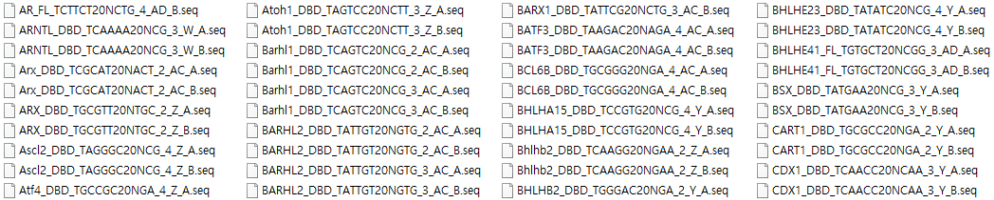
Cross Validation & Testing

- 기존 사례 모델 대비 성능이 향상된 예측 모델을 개발
- 결합 단백질과 유전물질의 생물학적 정보만으로 예측가능한 일반화된 모델 생성

2. 데이터 전처리

원본 데이터 설명

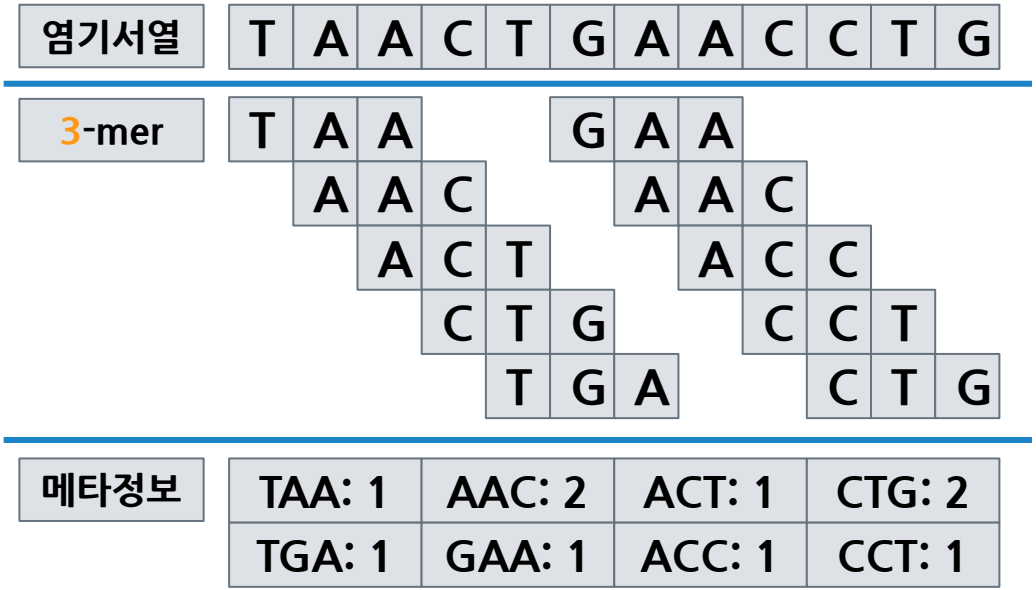




- 파일명: 단백질_Legand서열_A/B.seq
- A: Bound가 1(결합)인 정보만 존재
  B: Bound가 0(미결합) & 1인 정보 혼재

- 약 500종류 가량의 API(Aptamer Protein Interaction) 데이터 존재

k-mer 지표



- 주어진 문자열 내 길이가 k인 문자열의 집합을 의미
- k값이 클수록 주변 시퀀스와의 관계를 고려하는 비중이 커지나 메모리 필요 용량이 증가
- k값이 작으면 메모리에 저장되는 정보량이 적으나, 주변 서열의 관계 정보도 줄어들게 됨
- k=1인 경우 One-hot Encoding과 유사한 형태의 데이터 추출 가능

k-mer 지표

```python
from itertools import product

def kmer(seq, k):
    base = ['A', 'C', 'G', 'T']

    keys = list(map(''.join, product(base, repeat=k)))
    dicts = {}
    for key in keys: dicts[key] = 0

    for i in range(len(seq)-k+1):
        now = seq[i:i+k]
        dicts[now] += 1

    for key in keys: dicts[key] /= (len(seq)-k+1)
    return dicts
```

- k=1, 2, 3, 4인 경우의 메타정보 추출
- 각 k값을 기준으로 상대적인 비율(합산 1)의 형태로 염기서열 전처리를 수행

- A, C, G, T 이외의 N으로 표기된 문자는 A, C, G, T의 문자를 각각 넣어 서열을 증배하였음

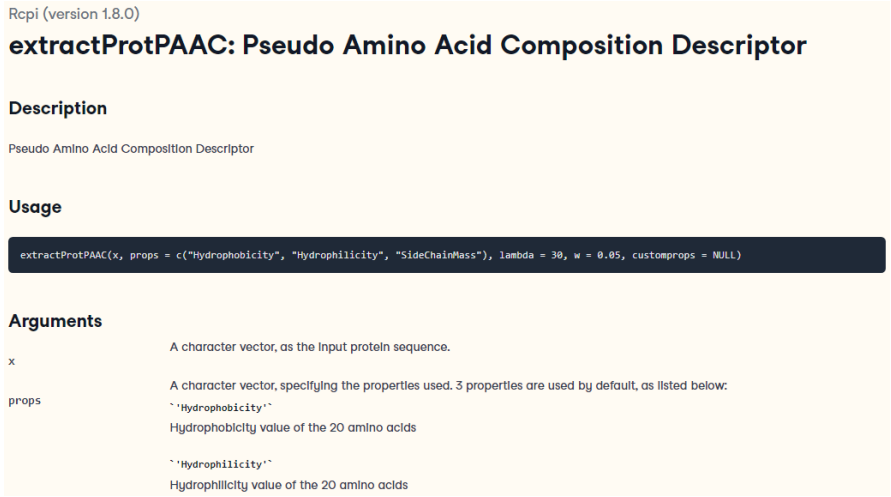
```python
def get_base_list(rslt_lst, seq):
    n_pos = seq.find('N')

    if n_pos >= 0:  # N(Any)이 있는 경우
        for base in ['A', 'C', 'G', 'T']:  # 재귀적으로 추가
            get_base_list(rslt_lst, seq[:n_pos] + base + seq[n_pos+1:])
    else:  # N이 없는 경우 정보를 list에 저장
        rslt_lst.append(seq)
```

k-mer 지표 전처리 후 데이터

protein	seq	bound	A	C	G	T	AA	AC	AG	AT
Alx1_DBD	GCAGATAATCTAATTACCCC	1	0.35	0.3	0.1	0.25	0.105263	0.052632	0.052632	0.157895
Alx1_DBD	CTCAGTCCTCGTCTCGATGG	1	0.1	0.35	0.25	0.3	0	0	0.052632	0.052632
Alx1_DBD	TCATAATCTAATTACGCTCG	1	0.3	0.25	0.1	0.35	0.105263	0.052632	0	0.157895
Alx1_DBD	GACTTCCTCAATCTAATTAG	1	0.3	0.25	0.1	0.35	0.105263	0.052632	0.052632	0.105263
Alx1_DBD	GCAGTTAATCTAATTAACCG	1	0.35	0.2	0.15	0.3	0.157895	0.052632	0.052632	0.105263
Alx1_DBD	TCCAGGCCTGATTTAATTAA	1	0.3	0.2	0.15	0.35	0.105263	0	0.052632	0.105263

Amino Acid Composition & Pseudo Amino Acid Composition

[5]



Rcpi (version 1.8.0)

extractProtPAAC: Pseudo Amino Acid Composition Descriptor

Description

Pseudo Amino Acid Composition Descriptor

Usage

```
extractProtPAAC(x, props = c("Hydrophobicity", "Hydrophilicity", "SideChainMass"), lambda = 30, w = 0.05, customprops = NULL)
```

Arguments

x	A character vector, as the input protein sequence.

props	A character vector, specifying the properties used. 3 properties are used by default, as listed below:
	``'Hydrophobicity'``
	Hydrophobicity value of the 20 amino acids
	
	``'Hydrophilicity'``
	Hydrophilicity value of the 20 amino acids

- 염기서열의 인코딩과 더불어 단백질 구성 형태도 고려하여 일반적인 모델을 제작
- 아미노산의 빈도, 단백질 결합 구조 정보를 표현하는 기법으로 AAC와 PseAAC 존재

- AAC: 단백질 시퀀스 내 특정 아미노산의 출현 빈도를 단순히 표현 (결합정보 무시)
- PseAAC: 2001년 Kuo-Chen Chou에 의해 [6] 등장한 결합정보를 고려한 단백질 표현 기법

Amino Acid Composition & Pseudo Amino Acid Composition

```
# 라이브러리 불러오기
library("Rcpi")

file = "csv파일 경로 작성"
labelList = read.csv(file, header=TRUE)

rslt_df = c()
(i 1:nrow(labelList)){
        nowe = labelList[i, 1]
        nowl = labelList[i, 2]

        # 단백질 구조 추출
        seqs = getProt(nowe, from = 'uniprot', type = 'aaseq')[[1]][[1]]
        # 단백질 meta 정보 추출
        rslt = extractProtPAAC(seqs)

        trslt = t(rslt)  # 행/열 뒤집기
        rownames(trslt) = nowl  # 행 이름 지정

        rslt_df = rbind(rslt_df, trslt)
}

directory_path = '저장할 폴더 위치'
setwd(directory_path)
write.csv(rslt_df, file='protein_data.csv', quote=FALSE)
```

- R언어 환경에서 Rcpi 라이브러리를 이용해 단백질 PseAAC 정보를 추출
- 단백질 정보는 UniProt[7] 의 DB 정보를 기반으로 얻어내었음

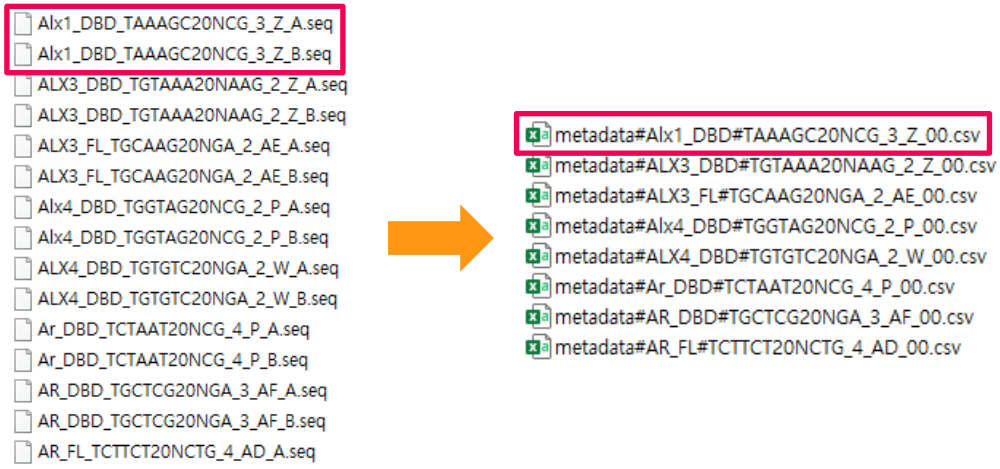
PseAAC 정보가 담긴 데이터

	Xc1.A	Xc1.R	Xc1.N	Xc1.D	Xc1.C	Xc1.E	Xc1.Q	Xc1.G	Xc1.H	Xc1.I	Xc1.L
Alx1	5.777514	5.777514	4.727057	3.6766	1.575686	5.777514	3.939214	4.464442	3.151371	1.575686	6.302742
ARX	29.79562	9.746231	1.113855	6.68313	1.949246	13.92319	4.45542	12.80933	2.784638	1.949246	15.87243
ALX3	7.778015	5.555725	2.22229	3.333435	1.666717	5.277939	3.333435	9.444732	3.333435	1.388931	8.333587
AR_FL	22.95207	11.61771	5.667177	10.48428	7.650689	15.58474	20.11848	27.20245	5.383818	6.517254	24.65222

전체 데이터 전처리

bound	A	C	G	T	AA	...	Xc2.lambc	Xc2.lambc	Xc2.lambda.30
1	0.5	0.1	0.05	0.35	0.315789		0.022982	0.02506	0.023447
1	0.25	0.3	0.15	0.3	0.052632		0.022982	0.02506	0.023447
0	0.5	0.1	0.05	0.35	0.157895		0.022982	0.02506	0.023447
1	0.55	0.1	0	0.35	0.210526		0.022982	0.02506	0.023447
1	0.4	0.2	0.1	0.3	0.105263		0.022982	0.02506	0.023447

- 앞선 k-mer 지표와 PseAAC 정보를 합쳐서 각 API별 학습용 데이터셋을 구축



- 데이터셋 생성 과정에서 같은 종류의 단백질 데이터인 경우 하나의 데이터셋으로 합쳐서 생성 (단, Legand sequence가 다른 경우엔 일단 분리해서 저장)

데이터 샘플링

```
# 각 dataframe에서 몇 퍼센트 가량 데이터 추출할지 결정
percentage = 1.0

# 최소 샘플링 데이터 개수. 만일 전체 데이터가 이것보다 작으면 모든 데이터를 추출
min_sample = 10000

# 최대 샘플링 데이터 개수. 최소/최대 반대로 값 작성 시 논리오류 발생할 수 있음
max_sample = 10000
```

```
for file in file_list:
    now_protein = file.split('#')[1]
    file_name = file
    print(now_protein + ' 진행 중....  ', end='')

    datafile = datadir + '/' + file_name
    nowdf = pd.read_csv(datafile)

    nowsize = len(nowdf)
    sample_size = min(nowsize, max(min_sample, min(int(nowsize*percentage), max_sample)))

    one_protein = protein_data[protein_data.seq  == now_protein]

    nowdf_rand = nowdf.sample(n=sample_size)

    nowdf_cut = nowdf_rand.drop(['seq'], axis=1)
    nowdf_cut.protein = now_protein

    one_full_data = pd.merge(nowdf_cut, one_protein, left_on='protein', right_on='seq').drop(['protein'], axis=1)
    # 실제 파일 저장.
    one_full_data.to_csv(nowdir + '/data/finalcsv/sample#' + now_protein + '.csv', index = False)
    print('>>완료!<<', end='  [>  ')
    print('추출된 사이즈: ' + str(sample_size))
```

- 전체 데이터로 학습하게 될 경우 200GB 이상의 용량 필요
- 각 API 데이터별 대표 sample만 추출하는 샘플링 코드를 별도로 제작
- 모델 학습을 위한 경량화된 데이터셋 랜덤 생성

sample#Alx1_DBD.csv	sample#BARX1_DBD.csv	sample#CEBPD_DBD.csv	sample#Dbp_DBD.csv	sample#E2F4_DBD.csv
sample#ALX3_DBD.csv	sample#BATF3_DBD.csv	sample#CEBPE_DBD.csv	sample#DBP_FL.csv	sample#E2F7_DBD.csv
sample#ALX3_FL.csv	sample#BCL6B_DBD.csv	sample#CEBPG_DBD.csv	sample#DLX1_DBD.csv	sample#E2F8_DBD.csv
sample#Alx4_DBD.csv	sample#BHLHA15_DBD.csv	sample#CEBPG_FL.csv	sample#Dlx2_DBD.csv	sample#EBF1_FL.csv
sample#Ar_DBD.csv	sample#Bhlhb2_DBD.csv	sample#CENPB_FL.csv	sample#DLX3_DBD.csv	sample#EGR1_DBD.csv
sample#AR_FL.csv	sample#BHLHB3_FL.csv	sample#CLOCK_DBD.csv	sample#DLX4_DBD.csv	sample#EGR1_FL.csv
sample#ARNTL_DBD.csv	sample#BHLHE22_DBD.csv	sample#CPEB1_FL.csv	sample#DLX5_FL.csv	sample#EGR2_DBD.csv
sample#Arx_DBD.csv	sample#BHLHE23_DBD.csv	sample#CREB3_FL.csv	sample#DLX6_DBD.csv	sample#EGR2_FL.csv

➡ 데이터를 병합하여 모델 학습에 이용!

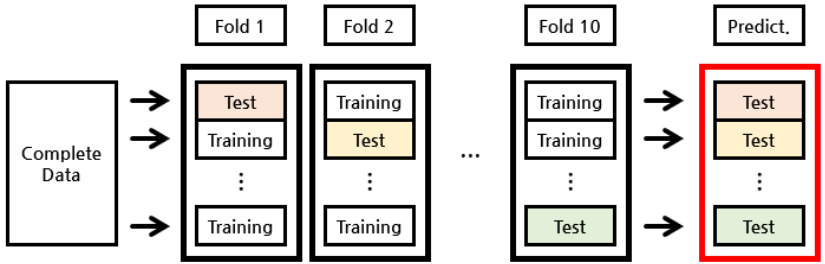
3. 모델 구축 및 성능비교

평가 지표 선정

$$(F1\ score) = 2 \times \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}}$$

$$= 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

- 정밀도와 재현율을 동시에 고려하여 데이터의 불균형 상태에서도 성능평가가 용이한 F1-score를 기반으로 성능 측정

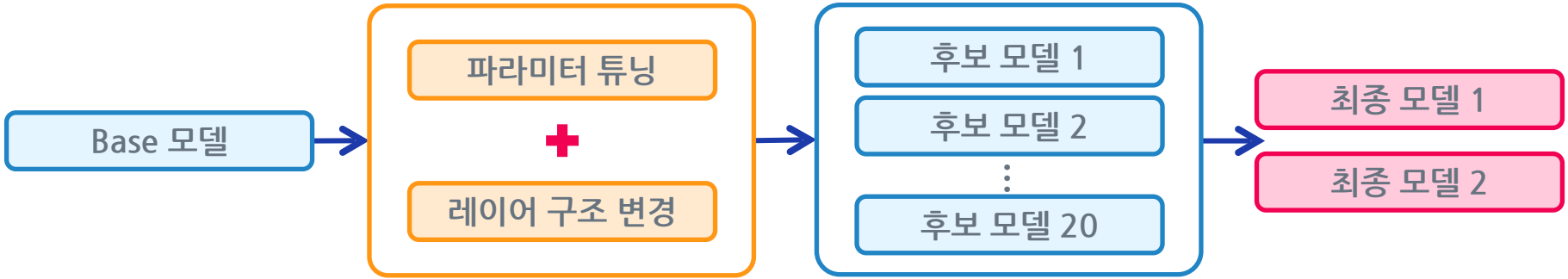


- 정확한 성능 측정을 위해 10-Fold Cross Validation 과정을 거쳐 F1-score의 평균을 계산

모델 구축

```
#Optimization Algorithm
opt=keras.optimizers.RMSprop(lr=0.00014, rho=0.9, epsilon=None, decay=0.0)

#Multi Layer Perceptron Model
model = Sequential()
model.add(Dense(128, input_dim=639, activation='relu'))
keras.layers.AlphaDropout(0.3, noise_shape=None, seed=None)

model.add(Dense(128, activation='relu'))
keras.layers.AlphaDropout(0.3, noise_shape=None, seed=None)
```

- 구조적으로 변형이 용이한 AptaNet 모델을 기반으로 튜닝을 수행
- 이후 레이어의 구조를 변경해가며 약 20가지의 후보 모델을 선정
- 최종적으로 기존 모델 대비 성능이 좋은 두 모델을 채택하였음



하이퍼 파라미터 튜닝 (batch_size, epochs)

```
seed = 6
np.random.seed(seed)
model = KerasClassifier(build_fn = create_model, verbose =1)

batch_size = [50, 200, 100]
epochs = [50, 100,200]
param_grid = dict(batch_size=batch_size, epochs=epochs)

grid = GridSearchCV(estimator = model, param_grid = param_grid,
        cv = KFold(random_state=seed,shuffle=True), verbose =1)
grid_results = grid.fit(x_resampled, y_resampled)
print("Best: {0}, using {1}".format(
        grid_results.best_score_, grid_results.best_params_))

means = grid_results.cv_results_['mean_test_score']
stds = grid_results.cv_results_['std_test_score']
params = grid_results.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print('{0} ({1}) with: {2}'.format(mean, stdev, param))
```

```
Best: 0.8998814105987549, using {'batch_size': 100, 'epochs': 100}
0.8619400858879089 (0.04143294163598879) with: {'batch_size': 50, 'epochs': 50}
0.8926722049713135 (0.01974376316175716) with: {'batch_size': 50, 'epochs': 100}
0.8926671862602233 (0.023526674849552395) with: {'batch_size': 50, 'epochs': 200}
0.8448996663093566 (0.007430710366369083) with: {'batch_size': 200, 'epochs': 50}
0.863494050502777 (0.030913171415396783) with: {'batch_size': 200, 'epochs': 100}
0.8998814105987549 (0.014584248215795424) with: {'batch_size': 100, 'epochs': 100}
0.8384358286857605 (0.029215181191311116) with: {'batch_size': 100, 'epochs': 50}
0.84601229429245 (0.034495465520618294) with: {'batch_size': 200, 'epochs': 200}
0.8786211371421814 (0.04362401317217036) with: {'batch_size': 100, 'epochs': 200}
```

- 과적합 현상 방지를 위해 GridSearch를 기반으로 최적의 batch_size와 epochs를 찾는 작업 수행
- batch_size 후보: 50, 100, 200
- epochs 후보: 50, 100, 200

- 최적 파라미터: batch_size=100, epochs=100

하이퍼 파라미터 튜닝 (Learning Rate, rho)

```
seed =6
np.random.seed(seed)
model = KerasClassifier(build_fn = create_model,
                        epochs =5, batch_size =20, verbose =0)

rho = [0.9, 0.8, 0.7]
lr = [0.0001, 0.001, 0.00001]
param_grid = dict(lr=lr, rho=rho)
grid = GridSearchCV(estimator = model, param_grid = param_grid,
                    cv = KFold(random_state=seed,shuffle=True), verbose =1)
grid_results = grid.fit(x_resampled, y_resampled)
```

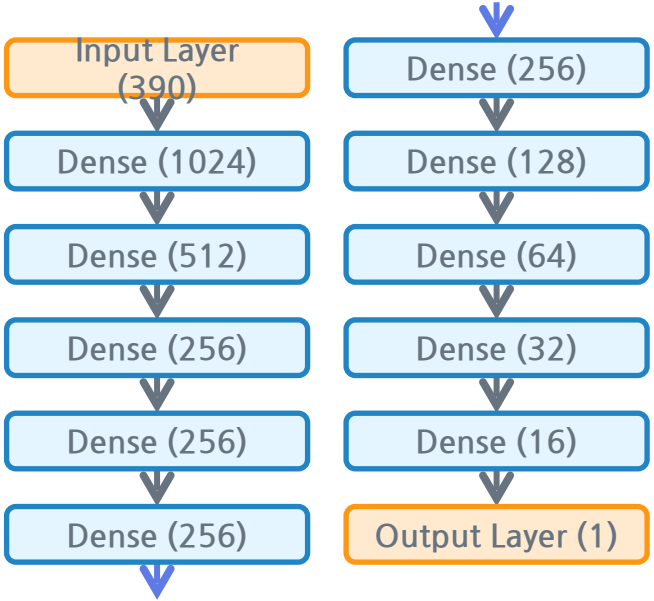
```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
Best: 0.8376875996589661, using {'lr': 0.0001, 'rho': 0.9}
0.7038209438323975 (0.015536282900006072) with: {'lr': 0.001, 'rho': 0.9}
0.68790682554245 (0.015500385441618405) with: {'lr': 0.0001, 'rho': 0.8}
0.6882755398750305 (0.013634766766607386) with: {'lr': 0.0001, 'rho': 0.7}
0.8376875996589661 (0.019345160104122565) with: {'lr': 0.0001, 'rho': 0.9}
0.8232771158218384 (0.02733706506326075) with: {'lr': 0.001, 'rho': 0.8}
0.8278290510177613 (0.010495489895340363) with: {'lr': 0.001, 'rho': 0.7}
0.6776666164398193 (0.010234468325862856) with: {'lr': 1e-05, 'rho': 0.9}
0.6776666164398193 (0.010234468325862856) with: {'lr': 1e-05, 'rho': 0.8}
0.6776666164398193 (0.010234468325862856) with: {'lr': 1e-05, 'rho': 0.7}
```

- 빠른 최적해 학습을 위해 Learning rate(lr)와 Rho 값 역시 GridSearch를 기반으로 파라미터 탐색
- lr 후보: 0.001, 0.0001, 0.00001
- rho 후보: 0.7, 0.8, 0.9

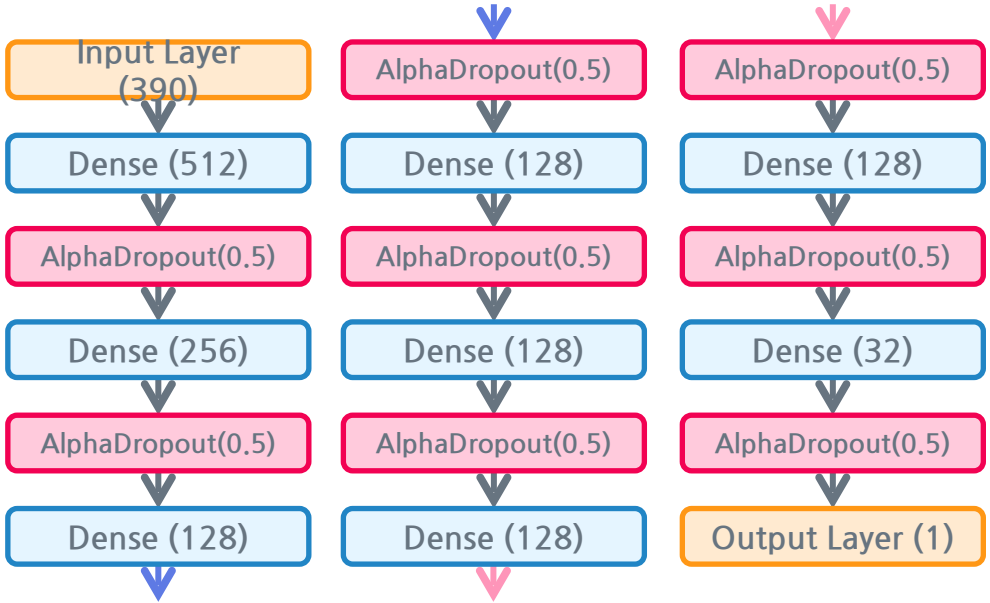
- 최적 파라미터: lr=0.0001, rho=0.9

최종 예측 모델 선정

- 모델1 구조

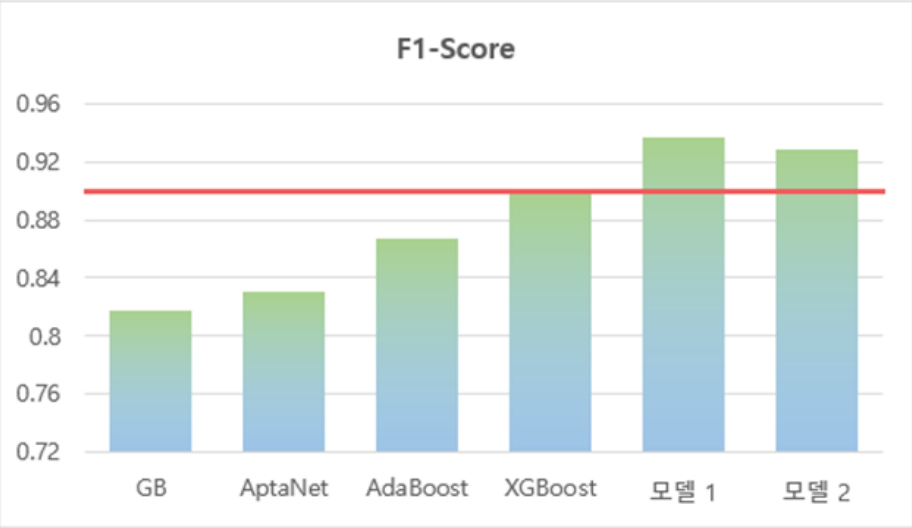


- 모델2 구조



성능 비교

	GradientBoost	AptaNet	AdaBoost	XGBoost
F1-Score (CV)	0.817	0.831	0.867	0.898



- 기존 모델들의 성능 => F1-Score 기준 0.900 미만의 값을 보임

- 모델 1 F1-Score: 0.937
- 모델 2 F1-Score: 0.929
- 모델 1, 2 모두 0.900 이상의 F1-Score 값을 얻어내었음 (가장 성능이 좋았던 기존의 XGBoost 모델 점수 대비, 절대치로 모델 1은 4%, 모델 2는 3% 가량의 성능 향상 달성)

연구 결론 및 발전가능성 제시

데이터 샘플링

- 약 500종류 총 9천만 개 API 데이터를 기반으로 생물학 정보를 속성으로 추가하여 기존 모델 대비 향상된 성능의 모델 구축을 하였음 (F1-Score 기준)
- 단백질의 명칭을 고려하지 않아도 데이터 기반으로 결합 여부를 예측할 수 있다는 장점 존재

모델 구축

- MLP 구조 기반으로 모델의 하이퍼파라미터를 GridSearch 기반 튜닝하여 최적의 파라미터를 찾고, 적절한 학습을 통해 성능 향상을 이뤄낼 수 있었음
- Drop-out의 파라미터 조정을 고려한 일반화된 모델을 생성해야 할 필요 존재

향후 연구 방향

- Drop-out, EarlyStopping 등 딥러닝 모델을 섬세하게 조정할 수 있는 기법을 적용하여 더욱 최적화된 모델을 생성하는 방향으로 연구
- 생물학적 정보를 토대로 적절한 파생변수 생성과 차원축소 기법을 적용하여 설명력 높은 모델을 제작하는 방식으로 연구

4. 연구 정보

개발 일정

		5월		6월		7월		8월		9월		10월	
		上	下	上	下	上	下	上	下	上	下	上	下
계획	착수보고서 작성												
분석	생물학 전문지식 이해												
	사용할 주요 기술 사전 조사												
설계	학습 모델 기법 연구												
	데이터셋 소스 탐색 및 선정												
	개발 환경 구축												
개발	베이스 모델 구축												
	학습용 데이터셋 전처리												
	중간보고서 작성												
	모델 학습 및 클리닝												
테스트	모델 테스트 및 최적화												
마무리	최종보고서 작성 및 발표												
	결과물 업로드 및 후속 처리												

조원별 수행 역할

학번	성명	구성원별 역할
201724512	유경민	- 학습용 데이터셋 탐색 - 딥러닝 기반 AptaNet 모델 연구 및 학습 모델 구축 - 학습 모델 파라미터 튜닝 및 최적화 - 보고서 및 발표 자료 작성
201724617	허수민	- 학습용 데이터셋 탐색 및 전처리 - 모델 평가 지표 선정 - 프로그램 개발환경 선정 및 구축 - 보고서 및 발표 자료 작성

출처 및 주석

[1] Shuanghui Yang, Huan Li, Ling Xu, Zhenhan Deng, Wei Han, Yanting Liu, Wenqi Jiang, Youli Zu, Oligonucleotide Aptamer-Mediated Precision Therapy of Hematological Malignancies, Molecular Therapy - Nucleic Acids, Volume 13, Pages 164-175, 2018.

[2] Yi Xi Wu, Young Jik Kwon, Aptamers: The “evolution” of SELEX, Methods, Volume 106, Pages 21-28, 2016.

[3] Alipanahi, B., Delong, A., Weirauch, M. et al. Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. Nat Biotechnol 33, 831-838, 2015.

[4] Emami, N., Ferdousi, R. AptaNet as a deep learning approach for aptamer-protein interaction prediction. Sci Rep 11, 6074, 2021.

[5] "extractProtPAAC function". RDocumentation 페이지. 2022년 10월 2일 접속, https://www.rdocumentation.org/packages/Rcpi/versions/1.8.0/topics/extractProtPAAC.

[6] Ding, Y. S., Zhang, T. L. & Chou, K. C. Prediction of protein structure classes with pseudo amino acid composition and fuzzy support vector machine network. Protein Pept. Lett. 14, 811-815, 2007.

[7] UniProt 메인 페이지, https://www.uniprot.org/ - 해당 사이트에서 단백질 정보를 검색할 수 있다.

Thank you!