
2022 전기 졸업과제 중간보고서

목차

목차	1
1. 기존 계획 및 추가 요구사항	2
1. 기존 목표	2
2. 추가 요구사항 및 대책	
- 데이터 전처리	2
- DeepBind	2
- Aptanet	3
2. 세부 프로그램 설계 명세 및 계획	4
1. 데이터 선정	4
2. 데이터 전처리	5
3. 대표모델 테스트	6
3. 갱신된 과제 추진 계획 및 진척도	17
1. 개발 프로세스 일정	17
2. 조원별 진척도	18
4. 현 시점까지의 중간결과	19
1. 데이터 전처리 샘플	19
2. 대표모델 테스트	19
출처 및 참조	21

1. 기존 계획 및 추가 요구사항

1. 기존 목표

압타머는 항체와 비교될 수 있을 정도의 높은 결합 친화도를 가지기 때문에 항체의 대체 물질로서 각광받고 있다. 다양한 후보 압타머들 중에 가장 결합 친화도가 높은 것을 선택하기 위해서, 본 과제에서는 컴퓨터 시뮬레이션 시스템을 이용하기로 했으며, 그 중에서도 MLP, CNN 등의 인공신경망을 적용한 예측 모델을 생성하되, 기존에 개발된 모델과 성능을 비교하여 모델 최적화를 수행할 계획이었다. 또한, 예측 과정에서 어떠한 변인이 주 영향을 끼치는지를 파악하고, 단백질에 결합할 가능성이 높은 압타머의 염기 서열 특징을 파악하여 일반화된 형태를 제시할 계획이었다.

현재 진행상황까지는 기존의 모델(DeepBind, AptaNet 등)을 토대로 개선된 모델을 제작하는 과정을 진행하고 있으며, 별도의 데이터를 수집하여서 다양한 종류의 API 결합에 대해서 학습 및 예측할 수 있도록 데이터 전처리도 진행중이다.

2. 추가 요구사항 및 대책

- 데이터 전처리

기존의 선례 연구를 토대로 API 데이터를 선정하고 전처리하는 과정에선 4종류의 정보(Alx1_DBD, ALX3_FL, AR_DBD, AR_FL)만을 사용하여 데이터 전처리를 수행하였다. 그러나, 이후 DeepBind 모델에서 사용했던 추가적인 데이터셋을 찾으면서 550종류 이상의 결합 정보를 얻게 되었다. 따라서 이러한 막대한 데이터를 일괄적으로 처리할 수 있도록 전처리 방식을 수정하게 되었다.

기존 요구사항	<p>대표 결합 정보들을 하나의 데이터셋으로 일괄적으로 통합한 뒤 한번에 전처리 작업을 수행</p> <ul style="list-style-type: none"> • One-Hot Vector 방식을 적용하여 염기서열 인코딩 • k-mer 지표를 활용한 염기서열 비율 정보 추가 • 단백질 정보를 고려하기 위해 PseAAC 정보를 추가 • 기타 파생변수를 생성하여 모델 학습 진행 • 불균형 데이터의 경우 데이터 샘플링 방식을 고려해서 진행
수정 및 추가 요구사항	<p>각 API 정보마다 개별 데이터셋으로 분리 및 전처리하고, 학습 직전 단계에서 데이터를 샘플링 & 통합하는 방식으로 진행</p> <p>One-Hot Vector 방식의 경우 1-mer 지표와 유사한 방식이므로 생략.</p> <p>데이터 불균형을 해소하기 위해 Oversampling과 Undersampling을 혼합하여 샘플링 진행 예정</p>

- DeepBind

기존 논문에 첨부된 code를 이용하기 위해서 리눅스 및 윈도우 환경에서 개발 환경 설정을 각각 시도했으나, 지속되는 원인 불명의 문제들로 인해 보류를 해두고, 우선 모델을 만드는 것에 집중하고 있다.

기존 요구사항	<p>기존의 모델을 토대로 개선된 모델 제작 진행</p> <ul style="list-style-type: none"> • deepbind 모델을 위한 환경 설정 • 간단한 테스트 후 매개변수 및 모델 층 조정 • 개선된 데이터 모델 도출
수정 및 추가 요구사항	<p>기존의 모델을 토대로 새로운 코드 작성 및 기존 코드 활용 방안 검토 병행</p> <p>input에 적절하도록 transformation 코드 작성</p>

- AptaNet

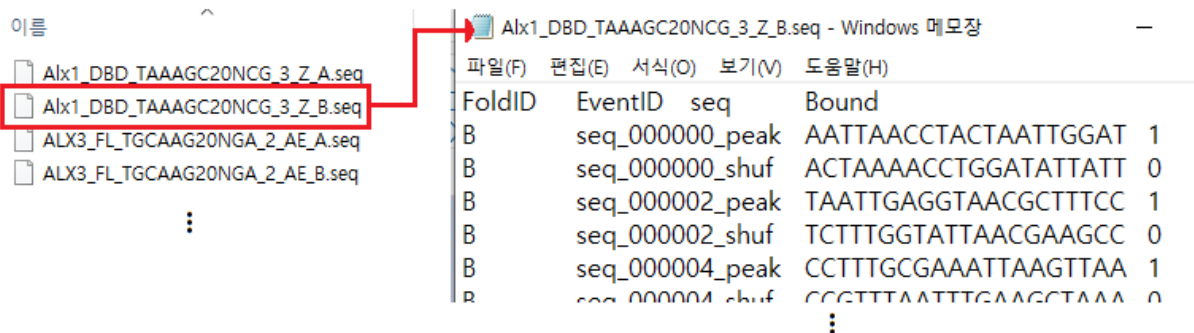
기존에는 기존 데이터 셋을 먼저 설정하고 실험을 진행하려 했으나 데이터 처리가 아직 진행 중이므로 k-mer만 적용된 데이터셋을 설정하고, 여러 비교 대상 모델들의 성능을 측정하였다. CNN모델을 기반으로 해서 응용 모델을 구축하려 하였으나 AptaNet의 구성 요소들을 수정하면서 더 나은 모델을 만들어냈다.

기존 요구사항	<p>흔히 알려진 모델들과 기존 AptaNet의 성능 평가 후 비교해서 더 나은 CNN 모델 구축</p> <ul style="list-style-type: none"> • 기존 데이터셋 설정 • 기존 성능평가지표 설정 • 여러 모델들의 성능 측정 • CNN 기반의 응용 모델 구축
수정 및 추가 요구사항	<p>AptaNet 모델을 기반으로 더 나은 모델 구축</p>

2. 세부 프로그램 설계 명세 및 계획

1. 데이터 선정

API 결합 데이터를 확보하기 위해 기존의 선례 연구에서 얻어낸 4가지 대표 API 정보 (Alx1_DBD, ALX3_FL, AR_DBD, AR_FL)를 토대로 데이터 병합 및 초기 전처리 과정을 진행하였다. API raw data의 경우 [그림 n]의 좌측 이미지처럼 .seq의 파일로 저장되어 있으며, 내부 데이터 형태는 [그림 n]의 우측과 같다. tab으로 각 값이 분리되어 있고, FoldID(A 그룹 or B 그룹), EventID(데이터 정보), seq(염기서열), Bound(API 결합 유무)에 해당하는 각 데이터가 나와있다. 이때, FoldID가 그룹 A인 경우는 전부 Bound가 1로만 이뤄져있으며, 그룹 B의 경우 1과 0이 고르게 나와있었다. 따라서 전처리 과정에선 API의 종류가 같으면 그룹 A, B와 상관없이 하나로 병합하여 데이터를 전처리하였다.



파일(F)	편집(E)	서식(O)	보기(V)	도움말(H)
FoldID	EventID	seq	Bound	
B	seq_000000_peak	AATTAACCTACTAATTGGAT	1	
B	seq_000000_shuf	ACTAAAACCTGGATATTATT	0	
B	seq_000002_peak	TAATTGAGGTAACGCTTTCC	1	
B	seq_000002_shuf	TCTTTGGTATTAACGAAGCC	0	
B	seq_000004_peak	CCTTTGCGAAATTAAGTTAA	1	
B	seq_000004_shuf	CCCTTTAATTGAAGCTAAA	0	

[그림 n] raw data의 형태(좌측)와 내부 기록 형태(우측)

4종류의 API 정보를 토대로 기본적인 전처리 코드를 작성하고 추가 데이터셋을 탐색하던 도중, DeepBind 모델에서 활용했던 원본 데이터셋을 추가로 얻어낼 수 있었다. 따라서 현재는 550종류 이상의 API 정보에 관해서 데이터 전처리 및 병합을 수행할 수 있도록 요구사항을 수정하고 코드 구조를 변경하는 중이다.

2. 데이터 전처리

먼저 [그림 n]의 raw data를 분리하여 DataFrame 생성 및 csv파일 형태로 정보를 저장하는 추출 코드를 작성하였다. 주어진 .seq 파일을 전처리하여 하나의 csv 파일로 제작하는 함수(merge_data)를 제작하였고, 그 이후엔 bound 예측을 위해 필요한 속성들을 추가하는 방식으로 전처리를 진행하였다.

주어진 데이터에서 염기 서열을 분리하는 과정에서 'A', 'C', 'G', 'T' 외에도 'N'이라는 별도의 항목이 존재하였다. 관련 정보를 탐색한 결과, 결측치(측정 X) 내지는 와일드 카드(A, C, G, T 모두 가능)라는 의미로 사용할 수 있는 문자임을 알아내었다. 따라서, N이라는 서열이 존재하는 경우에는 재귀적으로 A, C, G, T의 모든 문자열을 넣어서 API 정보를 복제(get_base_list, [코드 n] 참고)하는 방식으로 코드를 구성하였다.

```
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)
```

[코드 1] 염기서열의 N 정보를 처리하기 위한 get_base_list 함수

기본적으로 얻어낸 염기서열 정보에다가 k-mer 지표를 적용하기 위해 [코드 n]의 kmer 함수를 제작했고, 각 데이터마다 함수를 적용해 정보를 도출해내었다. seq는 A, C, G, T로 이뤄진 염기서열이며, k는 1 이상의 정수값이 들어간다. 최종 결과는 [그림 n]처럼 절대적인 개수가 아닌 상대적인 비율(전체 합 1)의 형태로 도출된다. 이번 과제에서는 k의 값이 1~4에 해당하는 정보를 얻어내었다.

```
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
```

[코드 2] k-mer 정보 생성을 위한 kmer 함수

	A	B	C	D	E	F	G	H	I	J	K
1	protein	seq	bound	A	C	G	T	AA	AC	AG	AT
2	Alx1_DBD	GCAGATAATCTAATTACCCC	1	0.35	0.3	0.1	0.25	0.105263	0.052632	0.052632	0.157895
3	Alx1_DBD	CTCAGTCCTCGTCTCGATGG	1	0.1	0.35	0.25	0.3	0	0	0.052632	0.052632
4	Alx1_DBD	TCATAATCTAATTACGCTCG	1	0.3	0.25	0.1	0.35	0.105263	0.052632	0	0.157895
5	Alx1_DBD	GACTTCCTCAATCTAATTAG	1	0.3	0.25	0.1	0.35	0.105263	0.052632	0.052632	0.105263
6	Alx1_DBD	GCAGTTAATCTAATTAACCG	1	0.35	0.2	0.15	0.3	0.157895	0.052632	0.052632	0.105263
7	Alx1_DBD	TCCAGGCCTGATTTAATTAA	1	0.3	0.2	0.15	0.35	0.105263	0	0.052632	0.105263

[그림 1] k-mer(k=1, ..., 4) 지표를 적용하여 얻어낸 결과 샘플

단백질 정보의 경우 PseAAC 정보를 활용하려 했다. 따라서 R언어에서 코드를 별도로 작성하여 Rcp1 라이브러리를 이용해 단백질 정보를 추출해내었다. [코드 n]과 같이 틀을 구성하였으며, entry에는 원하는 단백질의 고유 라벨 번호를 기입했고, label에는 해당 단백질의 이름을 넣어주면 protein_data.csv라는 정보를 얻어낼 수 있다. 현재 코드의 단백질 정보는

UniProtTM의 DB 정보를 기반으로 얻어내고 있다. 해당 정보의 샘플 형태는 문서 뒷 파트에 첨부한 [그림 n]과 같이 나온다.

```
# 기본 모듈 설치 (설치되어 있는 경우 생략 가능)
install.packages("BiocManager")
BiocManager::install("Rcpi", dependencies = c("Imports", "Enhances"), force=TRUE)
BiocManager::install("GenomeInfoDb")
BiocManager::install("XVector")
BiocManager::install("bit")
BiocManager::install("Biobase")
BiocManager::install("GO.db")
BiocManager::install("rcdklibs")

# 라이브러리 불러오기
library("Rcpi")

# 샘플 정보 순서: Alx1( Mus musculus ), AXR ( Homo sapiens ),
# ALX3 ( Homo sapiens ), AR_FL ( Homo sapiens )
entry = c('Q8C8B0', 'Q96QS3', 'O95076', 'P10275')
label = c('Alx1', 'ARX', 'ALX3', 'AR_FL')

rslt_df = c()
for(i in 1:length(entry)){
  nowe = entry[i]
  nowl = label[i]

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

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

  rslt_df = rbind(rslt_df, trsplt)
}

directory_path = '/* 파일 저장 경로명 작성 */'
setwd(directory_path)
write.csv(rslt_df, file='protein_data.csv', quote=FALSE)
```

[코드 3] PseAAC 정보를 도출하기 위한 Rcpi 활용 R 코드

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		Xc1.A	Xc1.R	Xc1.N	Xc1.D	Xc1.C	Xc1.E	Xc1.Q	Xc1.G	Xc1.H	Xc1.I	Xc1.L	Xc1.M
2	Alx1	5.777514	5.777514	4.727057	3.6766	1.575686	5.777514	3.939214	4.464442	3.151371	1.575686	6.302742	4.911371
3	ARX	29.79562	9.746231	1.113855	6.68313	1.949246	13.92319	4.45542	12.80933	2.784638	1.949246	15.87243	5.511371
4	ALX3	7.778015	5.555725	2.22229	3.333435	1.666717	5.277939	3.333435	9.444732	3.333435	1.388931	8.333587	4.911371
5	AR_FL	22.95207	11.61771	5.667177	10.48428	7.650689	15.58474	20.11848	27.20245	5.383818	6.517254	24.65222	11.61771

[그림 2] 해당 단백질의 PseAAC 정보 샘플

이후엔 k-mer 정보의 PseAAC 정보를 하나로 결합하여 하나의 데이터셋을 완성하였다. 단일 데이터셋으로 합치는 과정에서 메모리 초과 현상을 방지하기 위해서 데이터를 부분마다 나눠서 파일로 저장하였으며, 나중에 일괄적으로 데이터를 합쳐서 단일 셋의 형태로 완성하였다. 샘플 개형은 [그림 n]에 나와있다.

또한, 추가로 간단한 EDA 과정을 진행하였으며, bound 및 단백질 종류에 따른 데이터 비율 정도를 간단하게 살펴봤다. [그림 n (a)]의 정보는 bound의 비율이며, 1이 ⅔ 정도를 차지하고 있다. [그림 n (b)]는 단백질 데이터에 따른 비율이며, 다른 단백질 비율과 달리 AR_FL 정보가 상대적으로 적은 것을 알 수 있다. 마지막으로 [그림 n (c)]는 bound와 protein 정보를 고려한 비율이다.

Out [7]: bound 1 66.667432 0 33.332568 dtype: float64	Out [12]: protein A1x1_DBD 33.890823 ALX3_FL 31.299555 AR_DBD 30.546502 AR_FL 4.263120 dtype: float64	Out [13]: bound protein 1 A1x1_DBD 22.600417 ALX3_FL 20.860925 AR_DBD 20.364394 0 A1x1_DBD 11.290407 ALX3_FL 10.438630 AR_DBD 10.182108 1 AR_FL 2.841697 0 AR_FL 1.421423 dtype: float64
(a) bound only	(b) protein only	(c) considering bound & protein info.

[그림 3] 현재까지 제작된 데이터셋의 EDA 정보

다음으로, 파생변수를 제작하려던 찰나에 추가적인 데이터셋을 확보하게 되었고, 추가로 받은 데이터셋을 앞선 과정을 거쳐서 하나의 단일 데이터로 만들 생각이었다. 그러나, 데이터의 양이 너무 방대해서 기존처럼 하나의 데이터로 합치는 방식으로 진행하지 못한다는 문제점에 직면했다. 따라서, 방식을 바꿔 각 단백질 정보들마다 별도의 DataFrame 정보를 형성해서 저장하고, 학습 및 예측용 dataset을 생성하는 경우에만 별도의 데이터 생성 함수를 제작하여 데이터 샘플링 및 병합하는 방식을 쓰겠다고 결정했다. 본 문서 뒷부분에 있는 [그림 n]이 이러한 모습을 보여주고 있다.

그런데, 앞선 방식으로 550종류 이상의 API 정보 내 염기서열을 처리하면서, 기존 4종류 API의 전처리 과정에서 보이지 않던 재귀함수의 메모리 초과 현상이 발생하였다. 원인을 찾아본 결과, [그림 n]과 같이 일부 염기서열에서 N 정보가 6개 이상은 물론, 심한 경우 11개 이상 출현하는 경우도 있었다. 해당 염기서열을 복제하게 될 경우 4¹¹개, 즉 약 사백 십만개가량의 불필요한 데이터를 생성해야 한다는 문제점이 존재했다. 따라서, 염기서열의 중첩 복제를 허용하는 경우는 최대 N이 5개까지 존재하는 서열(약 1000개까지 복제 허용)에 관해서만 적용했고, 그 이상 N이 존재하는 염기서열은 사용하지 않았다.

```

In [14]: bound_df = merge_data()
          print()
          #print(bound_df.head())
          #bound_df.to_csv('raw_data.csv', index=False)

BarH11_DBD#TCAGTC2ONGG_3_AC clear!
BARHL2_DBD#TATTGT2ONGTG_2_AC progressing...
BARHL2_DBD#TATTGT2ONGTG_2_AC clear!
BARHL2_DBD#TATTGT2ONGTG_3_AC progressing...
BARHL2_DBD#TATTGT2ONGTG_3_AC clear!
BARHL2_FL#TCCAGT4ONGAC_3_A1 progressing...
=> Removed: TCATCTAAGGACGATTAGTTTTAACNCNCTAGNNNNNGGGA
=> Removed: CTCGAGAGCATAGTTTATATAGATGNNTTNCNNNNNGGAC
=> Removed: CATGAAAATGCGAAAGGAAGAAGCGANNGAGGNNNNTTGG
=> Removed: CTCCTTTTGCAGGGCACACTCAATNANGAGACNGNNCNTN
=> Removed: AAACGCTTTAACATACGAATGTTANANCAAGTNCNATNAN
=> Removed: TGTAATGAATTAACGAGAACACTNANTGCATNANTTNGN
=> Removed: TTAANCATAATTAATAGAGNAGNCNNNCNAGTNTNNANAN

```

[그림 4] N이 과도하게 존재하는 염기서열 정보의 일부

현 시점까진 550종류의 API 정보를 분리하고 개별 저장하는 과정까지 마무리했다. 이후 기존에 4종류의 API를 전처리하는 과정과 동일하게 ‘염기서열에서 k-mer 지표 적용 & 단백질 결합정보 추출 -> 추출된 API 정보에 결합’하는 과정을 진행할 계획이다. 또한 별도의 데이터 샘플링 및 병합용 메소드를 제작하여서 향후 모델 학습 시 데이터를 쉽게 샘플링할 수 있도록 제작할 계획이다.

3. 대표모델 테스트


```

def testModel(x,y,num):
    kfold = KFold(n_splits=10,shuffle=True,random_state=7)
    n_iter = 0
    cv_f1_score = []

    for train_idx, test_idx in kfold.split(x):
        c = createModel(x,num)

        X_train, X_test = x[train_idx], x[test_idx]

        y_train, y_test = y[train_idx], y[test_idx]
        if num == 2 or num == 3 or num == 5:
            c.fit(X_train,y_train)

        else:
            c.fit(X_train,y_train,epochs=100)

        fold_pred = c.predict(X_test)
        n_iter += 1
        #print(y_test)
        #print(fold_pred)
        c_f1_score = np.round(f1_score(y_test,fold_pred>0.5),4)
        # print(accuracy_score(y_test,fold_pred>0.5))
        print(c_f1_score)
        print('\n{} 교차검증 정확도 : {}, 학습 데이터 크기 : {}, 검증 데이터 크기 :
        {}'.format(n_iter,f1_score,X_train.shape[0],X_test.shape[0]))
        cv_f1_score.append(c_f1_score)

    print('\n average f1_score : ',np.mean(cv_f1_score))
    return np.mean(cv_f1_score)

def getResample(x,y):
    clf = RandomForestClassifier(n_estimators=300, max_depth=9,
                                random_state=0)

    clf.fit(x, y)
    #clf.feature_importances_
    model = SelectFromModel(clf, prefit=True)
    x = model.transform(x)
    ncr = NeighbourhoodCleaningRule()
    x_resampled, y_resampled = ncr.fit_resample(x, y)
    return x_resampled, y_resampled

def createModel(x,num):
    if num == 0:
        #Multi Layer Perceptron Model
        model = Sequential()
        opt=RMSprop(lr=0.00014, rho=0.9, epsilon=None, decay=0.0)
        model.add(Dense(128, input_dim=x.shape[1], activation='relu'))
        tensorflow.keras.layers.AlphaDropout(0.3, noise_shape=None, seed=None)

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

```

[코드 4] 모델 테스트 코드

AdaBoost (DT)

```
1 ga.testModel(x_resampled,y_resampled,2)

0.8772

1 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.8098

2 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.8408

3 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.9091

4 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.8913

5 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.8933

6 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.8929

7 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.8734

8 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
74, 검증 데이터 크기 : 263
0.8861

9 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
74, 검증 데이터 크기 : 263
0.8

10 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 2
374, 검증 데이터 크기 : 263

average f1_score : 0.86739
```

[그림 5] AdaBoost 결과

Gradient Boosting

```
1 ga.testModel(x_resampled,y_resampled,3)
0.8

1 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.8293

2 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.7857

3 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.8408

4 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.807

5 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.8571

6 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.875

7 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.7943

8 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
74, 검증 데이터 크기 : 263
0.7862

9 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
74, 검증 데이터 크기 : 263
0.7941

10 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 2
374, 검증 데이터 크기 : 263

average f1_score : 0.81695
```

[그림 6] Gradient Boosting 결과

XGBoost

```
1 ga.testModel(x_resampled,y_resampled,5)

0.8929

1 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.9186

2 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.8701

3 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.8696

4 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.8962

5 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.9067

6 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.8916

7 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
73, 검증 데이터 크기 : 264
0.9161

8 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
74, 검증 데이터 크기 : 263
0.9125

9 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 23
74, 검증 데이터 크기 : 263
0.9067

10 교차검증 정확도 : <function f1_score at 0x7fea4eb48a60>, 학습 데이터 크기 : 2
374, 검증 데이터 크기 : 263

average f1_score : 0.8981
```

[그림 7] XGBoost 결과

```
1 ga.testModel(x_resampled,y_resampled,0)

75/75 [=====] - 0s 2ms/step - loss: 0.0394 - accuracy: 0.9890
Epoch 95/100
75/75 [=====] - 0s 2ms/step - loss: 0.0354 - accuracy: 0.9882
Epoch 96/100
75/75 [=====] - 0s 2ms/step - loss: 0.0494 - accuracy: 0.9848
Epoch 97/100
75/75 [=====] - 0s 2ms/step - loss: 0.0278 - accuracy: 0.9916
Epoch 98/100
75/75 [=====] - 0s 1ms/step - loss: 0.0406 - accuracy: 0.9865
Epoch 99/100
75/75 [=====] - 0s 2ms/step - loss: 0.0391 - accuracy: 0.9878
Epoch 100/100
75/75 [=====] - 0s 2ms/step - loss: 0.0380 - accuracy: 0.9882
0.831

10 교차검증 정확도 : <function f1_score at 0x7f8e18d49a60>, 학습 데이터 크기 : 2374, 검증 데이터 크기 : 263

average f1_score : 0.8307

0.8307
```

[그림 8] AptaNet 결과

DeepBind

우선, 베이스 모델 구축을 위해서 논문에 첨부된 코드와 데이터를 활용하고자 했으나 환경 설정 등에 다소 어려움을 겪었고 보류해둔 상태이다. 윈도우 및 맥 환경에서 지속적으로 시도중이지만 해당 문제를 해결하지 못한다면, 논문에 존재하는 스코어들을 활용할 예정이다.

이와 별개로 새로운 모델을 생성하기 위해서 최소한의 레이어로 데이터 적용과 학습 경험 자체에 초점을 맞춰 코드를 작성했다. 학습은 많은 횟수가 큰 의미가 없을 것으로 예상해 20회만 진행했다.

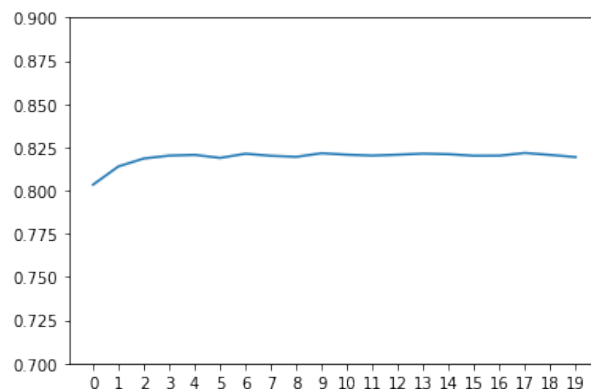
Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 34, 16)	336
re_lu (ReLU)	(None, 34, 16)	0
max_pooling1d (MaxPooling1D)	(None, 1, 16)	0
flatten (Flatten)	(None, 16)	0
dropout (Dropout)	(None, 16)	0
dense (Dense)	(None, 1)	17

=====
 Total params: 353
 Trainable params: 353
 Non-trainable params: 0

[그림 9] DeepBind 모델 개요

아직 모델의 뼈대만 세운 상태이므로 f1-score가 큰 의미가 없으나 해당 스코어를 측정 한 값은 다음과 같다.



[그림 10] DeepBind 결과

3. 갱신된 과제 추진 계획 및 진척도

1. 갱신된 개발 프로세스 일정

구분	추진내용	추진일정 (월별, 상/하 구분)											
		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	생성, 판별용 베이스모델 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												

[표 1] 기존 수행 일정 및 향후 개발 일정에 대한 Gantt chart

2. 현재까지 조원별 진척도

구분	조원	담당업무
1	김유진	<p>개발환경 설정 및 기존 논문 연구</p> <ul style="list-style-type: none"> • Deepbind 관련 레퍼런스 참고 및 Deepbind 논문 분석 • 베이스 모델 구축을 위해 리눅스 및 윈도우 환경에서 개발 환경 설정을 시도했으나 다양한 문제로 보류 <p>간단한 모델과 기존 논문 데이터를 활용한 테스트</p> <ul style="list-style-type: none"> • 논문 제공 데이터를 이용한 train • 실제 데이터 적용을 위한 테스트 코드 작성 중
2	유경민	<p>데이터셋 결정 및 성능 지표 결정</p> <ul style="list-style-type: none"> • k-mer(k=1, 2, 3, 4)를 적용한 임시 데이터셋을 기준 • f1-score를 기준으로 모델 성능 평가 <p>모델 테스트 및 구축</p> <ul style="list-style-type: none"> • AptaNet, AutoML, AdaBoost(DT), Gradient Boosting, XGBoost를 이용하여 성능 평가 • AptaNet을 수정하여 조금 더 나은 성능의 모델을 만들어냄 • 모델만 input으로 넣으면 바로 10-fold validation이 실행되고 평균 f1-score가 나오도록 모듈 구축
3	허수민	<p>데이터셋 선정 및 이해</p> <ul style="list-style-type: none"> • Deepbind 모델 구성 시 사용했던 API data로 선정 및 활용 • API raw 데이터에서 Dataframe 형태로 변환 <p>개발환경 선정 및 활용기술 파악</p> <ul style="list-style-type: none"> • Jupyter notebook 상에서 Tensorflow + Keras를 활용하기로 결정 • k-mer 지표, PseAAC 정보 이해 <p>데이터 전처리</p> <ul style="list-style-type: none"> • k-mer(k=1, 2, 3, 4)를 적용하여 염기서열 전처리 • R에서 Rcpa를 이용하여 PseAAC 정보 추출 및 결합 • F1-score를 기반으로 평가지표 통일 • 기본 정보를 바탕으로 간단한 EDA 수행

[표 2] 조원별 진척도

4. 현 시점까지의 중간결과

1. 데이터 전처리 파트

	A	B	C	D	E	F	G	H		ME	MF	MG	MH
1	protein	seq	bound	A	C	G	T	AA		TTTT	Xc1.A	Xc1.R	Xc1.N
2	Alx1_DBD	GCAGATAATCTAATTACCCC	1	0.35	0.3	0.1	0.25	0.105263		0	5.777514	5.777514	4.727057
3	Alx1_DBD	TTAAACACTAATGTAATTAA	1	0.5	0.1	0.05	0.35	0.263158		0	5.777514	5.777514	4.727057
4	Alx1_DBD	TATTATACAAAATTGCACTT	0	0.4	0.15	0.05	0.4	0.157895		0	5.777514	5.777514	4.727057
5	Alx1_DBD	CCCGAATTACATTGAATTAC	1	0.35	0.25	0.1	0.3	0.105263		0	5.777514	5.777514	4.727057
6	ALX3_FL	GGAAGCCAGATAAATACAGC	1	0.45	0.2	0.25	0.1	0.157895	0	7.778015	5.555725	2.22229
7	ALX3_FL	GCCATAGCCCCGTTTGCCAC	1	0.15	0.45	0.2	0.2	0		0	7.778015	5.555725	2.22229
8	ALX3_FL	CCCCGGTCCCCCAATGAA	0	0.2	0.55	0.15	0.1	0.105263		0	7.778015	5.555725	2.22229
9	AR_DBD	ACAAACCTCACCTATATCC	1	0.35	0.45	0	0.2	0.105263		0	29.79562	9.746231	1.113855
10	AR_DBD	ATGGGCCCGCTAAGAACCA	0	0.3	0.35	0.25	0.1	0.105263		0	29.79562	9.746231	1.113855
11	AR_DBD	GGACACTCTGTACCACAGGC	1	0.25	0.35	0.25	0.15	0		0	29.79562	9.746231	1.113855
12													

[그림 11] API 정보에 따른 전처리 결과 sample data

Alx1_DBD#TAAAGC20NCG_3_Z.csv	ATF7_DBD#TGGGCG30NCGT_4_AI.csv	BHLHB3_FI
ALX3_DBD#TGTAAG20NAAG_2_Z.csv	Atoh1_DBD#TAGTCC20NCTT_3_Z.csv	BHLHE22_I
ALX3_FL#TGCAAG20NGA_2_AE.csv	Barhl1_DBD#TCAGTC20NCG_2_AC.csv	BHLHE23_I
Alx4_DBD#TGGTAG20NCG_2_P.csv	Barhl1_DBD#TCAGTC20NCG_3_AC.csv	BHLHE41_I
ALX4_DBD#TGTC20NGA_2_W.csv	BARHL2_DBD#TATTGT20NGTG_2_AC.csv	BSX_DBD#
Ar_DBD#TCTAAT20NCG_4_P.csv	BARHL2_DBD#TATTGT20NGTG_3_AC.csv	CART1_DBD
AR_DBD#TGCTCG20NGA_3_AF.csv	BARHL2_FL#TCCAGT40NGAC_3_AI.csv	CDX1_DBD
AR_FL#TCTTCT20NCTG_4_AD.csv	BARHL2_FL#TCCAGT40NGAC_4_AI.csv	CDX2_DBD

[그림 12] 활용할 데이터셋을 분리한 결과 일부분

2. 모델 구축 및 학습 파트

- 기존 Aptanet 모델을 수정하여 f1-score 기준 더 나은 모델 도출
- 기존 Aptanet 모델 (f1-score: 0.8307) 대비 성능 개선

```

1 ga.testModel(x_resampled,y_resampled,0)
75/75 [=====] - 0s 4ms/step - loss: 0.0328 - accuracy: 0.9878
Epoch 95/100
75/75 [=====] - 0s 5ms/step - loss: 0.0294 - accuracy: 0.9890
Epoch 96/100
75/75 [=====] - 0s 4ms/step - loss: 0.0332 - accuracy: 0.9890
Epoch 97/100
75/75 [=====] - 0s 4ms/step - loss: 0.0260 - accuracy: 0.9903
Epoch 98/100
75/75 [=====] - 0s 4ms/step - loss: 0.0270 - accuracy: 0.9912
Epoch 99/100
75/75 [=====] - 0s 5ms/step - loss: 0.0336 - accuracy: 0.9890
Epoch 100/100
75/75 [=====] - 0s 4ms/step - loss: 0.0266 - accuracy: 0.9895
0.8125

10 교차검증 정확도 : <function f1_score at 0x7fcbf8448a60>, 학습 데이터 크기 : 2374, 검증 데이터 크기 : 263
average f1_score : 0.8484400000000001
- : 0.8484400000000001

```

[그림 13] Aptanet 응용 모델 학습 결과 (f1-score : 0.848)


```
opt=RMSprop(lr=0.0007, rho=0.9, epsilon=None, decay=0.0)
#Multi Layer Perceptron Model
model = Sequential()
#model.add(Dense(128, input_dim=639, activation='relu'))

model.add(Dense(512, input_dim=x.shape[1], activation='relu'))
tensorflow.keras.layers.AlphaDropout(0.5, noise_shape=None, seed=None)

model.add(Dense(256, activation='relu'))
tensorflow.keras.layers.AlphaDropout(0.5, noise_shape=None, seed=None)

model.add(Dense(128, activation='relu'))
tensorflow.keras.layers.AlphaDropout(0.5, noise_shape=None, seed=None)

model.add(Dense(128, activation='relu'))
tensorflow.keras.layers.AlphaDropout(0.5, noise_shape=None, seed=None)

model.add(Dense(128, activation='relu'))
tensorflow.keras.layers.AlphaDropout(0.5, noise_shape=None, seed=None)

model.add(Dense(128, activation='relu'))
tensorflow.keras.layers.AlphaDropout(0.5, noise_shape=None, seed=None)

model.add(Dense(128, activation='relu'))
tensorflow.keras.layers.AlphaDropout(0.5, noise_shape=None, seed=None)

model.add(Dense(32, activation='relu'))
tensorflow.keras.layers.AlphaDropout(0.5, noise_shape=None, seed=None)

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

[그림 14] 응용 모델 구조

- DeepBind 뼈대 코드 작성 및 Epoch=20일 때, 평균 f1 score 도출

```
r = np.array(history.history['recall'])
p = np.array(history.history['precision'])

f1 = 2 * (r * p) / (r + p)

print(f1.mean())

0.8191635184353718
```

[그림 15] DeepBind 최소 레이어 모델 학습 결과 (f1-score: 0.8191653)

출처 및 참조

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

[2] 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.
