과제번호*	2019 과학영재 창의연구(R&E) 과제 결과보고서 요약

연구과제명	생명의료 데이터 분석 정확도 향상을 위한 데이터 마이닝 기법 개발 (세부주제 : PPI 예측 모델을 이용한 알레르겐에서의 에피토프 부위 예측) Designing data mining approaches for improving the accuracy of biomedical big data analysis (Epitope prediction in allergens using PPI prediction model)		
연구분야	정보과학	연구비	○,○○○,○○○원
지도교사	박소영	참여학생 수	4명
공동연구원	송길태 부산대학교	참여학생 명단	배민성, 정재현 정현서, 강신재
연구의 필요성 및 목 적 (요약)	- 오픈소스 머신러닝 프레임워크 및 대용량 생명의료 정보 데이터를 활용한 데이터 마이닝 기법과 빅데이터 분석 시스템 연구 - 다양한 알레르기 반응에 관여하는 항원들의 에피토프 부위 예측을 통한 항체-항원 반응 여부 예측 및 새로운 에피토프 발견에 대한 가능성 제시		
연구방법 / 연구내용 (요약)	- **학습용 데이터셋 구축** : Yeast PPI 데이터셋 구축 (아미노산 분류를 통한 CTF 기법 적용 및 Positive/Negative Pair로 구성을 위한 무작위 표본 추출) - **테스트용 데이터셋 구축** : 알레르겐 단백질들의 정보 (아미노산 및 에피토프 서열 등) 데이터셋 구축 알레르겐에 대응되는 IgE 항체들의 서열 데이터셋 구축 - **학습 모델 제작** : scikit-learn 라이브러리를 활용한 머신 러닝 모델 제작 keras 라이브러리를 활용한 딥 러닝 모델 제작 각 모델 별 성능 평가 (accuracy / precision – recall) - **에피토프 서열의 예측 :** Random Forest 모델에서 treeinterpreter 라이브러리를 활용한 각 feature의 기여도 추출 테스트 데이터의 에피토프 예측 및 실제 에피토프와의 비교		
향후 연구계획 (요약)	- CTF의 고정 길이 변화 및 아미노산의 카테고리화(차원축소) 없이 데이터셋 가공 - 인간의 PPI 데이터셋으로 학습 진행 - 아미노산 서열이 형성하는 단백질 구조 특징을 학습 데이터, 알고리즘에 추가 - 두 단백질의 아미노산 서열을 일렬로 붙이지 않고 다른 방식으로 학습 진행		
주제어 (5개 이내)	알레르겐, 에피토프, PPI(Protein-Protein Interaction), Python, 데이터 마이닝		

PPI 예측 모델을 이용한 알레르겐에서의 에피토프 부위 예측

배민성 · 강신재 · 정재현 · 정현서
부산과학고등학교

Epitope prediction in allergens using PPI prediction model

Bae Min-Sung · Kang Sin-Jae · Jung Jae-Hyun · Jung Hyun-Seo
Busan Science High School

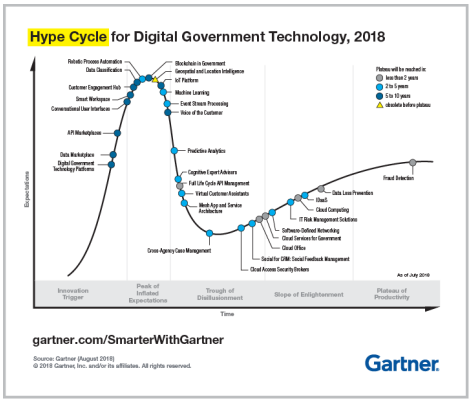
국문 초록

현재 컴퓨터공학 기술의 발달로 오픈소스 머신러닝·딥러닝 프레임워크 및 대용량 생명의료 정보 데이터를 활용한 데이터 마이닝 기법과 빅데이터 분석 시스템 연구에 대한 가능성이 꾸준히 제시되고 있다. 이에 우리는 관련 데이터와 머신러닝 프레임워크를 이용하여 Python으로 다양한 알레르기 반응에 관여하는 항원의 항원-항체 반응 여부 및 에피토프 부위가 알려지지 않은 알레르겐의 에피토프 서열을 예측해 보고자 한다. 대부분의 항원과 항체는 단백질이므로 항원-항체 반응 또한 단백질-단백질 상호작용의 일종인 점을 응용하여 단백질-단백질 상호작용 데이터를 통해 두 단백질의 아미노산 서열이 주어졌을 때, 단백질의 결합 여부를 예측하는 모델을 구축한다. 이 모델을 통해 우리는 특정 항체와 항원의 아미노산 서열이 주어졌을 때 항원·항체의 결합 여부를 예측한다. 만약 항원과 항체가 결합하는 경우, 항체가 항원에 결합하는 부위인 항원의 에피토프 서열을 예측한다.

주제어: 알레르겐, 에피토프, PPI(Protein-Protein Interaction), Python, 데이터 마이닝

Ⅰ. 서론

인공지능(Artificial Intelligence)과 딥러닝(Deep Learning)은 통신 기술의 발전 및 컴퓨터 계산 성능의 향상으로 인해 빠르게 연구가 이루어져 나가며 빅 데이터의 패턴 발견 및 예측 등을 위한 분석 기술의 하나로서 주목받고 있다. 글로벌 IT 리서치 기관인 가트너(Gartner)가 발표한 <2018 Gartner Hype Cycle>[1]에 따르면 인공지능과 딥러닝 기술은 최근 사회 핵심 기술로서 작용하며 전반적인 산업 분야에서 뚜렷한 활용 가치를 보여주고 있다.



[Fig. 1] 2018 Gartner Hype Cycle

1) [Fig. 1] 출처 : https://www.gartner.com/smarterwithgartner/5-trends-appear-on-the-gartner-hype-cycle-for-emerging-technologies-2019/

또한 분자생물학 분야에서는 생명공학 기술의 거듭된 발전으로 유전자 데이터 및 바이러스, 암의 극복 및 치료를 위한 여러 실험 데이터들이 생산되고 있으며 이를 효율적으로 분석하기 위한 기술의 필요성이 대두되고 있다. 현재 컴퓨터 공학의 인공지능 기반 데이터마이닝 및 패턴 클러스터링 기술을 통한 빅데이터 분석 연구는 이미 실제 상용화가 가능할 정도로 발전하였으며, 이에 인공지능 기반 기술의 다양한 타 분야 적용 가능성이 계속해서 제시 되고 있는 상황이다. 하지만 이러한 상황에도 데이터마이닝 기술을 생명의료 분야에 빠르게 적용하고 유의미한 분석 결과를 도출하는 연구는 아직 소수에 불과하며, 우리는 이러한 분야의 높은 진입 장벽을 허물기 위해 생명의료 빅데이터와 인공지능 기술을 접목한 시스템의 연구가 필요하다고 판단했다.

알레르기(Allergy)는 체내에서 일어나는 항원-항체 반응의 일종으로 알레르겐(allergen)이라고 불리는 특정 단백질이 항원으로 작용하여 재채기, 콧물, 눈물, 과민성 쇼크 등 다양한 증상을 일으킨다. 알레르기는 많은 사람들에게서 다양한 알레르겐에 의해 나타나는 현상으로서 주변에서 쉽게 찾아볼 수 있는 질병 중 하나이다. 그러나 현재까지 어떤 단백질이나 생물학적 물질이 체내에서 알레르겐으로 작용하는지가 모두 밝혀지지 않았을 뿐만 아니라, 알레르겐의 어떤 부분이 항체와 결합하는지, 즉 알레르겐의 에피토프(epitope) 부위가 어디인지 밝혀지지 않은 알레르겐이 다수 존재한다. 이에 우리는 어떤 알레르겐에 대하여 그 알레르겐의 에피토프 부위를 예측할 수 있는 연구의 필요성을 느꼈다.

따라서 우리는 알레르기 반응, 즉 항원-항체 반응이 단백질 간의 상호작용이라는 점을 활용하여, 기존의 PPI (Protein-Protein Interaction) 데이터베이스 및 알레르겐・항체 서열 데이터를 가공・이용해 머신러닝 및 딥러닝 모델을 제작한 후 알레르겐의 에피토프를 예측하는 연구에 활용하고자 한다.

Ⅱ. 이론적 배경

1. 항원-항체 반응(Antigen-Antibody Reaction)

1.1. 항원-항체 반응

항원-항체 반응[2]은 체내에서 일어나는 면역 반응의 일종으로. 항체와 항원 간의 화학적 상호작용을 말한다.

항원(Antigen)은 체내에서 면역 반응을 일으키는 물질로, 체내에서 어떤 물질이 항원제시세포에 의하여 항원으로 인식되면 형질B세포가 항체를 제작하여 분비한다. 보통 병원체 및 병원체가 분비하는 독소, 세균, 바이러스 껍질 등의 단백질이 항원으로 작용하며, 인공적으로 합성되거나 외부에서 유입된 물질도 체내에서 항원으로 작용할 수 있다.

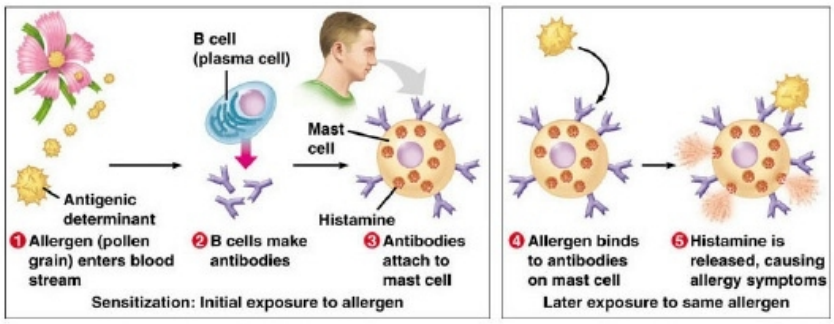
항체(Antibody)는 체내에 들어오는 항원에 대항하는 물질로서 항원에 대한 특이적 결합을 한다. 즉, 항체는 오직 그 항체를 만들게 한 항원에만 결합한다. 항체는 면역글로불린(Immunoglobulin, Ig)라고도 하며 면역글로불린은 항체로서 작용하는 당단백질이다. 기본 구조는 Y자 형태의 단백질이며, 중쇄와 경쇄로 구성되어 있다. 중쇄의 불변 영역에 따라서 항체는 IgG, IgA, IgM, IgD, 그리고 IgE 5가지의 동형상으로 분류된다.

[Fig. 2] 항원-항체 반응의 모식도

1.2. 알레르기

2) [Fig. 2] 출처 : https://news.v.daum.net/v/20180520130303171?f=m

알레르기(Allergy)[3]는 알레르겐(Allergen)이라고 불리는 특정 항원에 대한 과도한 면역 반응이다. 알레르겐에 의한 초기 노출 이후 체내에서는 그 알레르겐에 대한 항체가 생성된다. 이때 알레르기 반응에 관여하는 항체는 IgE 계열이다. 생성된 항체는 후미 부분을 통하여 결합조직에 존재하는 비만세포에 결합하게 된다. 이후 동일한 알레르겐에 인체가 재노출되면 비만세포 표면에 결합된 IgE 항체와 알레르겐 사이에 항원-항체 반응이 일어나게 되고, 표면의 항체들이 서로 연결되어 뭉치게 된다. 이렇게 되면 비만세포는 세포내 과립에 저장되어 있던 혈관 확장을 유발하는 물질인 히스타민과 다양한 염증 유발 물질을 분비하게 되고, 이로 인해 재채기, 콧물, 눈물 및 호흡 장애를 일으키게 하는 평활근 수축이 일어난다. 전신에서 일어나는 급성 알러지 반응은 아나필라시스 쇼크를 유발할 수 있고, 이는 사람을 사망에 이르게까지 할 수 있다.



[Fig. 3] 알레르기 반응의 모식도

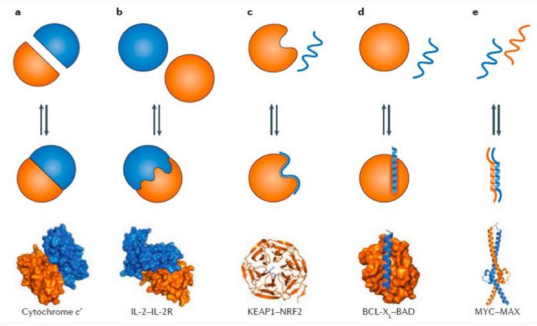
2. 단백질-단백질 상호작용(Protein-Protein Interaction, PPI)

2.1. 단백질-단백질 상호작용

단백질-단백질 상호 작용은 소수성 상호작용을 포함한 정전기력에 의해 조절되는 생화학적 반응의 결과로 두 개 이상의 단백질 분자 간에 확립된 높은 특이성의 결합이다. 대부분은 세포 또는 살아있는 유기체에서 존재하는 폴리펩타이드 사슬 사이의 물리적 접촉이다. 단백질은 여러 가지 기능들을 수행하기 때문에 세포 내에서 단독으로 작용하는 경우가 드물다. 따라서 세포 내 많은 과정은 PPI에 의해 조직된 단백질 성분으로 만들어진 분자에 의해 수행된다.

2.2. 단백질-단백질 상호작용의 5가지 양상[4]

- 두 개의 단백질이 서로의 입체구조를 변화시키지 않은 채 결합
- 두 개의 단백질이 결합하면서 인터페이스의 구조를 변화시킴
- 구조가 변하지 않는 단백질에 다른 단백질의 disordered 된 영역이 상대 단백질의 모양에 맞추어서 결합됨
- 구조가 변하지 않는 단백질에 다른 단백질의 a-helix 등의 2차구 조가 2차 구조를 유지하며 결합
- 두 개의 단백질이 공통된 2차 구조간의 상호작용을 통해 coiled-coil 등을 형성하며 결합함



[Fig. 4] 단백질 상호작용의 5가지 양상 모식도

3) [Fig. 3] 출처 : https://www.gastroepato.it/atopia_allergia.html
4) [Fig. 4] 출처 : http://wavefunction.fieldofscience.com/2016/08/protein-protein-interactions-cant-live.html

3. 머신러닝(Machine Learning)

3.1. 머신러닝

머신러닝이란 컴퓨터가 주어진 데이터를 통하여 컴퓨터가 수행해야 할 작업을 스스로 학습하도록 하게 만드는 기술이다. 기존의 컴퓨터 사용이 사용자가 제시하는 처리 방법에 대한 빠른 속도의 연산을 위함이었다면, 머신러닝에서는 사용자가 컴퓨터에게 원하는 결과 샘플, 즉 관련 데이터들을 제공하고 그에 합당한 처리 방법을 컴퓨터 스스로 찾길 원한다. 머신 러닝에는 입력 값으로부터 데이터들 간의 함수를 유추하는 지도 학습(supervised learning), 입력 값에 대한 목표치가 주어지지 않으며 주어진 입력에 대한 확률밀도를 모델링하는 비지도 학습(unsupervised learning), 현재 상태에서 외부에서 주어지는 보상에 대한 누적이 최대가 되도록 하는 행동을 선택하는 과정을 반복적으로 수행하는 강화 학습(reinforcement learning)으로 나뉘게 된다.

3.2. 머신러닝 프레임워크

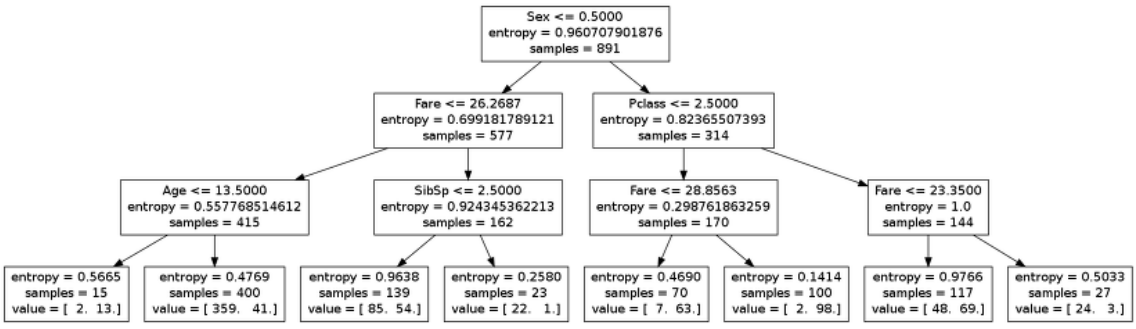
머신러닝을 Python에서 쉽게 구현하기 위해 다양한 라이브러리를 제공하는 대표적인 머신러닝 프레임워크에는 scikit-learn이 있다. scikit-learn은 2007년 구글 썸머 코드에서 처음 구현됐으며 현재 파이썬으로 구현된 가장 유명한 기계 학습 오픈 소스 라이브러리다. scikit-learn은 다른 라이브러리와의 호환성이 좋고 내적으로는 통일된 인터페이스를 가지고 있기 때문에 매우 간단하게 여러 기법을 적용할 수 있어 쉽고 빠르게 최상의 결과를 얻을 수 있다.

라이브러리의 구성은 크게 지도 학습, 비지도 학습, 모델 선택 및 평가, 데이터 변환으로 나눌 수 있다(http://scikit-learn.org/stable/user_guide.html). 지도 학습에는 Support Vector Machine(SVM), Nearest Neighbors(NN) 등의 기법을 구현한 함수들이 있으며 비지도 학습에는 군집화, 이상치 검출 등이 있다. 모델 선택 및 평가에는 교차 검증(cross-validation), 파이프라인(pipeline) 등이 있으며 마지막으로 데이터 변환에는 속성 추출(Feature Extraction), 전처리(Preprocessing) 등이 있다. 이 중 우리는 지도 학습 관련 함수들과 모델 평가 관련 라이브러리를 중점적으로 사용하였다.

4. Random Forest 알고리즘

4.1 의사 결정 트리 (Decision Tree)

의사 결정 트리[5]란 여러 개의 기준을 통해 각 독립변수를 분류하고, 그를 통해 종속변수를 예측하는 예측 기법이다. 예를 들어, 타이타닉 생존자를 찾기 위한 의사결정트리를 아래 그림과 같이 설정할 수 있다. 성별, 좌석 등급, 나이 등의 기준으로 각 승객들을 분류할 수 있다. 이렇게 분류된 각 집단들의 생존 확률 정보를 통해 어떤 승객의 정보가 주어졌을 때 생존확률을 예측할 수 있다. 의사결정트리는 회귀나무와 분류나무로 나뉘는데, 회귀나무는 숫자형 결과를 반환하고 분류나무는 범주형 결과, 즉 어떤 집단에 속할 것인지를 반환한다.



[Fig. 5] 의사 결정 트리

5) [Fig. 5] 출처 : https://swalloow.tistory.com/92

4.2 앙상블 기법(Ensemble)

의사결정트리에서는 각 기준을 설정하는 방법에 따라 모델이 달라지고, 이에 따라 예측된 결과가 달라진다. 모델에 따라 잘못된 결과가 예측되거나 모델이 학습 데이터에 과적합될 수 있다, 이 오차를 줄이기 위한 방법이 앙상블 기법이다. 앙상블(Ensemble) 프랑스어로 조화를 뜻한다. 이는 여러 학습 모델로부터 예측된 결과들을 종합하여 하나의 최종 예측 결과를 결정하는 방법이다.

4.3 랜덤 포레스트(Random Forest)

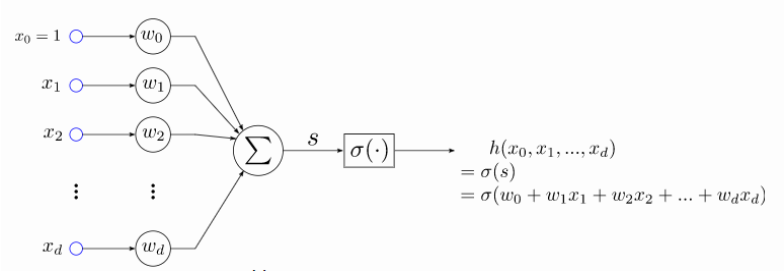
랜덤 포레스트는 대표적인 앙상블 기법 중의 하나로, 여러 모델의 투표 결과를 통해 결과를 예측하는 방법이다. 무작위로 생성된 트리 노드를 사용하여 오류나 과적합을 방지할 수 있다. 이때 전체 데이터를 여러 샘플로 분할하여 각 모델에 입력하는데, 이 과정을 통해 무작위로 분류 기준을 설정할 수 있다.

5. 딥러닝(Deep Learning)

5.1. 딥러닝

딥러닝은 머신러닝 기법 중 하나로 인공신경망(Artificial Neuron Network)를 이용한 학습 모델을 통해 머신러닝의 과정을 진행한다. 인공신경망은 여러 개의 퍼셉트론[6](perceptron)의 연결체로 생각할 수 있다, 퍼셉트론은 n개의 입력 벡터에 대하여 각 벡터에 대해 가중치(weight)를 곱한 값들의 합에 활성함수(activation function)을 적용하여 최종 출력 값으로 출력하는 선형 모델이다. 즉, n-dimensional 데이터 $x = (x_0, x_1, ..., x_n)$이 입력되면 각 성분에 가중치 $(w_0, w_1, ..., w_n)$을 곱하여 모두 합한 값을 $s = \sum_{k=1}^{n} w_k x_k$라고 하면, 퍼셉트론 $h(x) = \sigma(s)$로 나타낼 수 있다. 퍼셉트론의 반복을 통하여 다층의 인공신경망을 구현할 수 있으며, 이때 입력 벡터가 자리 잡는 층을 입력층, 출력 값이 자리 잡는 층을 출력층, 그 사이의 모든 층을 은닉층이라고 한다.

현재 CNN(Convolution Neural Network, 합성곱 신경망), RNN(Recurrent Neural Network, 순환 신경망) 등 다양한 인공 신경망의 구현 알고리즘이 개발되어 이미지 인식, 자연어 처리 등 다양한 분야에서 활용되고 있다.



[Fig. 6] 퍼셉트론의 모식도

5.2. 딥러닝 프레임워크

Python에서 딥러닝을 쉽게 구현하기 위해 다양한 프레임워크가 존재한다. 딥러닝 프레임워크에는 Tensorflow, Keras, Pytorch 등이 있다. 현재 가장 많이 사용되는 딥러닝 라이브러리는 Tensorflow이다. Keras는 Tensorflow를 기반으로 한 고급 API로 Tensorflow에 비해 사용이 용이하나 그 기능이 제한적이다. 본 연구에서는 관련 커뮤니티가 가장 잘 구축되어 있으며 기존 연구에 많이 활용되는 Tensorflow 기반 Keras를 사용할 예정이다.

6) [Fig. 6] 출처 : http://research.sualab.com/introduction/2017/10/10/what-is-deep-learning-1.html

Ⅲ. 연구 방법 및 과정

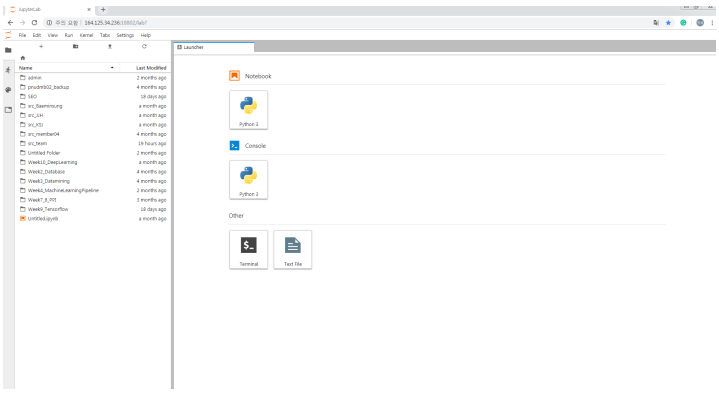
1. 연구 과정

월별 연구 과정은 다음과 같다.

Table 1. 월별 연구 과정

일련 번호	수행 내용	추진 일정									
		3 월	4 월	5 월	6 월	7 월	8 월	9 월	10 월	11 월	12 월
1	프로그래밍 언어 및 프레임워크 교육										
2	인공지능 기술 및 데이터 분석 기술 교육										
3	기계학습 및 데이터마이닝 프레임워크 교육										
4	생명의료 데이터베이스 활용 및 학습 서버 구현										
5	생명의료 데이터 패턴 분석 연구										
6	인공지능 기반 생명의료 분석 모델 설계 및 구현										
7	전체 시스템 및 모델 연동 및 테스트										
8	전체 시스템 및 연구의 성능 평가										

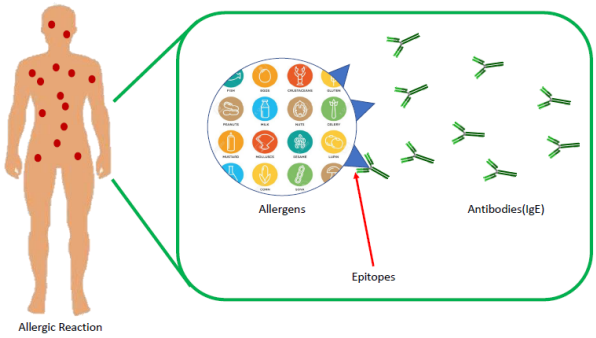
연구 및 실습에 사용한 환경은 JupyterLab이다. JupyterLab은 웹 기반 사용자 인터페이스로, 동일 서버에서 접속한 사용자 별로 개인 작업이 가능하다. 또한, 모든 파일 및 데이터는 공유할 수 있다. 기본적으로 Python 3를 지원하고, 확장 기능을 통해 Python2, R, C++ 등 여러 인터프리터를 지원하며 linux 기반 터미널을 제공한다.



[Fig. 7] JupyterLab 인터페이스

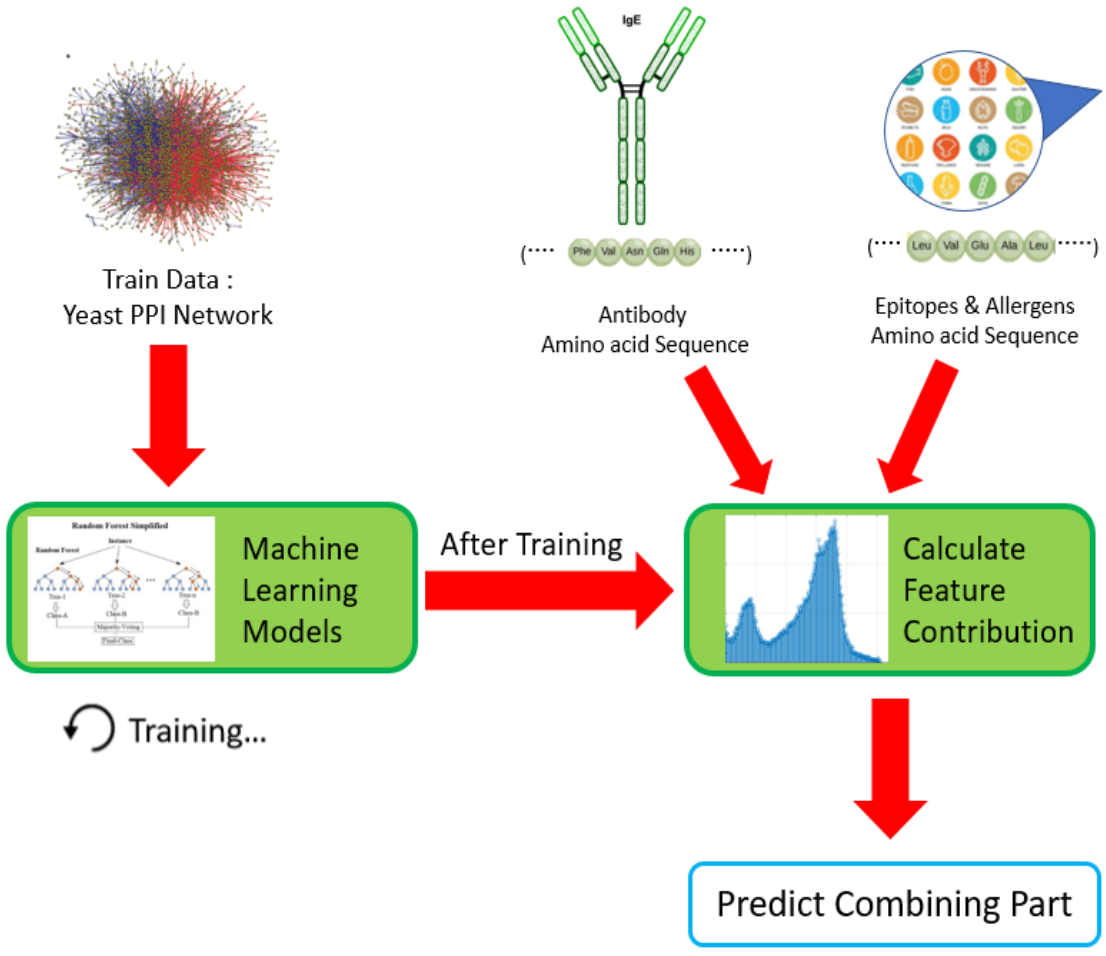
2. 전체 흐름도

[Fig. 8]과 같이 체내에서 알레르기 반응은 항원으로서 작용하는 분자인 알레르겐의 에피토프에 IgE 계열의 항체가 결합하는 형태로 일어난다.



[Fig. 8] 체내에서 일어나는 알레르기 반응의 모식도

[Fig. 9]는 이번 연구의 전체 흐름을 나타내는 그림이다. 우리는 대부분의 항원과 항체가 아미노산들의 중합체인 단백질이기 때문에 항원-항체 반응이 일종의 PPI라는 점을 활용하기로 하였다. 효모 단백질들의 서열과 상호작용 여부가 정리되어 있는 Yeast-PPI 데이터 셋을 통해 두 단백질의 아미노산 서열이 주어졌을 때, 단백질의 결합 여부를 예측하는 머신 러닝(딥러닝) 모델을 구축한다. 이 모델을 통해 데이터 마이닝・전처리를 통하여 수집한 항체와 항원의 아미노산 서열 데이터가 주어졌을 때 항원・항체의 결합 여부를 예측하고 만약 항원과 항체가 결합하는 경우, 항체가 항원에 결합하는 부위인 항원의 에피토프에 해당되는 서열을 예측한다.



[Fig. 9] 연구 흐름도

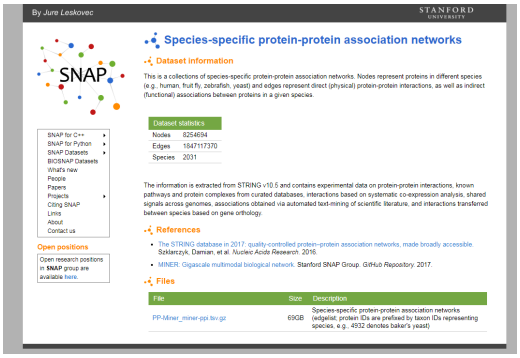
3. 데이터셋 구축

3.1. Yeast PPI 데이터셋의 활용 및 전처리

먼저 결합 여부 및 결합 부위 예측에 사용될 모델 구축의 기반이 되는 PPI 데이터셋을 구축하였다. 이 데이터셋 구축 과정에서는 현재 가장 연구가 많이 진행되어있으며 PPI 관련 연구에서 많이 사용되는 효모 단백질 관련 데이터를 활용하였다. 전처리 전 Yeast PPI Dataset의 경우 많은 효모 단백질의 서열과 RCSB PDB ID 및 서로 상호작용하는 효모들의 RCSB PDB ID 쌍에 대한 정보를 담고 있다.

결합하는 효모 단백질의 쌍의 경우 Stanford SNAP(http://snap.stanford.edu/biodata/index.html) 에서 가져왔으며, 해당 효모 단백질의 시퀀스 정보들은 SGD(https://www.yeastgenome.org/)에서 가져왔다.

Stanford SNAP(Stanford Network Analysis Project) 라이브러리는 2004년부터 스탠포드 대학교가 네트워크 분석 및 그래프 문제를 위해 연구하고 있는 프로젝트로, 생물정보학을 포함한 다양한 분야의 그래프 데이터가 존재한다. 우리는 이중 BIOSNAP 데이터셋에서 PP-Miner 데이터셋의 효모 데이터들을 사용하였다.



[Fig. 10] Stanford SNAP

SNAP의 PP-Miner 데이터셋에는 결합하는 두 단백질의 RCSB PDB ID pair들이 yeast.reduced.edgelist 파일로 존재하며, 각 효모 단백질들의 서열이 yeast.sequences.npz 파일로 존재한다. 기존의 yeast.reduced.edgelist 파일의 경우 결합하는 두 단백질의 ID pair들만 포함되어 있어 학습 및 테스트 데이터로서 활용하기 어렵다는 판단을 하였다. 따라서 기존 파일에서 단백질 ID 리스트와 결합하는 단백질들의 ID pair 리스트를 제작하여, 모든 단백질들에 대해 두 단백질의 결합 여부가 positive/negative인지를 저장한 새로운 형태의 npz 데이터베이스를 구축하였다.

[Fig. 11] Yeast PPI 데이터의 positive/negative pair 분류를 위한 코드

```
import random
from collections import defaultdict

def connection_check(p1, p2):
    if (p1,p2) in edge_list or (p2,p1) in edge_list:
        return True
    else:
        return False

def make_node_connection_db(node, edge):
    connection_db = []
    check = defaultdict(lambda :True)

    for p1 in node:
        for p2 in node:
```

```
                connection = []
                connection.append(p1)
                connection.append(p2)

                key, ikey = (p1,p2), (p2,p1)

                if p1!=p2:
                    if connection_check(p1,p2):
                        connection.append(1)
                    else:
                        connection.append(0)

                    if check[key]:
                        connection_db.append(connection)
                        check[key] = False
                        check[ikey] = False
                    else:
                        pass

    return connection_db
```

npz는 python numpy에서 지원하는 바이너리 파일 형식으로 npz를 통해 여러 개의 ndarray를 하나의 파일로 저장할 수 있다. numpy.savez와 numpy.load 메소드를 통하여 npz 파일의 저장과 불러오기를 구현할 수 있으며, 이번 연구 과정에서 대부분의 데이터 파일을 npz 형식으로 가공하였다.

그리고 효모 단백질의 RCSB PDB ID와 그 단백질의 아미노산 서열을 대응시킨 데이터베이스를 npz 형태로 제작하였다. Yeast PPI 데이터셋의 각 효모에 대해 RCSB PDB ID를 이용해 PDB 사이트에서 효모의 염기 서열을 웹 크롤링한다. RCSB PDB에 업로드된 각 효모의 염기 서열을 아미노산 서열로 번역한 후 단순화하여 저장했다.

데이터 전처리 과정에서는 먼저 20가지의 아미노산을 화학적 성질에 따라 7개의 그룹으로 분류했다. 분류표는 다음과 같다.

Group 1 : Alanine(A), Glycine(G), Valine(V)
Group 2 : Isoleucine(I), Leucine(L), Phenylalanine(F), Proline(P)
Group 3 : Tyrosine(Y), Methionine(M), Threonine(T), Serine(S)
Group 4 : Histidine(H), Asparagine(N), Glutamine(Q), Tryptophan(Y)
Group 5 : Arginine(R), Lysine(K)
Group 6 : Aspartic Acid(D), Glutamic Acid(E)
Group 7 : Cysteine(C)

[Fig. 12] 아미노산 분류 목록

[Fig. 12]의 분류를 이용해 효모의 염기 서열을 아미노산 서열로 번역한 후 각 아미노산들을 분류에 맞게 치환할 수 있다. 예를 들어 아미노산 A(알라닌), G(글리신), V(발린)은 모두 A 그룹에 속하므로 세 아미노산을 모두 A로 치환해 나타낸다.

다음으로 CTF(Conjoint Triad Feature) 기법을 이용해 단백질의 아미노산 서열을 가공한다. CTF는 연속된 아미노산의 패턴 분포를 이용해 단백질을 표현하는 방법이다. 예를 들어, 아미노산의 종류가 A,B 2가지인 경우를 생각해 보면 단백질의 1차 구조가 AABAABB일 때 아미노산을 3개씩 끊어 읽어서 나타나는 패턴은 AAB,ABA,BAA,AAB,ABB이다. 아미노산 2종류로 만들 수 있는 패턴은 AAA,AAB,ABA,ABB,BAA,BAB,BBA,BBB로 총 8개이다. 따라서 AABABB를 CTF로 가공하면 [0,2,1,1,1,0,0,0]이고 각 원소를 패턴의 총 개수인 5로 나누어 정규화하면 [0,0.4,0.2,0.2,0.2,0,0,0]이 된다. 이와 같이 7가지의 아미노산으로 표현되어 있는 단백질을 고정 길이 3으로 가공하면 단백질이 크기가 343인 1차원 리스트로 표현이 가능하다. 이를 이용해 데이터 전처리를 시행하면 단백질의 길이가 서로 다르더라도 피처의 길이는 일정해지므로 모델을 설정하기에

용이해진다는 장점이 있다.

[Fig. 13] 아미노산의 카테고리화를 통한 고정 길이 3의 CTF 코드

```
def ctf(amino_seq):
    count = [0]*343
    category = [['A','G','V'],['I','L','F','P'],['Y','M','T','S'],['H','N','Q','W'],
['R','K'], ['D','E'],['C']]
    for j in range(len(amino_seq)-2):
            seq = amino_seq[j:j+3]
            reduced_seq = ''
            for k in range(7):
                for l in seq:
                    if l in category[k]:
                        reduced_seq += str(k)
            num = int(reduced_seq[0])*49+int(reduced_seq[1])*7+int(reduced_seq[2])
            count[num]+=1
    for i in range(343):
        count[i]=count[i]/(len(amino_seq)-2)
    return count
```

4. 테스트용 데이터셋 구축

4.1. 알레르겐 염기 서열 데이터셋 제작

체내에서 알레르겐으로 작용할 수 있는 다양한 단백질들의 서열 데이터가 추후 모델 검증 및 예측에서 필요하였기 때문에, 알레르겐의 아미노산 혹은 mRNA 서열 데이터를 가져올 수 있는 데이터셋을 제작하기로 하였다.

이때 AllerBase의 알레르겐 목록과 NCBI의 알레르겐 mRNA 서열 데이터를 사용했다. AllerBase는 대표적인 알레르겐 및 알레르기 반응 관련 데이터베이스로 알레르겐 및 항체, 에피토프, 교차 반응에 대한 다양한 데이터를 제공한다. Allerbase의 알레르겐 목록에 따르면 알레르겐은 동물, 식물, 곰팡이, 박테리아, 바이러스의 다섯 종류로 구분한다. Allerbase의 알레르겐 목록을 5개의 분류별로 모두 웹 크롤링한다. 이후 각 알레르겐 별로 NCBI에 업로드된 mRNA 서열을 웹 크롤링한다. 이 데이터들을 npz 형식으로 가공하여 AllerBase의 알레르겐 ID 입력 시 염기 서열을 반환하도록 데이터셋을 제작하였다.

4.2. 알레르겐별 에피토프 정보 데이터셋 제작

또한 추후 예측의 정확도를 측정하기 위해 이미 에피토프 서열이 밝혀져 있는 알레르겐들의 에피토프 서열에 대한 정보가 필요하였다. AllerBase에 업로드된 에피토프 정보를 이용하면 에피토프 부위가 밝혀진 알레르겐 단백질의 에피토프 부위에 해당하는 아미노산 서열을 알 수 있다.

따라서 AllerBase의 IgE Epitopes 항목에서 알레르겐별로 알레르겐 ID, 이성질체 정보, 에피토프 아미노산 서열, 알레르겐 펩타이드 사슬 상에서 에피토프의 위치, 에피토프 검출에 사용된 IgE 항체의 정보를 웹 크롤링하여 이차원 리스트로 저장하였다. 이후 이를 npz 파일로 가공하여 알레르젠 ID 입력 시 저장한 정보를 반환하도록 하였다. 예를 들어, 알레르겐 ID Bos d 6을 입력하면 [['Bos d 6.0101'], ['LILNR'], ['478-482'], ['IgE from serum of patients allergic to beef']]가 출력되게 된다. 이때 Bos d 6.0101이 Bos d 6의 이성질체이며, 에피토프 부위의 아미노산 서열은 LINLR이고 이는 전체 알레르겐 아미노산 서열에서 478-482번째 아미노산이며, 해당 알레르겐 검출에 사용된 항체는 소고기 알레르기 환자에게서 추출한 IgE라는 것이다.

Ⅳ. 연구 결과

1. Yeast PPI 머신러닝 모델 제작 및 성능 평가

머신러닝 프레임워크를 활용하여 앙상블 학습 기반 머신러닝 모델을 제작해보았다. 앙상블 학습은 머신러닝에서 여러 개의 모델을 학습시켜 그 모델들의 예측결과들을 이용해 하나의 모델보다 더 나은 값을 예측하는 방법을 말한다. 대표적인 학습 모델인 Random Forest, Gradient Boosting, Voting, Bagging을 이용해 간단한 학습과 테스트를 진행한다.

파이썬의 scikit-learn 라이브러리에서 제공하는 각각의 Classifier 모델과 정확도 측정, 데이터셋을 나누는 함수 등을 가져와 데이터셋을 가공 후 학습하는 과정을 객체 형식으로 정리하여 간편하게 데이터셋을 테스트 할 수 있게 만들었다.

[Fig. 14] OOP로 구성한 머신러닝 모델 학습을 위한 코드 (Random Forest의 경우만 수록, 라이브러리 import 생략)

```
class myClassifier:
    def __init__(self, dataX, dataY):
        self.dataX=dataX
        self.dataY=dataY
        self.testX=0
        self.testY=0
    def splitData(self, randstate=0):
        self.dataX, self.testX, self.dataY, self.testY = train_test_split(self.dataX, self.dataY, random_state=50)
        print('myClassifier : DataX - ',self.dataX.shape)
        print('myClassifier : DataY - ',self.dataY.shape)
        print('myClassifier : testX - ',self.testX.shape)
        print('myClassifier : testY - ',self.testY.shape)

    def setTestData(self, dataX, dataY):
        self.testX=dataX
        self.testY=dataY
        print('myClassifier : Test Data Set')

    def cutData(self, start, length, rand=0):
        xlst=[]
        ylst=[]
        if not rand:
            self.dataX=self.dataX[start:start+length]
            self.dataY=self.dataY[start:start+length]
            print('myClassifier : Data Cut')
        else:
            for i in random.sample(range(len(self.dataX)), length):
                xlst.append(self.dataX[i])
                ylst.append(self.dataY[i])
            self.dataX=xlst
            self.dataY=ylst
            print('myClassifier : Data Select Random')

class myRF(myClassifier):
    def __init__(self, dataX, dataY):
        super().__init__(dataX, dataY)
        print('myRandomForestClassifier : Data Set Finished')

    def startTraining(self, estimators=100, randomstate=18):
        self.rf_clf=RandomForestClassifier(n_estimators=estimators, random_state=randomstate, n_jobs=-1,
max_depth=30)
```

```
            self.rf_clf.fit(self.dataX, self.dataY)
            print('myRandomForestClassifier : Finished Training')

        def checkAccuracy(self):
            print("myRandomForestClassifier : Training Data Set Accuracy - {:.3f}".format(self.rf_clf.score(self.dataX, self.dataY)))
            print("myRandomForestClassifier : Test Data Set Accuracy - {:.3f}".format(self.rf_clf.score(self.testX, self.testY)))
```

이때 1차 가공한 Yeast PPI 데이터셋은 데이터의 수가 지나치게 많다고 판단하여 numpy의 random 메소드를 활용해 두 단백질이 결합하는 데이터 30000개, 두 단백질이 결합하지 않는 데이터 30000개를 무작위 추출하여 데이터셋을 2차 가공 후 npz 형식으로 저장해 추후 학습 및 연구에 사용하였다.

[Fig. 15] Yeast PPI 데이터셋의 2차 가공을 위한 코드

```
def classify_true_false(db):
    false = []
    true = []

    for data in db:
        if data[2] == "0":
            false.append(data)
        else:
            true.append(data)

    false=np.array(false)
    true=np.array(true)

    return false, true

connection_db_false, connection_db_true = classify_true_false(db_load)

np.random.shuffle(connection_db_false)
np.random.shuffle(connection_db_true)

def random_db(size):
    db_false=[]
    db_true=[]

    db_false = connection_db_false[0:size]
    db_true = connection_db_true[0:size]

    return db_false, db_true

connection_db_reduced_false, connection_db_reduced_true = random_db(30000)
np.savez("Connection_DB_Reduced_False.npz", data=connection_db_reduced_false)
np.savez("Connection_DB_Reduced_True.npz", data=connection_db_reduced_true)
```

각 머신러닝 모델들을 이용해 샘플 데이터 60000개를 train:test 비율을 8:2로 split하여 학습을 진행한 후

scikit-learn의 classification_report 메소드를 이용해 precision-recall 성능 평가를 진행하였다. precision-recall 성능 평가는 학습 모델의 성능을 평가하는 가장 대표적인 방법으로, 성능 지표인 정밀도(precision)와 재현율(recall)은 컨퓨젼 행렬(confusion matrix)에서 얻은 값들을 통하여 다음 공식으로 구할 수 있다.

Table 2. Confusion Matrix

		<실제 값>	
		True	False
<예측 값>	True	TP(True Positive)	FP(False Positive - 제1종 오류)
	False	FN(False Negative – 제2종 오류)	TN(True Negative)

$$precision = \frac{TP}{TP+FP},\ recall = \frac{TP}{TP+FN}$$

이때 모델의 학습 및 테스트에 넣은 데이터들은 CTF를 거친 결합 혹은 결합하지 않는 두 단백질의 아미노산 서열을 일렬로 나열한 크기 686(7^3*2)의 데이터들이다.

또한 scikit-learn의 precision_recall_curve 메소드와 matplotlib를 이용하여 4가지 모델의 P-R curve를 그려보았고, AP value를 그래프와 함께 출력하였다. 4가지의 모델들에 대해 성능 평가를 진행하였을 때 Random Forest 모델이 accuracy 값이 가장 높았으며 PR curve의 면적(AP value)이 가장 컸다. 따라서 추후 연구에서 Random Forest 모델을 활용하기로 하였다.

[Fig. 16] 성능 평가표 및 P-R curve 작성을 위한 코드([Fig. 14]의 myRF 메소드 아래에 추가하는 코드)

```
def classification_report(self):
    y_pred = self.rf_clf.predict(self.testX)
    target_names = ['class 0', 'class 1']
    print(classification_report(self.testY, y_pred, target_names=target_names))

def PRcurve(self):
    y_pred = self.rf_clf.predict(self.testX)
    average_precision = average_precision_score(self.testY, y_pred)
    precision, recall, _ = precision_recall_curve(self.testY, y_pred)
    base = self.testY[self.testY==1].shape[0] / self.testY.shape[0]
    step_kwargs = ({'step': 'post'}
                   if 'step' in signature(plt.fill_between).parameters
                   else {})
    plt.step(recall, precision, color='b', alpha=0.2, where='post')
    plt.fill_between(recall, precision, alpha=0.2, color='b', **step_kwargs)
    plt.plot([0.0,1.0],[base, base], alpha=0.5, color='r', linestyle=':')

    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title('2-class Precision-Recall curve: AP={0:0.2f}'.format(average_precision))
    plt.show()
```

Table 3. Random Forest 모델의 학습 성능 평가표 (accuracy : 0.697)

	precision	recall	f1-score	support
class 0	0.70	0.69	0.70	6000
class 1	0.70	0.70	0.70	6000
avg / total	0.70	0.70	0.70	12000

Table 4. Gradient Boosting 모델의 학습 성능 평가표 (accuracy : 0.644)

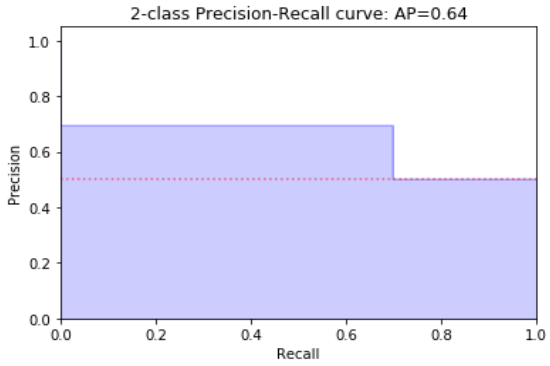
	precision	recall	f1-score	support
class 0	0.66	0.61	0.63	6000
class 1	0.63	0.68	0.66	6000
avg / total	0.65	0.64	0.64	12000

Table 5. Bagging 모델의 학습 성능 평가표 (accuracy : 0.612)

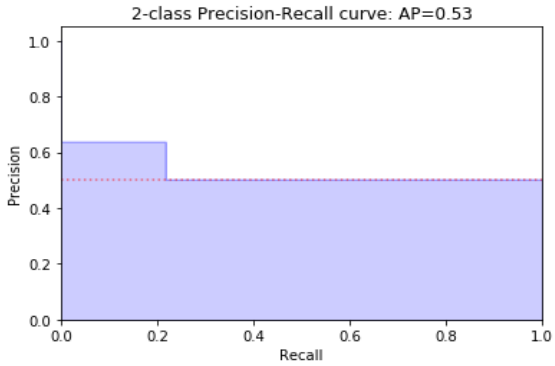
	precision	recall	f1-score	support
class 0	0.60	0.68	0.64	6000
class 1	0.63	0.54	0.58	6000
avg / total	0.61	0.61	0.61	12000

Table 6. Voting 모델의 학습 성능 평가표 (accuracy : 0.547)

	precision	recall	f1-score	support
class 0	0.53	0.88	0.66	6000
class 1	0.64	0.22	0.32	6000
avg / total	0.58	0.55	0.49	12000



[Fig. 17] Random Forest의 P-R curve (AP=0.64)



[Fig. 18] Voting의 P-R curve (AP=0.53)

2. Yeast PPI 딥러닝 모델 제작 및 성능 평가

대표적인 딥러닝 라이브러리인 keras를 활용하여 1D-CNN 딥러닝 모델을 제작해보았다.

학습 시 step을 100개로 나누어 각 step이 진행될 때마다 loss, accuracy 값을 출력할 수 있도록 코드를 구성하여 딥러닝 모델의 정확도를 측정하였다. 학습 결과 val_loss 값이 0.3782, val_acc 값이 0.6189로 머신러닝 모델들을 통한 학습으로 얻은 정확도 값에 비하여 비슷하거나 낮은 값이 나온 것을 볼 수 있었다. 이와 더불어 의사 결정 트리를 활용한 머신러닝 모델에서 각 feature들의 기여도를 도출해주는 treeinterpreter 라이브러리를 통하여 결합 부위를 예측할 수 있다고 생각하였기에 딥러닝 모델보다는 Random Forest 모델을 활용하는 것이 효율적이라고 판단하였다.

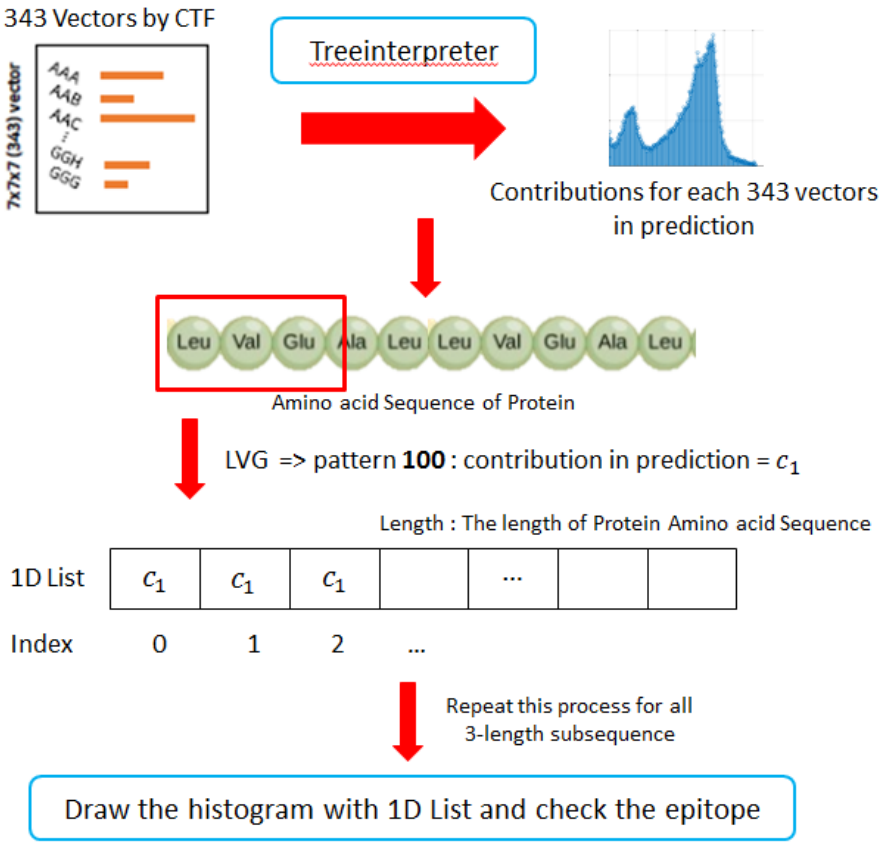
[Fig. 19] Yeast PPI 딥러닝 모델 제작 코드(일부 생략)

```
import keras
def base_model():
    input_x = keras.layers.Input(shape=(686,))
```

```
        x = keras.layers.Dense(units=384)(input_x)
        x = keras.layers.BatchNormalization()(x)
        x = keras.layers.Activation("relu")(x)
        x = keras.layers.Dense(units=128)(x)
        x = keras.layers.BatchNormalization()(x)
        x = keras.layers.Activation("relu")(x)
        x = keras.layers.Dense(units=1)(x)
        x = keras.layers.BatchNormalization()(x)
        x = keras.layers.Activation("sigmoid")(x)
        y = x

        model = keras.models.Model(input_x, y)
        model.summary()
        return model

model = base_model()

from keras.optimizers import Adam
model.compile(loss="binary_crossentropy", optimizer=Adam(3e-5), metrics=["acc"])

epochs = 100
batch_size = 512
train_y = np.expand_dims(train_y, axis=-1)
test_y = np.expand_dims(test_y, axis=-1)
model.fit(x=train_x,y=train_y,epochs=epochs, batch_size=batch_size, validation_data=(test_x, test_y))
```

3. treeinterpreter 라이브러리를 활용한 결합 부위 예측

treeinterpreter는 의사 결정 트리와 Random Forest 알고리즘 기반 모델을 해석하는데 주로 사용되는 라이브러리로써 Random Forest 알고리즘 상에서 각 feature들이 예측에 어떻게 기여하는지를 나타내주는 기능을 갖고 있다. treeinterpreter 라이브러리를 활용하여 우리는 [Fig. 20]과 같이 결합 부위를 예측하기로 하였다.



[Fig. 20] treeinterpreter를 활용한 결합 부위 예측 모식도

이번 연구 과정에서 20개의 아미노산을 7개의 카테고리로 분류하여 고정 길이 3의 vector를 통한 CTF를 진행하였기 때문에 7^3개인 343개 종류의 벡터가 존재한다. 우리는 treeinterpreter 라이브러리를 통하여 두 개의 단백질 아미노산 서열이 주어졌을 때 결합 부위 예측에 관여하는 각 벡터들의 기여도를 계산할 수 있다. 이를 이용해 항원의 아미노산 서열을 3개씩 끊어서 보면서 3개짜리 아미노산 서열 패턴에는 해당하는 벡터의 기여도를 계속해서 더해나간다. 항원 아미노산 서열의 길이만큼의 1차원 리스트를 제작한 후 각 인덱스에 각 아미노산이 결합에 얼마나 기여하는지를 계산하여 더한다. 예를 들어, 위의 그림과 같이 Leu, Val, Glu이 1,2,3번째 아미노산일 경우, 이 길이 3의 아미노산 서열을 우리가 분류한 카테고리에 따라 100이라는 패턴으로 볼 수 있고, treeinterpreter를 이용하여 계산한 그 패턴에 해당하는 벡터의 예측 기여도를 c_1이라고 하면, 1차원 리스트의 0,1,2번째 index에 c_1 값을 저장한다. 이 과정을 전체 아미노산 서열의 모든 3-length subsequence에 대해 진행하여 얻어진 1차원 리스트를 matplotlib 라이브러리를 통해 시각화하여 실제 결합 부위와의 유사도를 확인한다.

4. 테스트 데이터셋을 활용한 실제 항원-항체 반응의 결합 부위 예측

Ⅳ-1에서 제작하였던 머신러닝 모델 중 Random Forest 모델 class에 showtreeinterpreter라는 메소드를 추가하였고, 이를 통해 두 개의 단백질 서열과 결합 부위에 해당되는 서열 번호가 주어지면 위 과정을 수행하는 코드를 제작하였다.

[Fig. 21] chagepattern(아미노산 서열을 해당하는 그룹의 알파벳으로 바꾸는 함수) 및 showtreeinterpreter 메소드 코드

```
    def changepattern(seq):

catg={'A':'A','G':'A','V':'A','I':'B','L':'B','F':'B','P':'B','Y':'C','M':'C','T':'C','S':'C','H':'D','N':'D','Q':'D','W':'D','R':'E','K':'E','D':'F','E':'F','C':'G'}
        newseq=''
        for i in seq:
            newseq+=catg[i]
        return newseq

    def showtreeinterpreter(self, testX1, testX2, inx, type,prn=True ):

            ctf_list=['AAA', 'AAB', 'AAC', 'AAD', 'AAE', 'AAF', 'AAG', 'ABA', 'ABB', 'ABC', 'ABD', 'ABE', 'ABF',
'ABG', 'ACA', 'ACB', 'ACC', 'ACD', 'ACE', 'ACF', 'ACG', 'ADA', 'ADB', 'ADC', 'ADD', 'ADE', 'ADF', 'ADG', 'AEA',
'AEB', 'AEC', 'AED', 'AEE', 'AEF', 'AEG', 'AFA', 'AFB', 'AFC', 'AFD', 'AFE', 'AFF', 'AFG', 'AGA', 'AGB', 'AGC',
'AGD', 'AGE', 'AGF', 'AGG', 'BAA', 'BAB', 'BAC', 'BAD', 'BAE', 'BAF', 'BAG', 'BBA', 'BBB', 'BBC', 'BBD', 'BBE',
'BBF', 'BBG', 'BCA', 'BCB', 'BCC', 'BCD', 'BCE', 'BCF', 'BCG', 'BDA', 'BDB', 'BDC', 'BDD', 'BDE', 'BDF', 'BDG',
'BEA', 'BEB', 'BEC', 'BED', 'BEE', 'BEF', 'BEG', 'BFA', 'BFB', 'BFC', 'BFD', 'BFE', 'BFF', 'BFG', 'BGA', 'BGB',
'BGC', 'BGD', 'BGE', 'BGF', 'BGG', 'CAA', 'CAB', 'CAC', 'CAD', 'CAE', 'CAF', 'CAG', 'CBA', 'CBB', 'CBC', 'CBD',
'CBE', 'CBF', 'CBG', 'CCA', 'CCB', 'CCC', 'CCD', 'CCE', 'CCF', 'CCG', 'CDA', 'CDB', 'CDC', 'CDD', 'CDE', 'CDF',
'CDG', 'CEA', 'CEB', 'CEC', 'CED', 'CEE', 'CEF', 'CEG', 'CFA', 'CFB', 'CFC', 'CFD', 'CFE', 'CFF', 'CFG', 'CGA', 'CGB',
'CGC', 'CGD', 'CGE', 'CGF', 'CGG', 'DAA', 'DAB', 'DAC', 'DAD', 'DAE', 'DAF', 'DAG', 'DBA', 'DBB', 'DBC', 'DBD',
'DBE', 'DBF', 'DBG', 'DCA', 'DCB', 'DCC', 'DCD', 'DCE', 'DCF', 'DCG', 'DDA', 'DDB', 'DDC', 'DDD', 'DDE', 'DDF',
'DDG', 'DEA', 'DEB', 'DEC', 'DED', 'DEE', 'DEF', 'DEG', 'DFA', 'DFB', 'DFC', 'DFD', 'DFE', 'DFF', 'DFG', 'DGA',
'DGB', 'DGC', 'DGD', 'DGE', 'DGF', 'DGG', 'EAA', 'EAB', 'EAC', 'EAD', 'EAE', 'EAF', 'EAG', 'EBA', 'EBB', 'EBC',
'EBD', 'EBE', 'EBF', 'EBG', 'ECA', 'ECB', 'ECC', 'ECD', 'ECE', 'ECF', 'ECG', 'EDA', 'EDB', 'EDC', 'EDD', 'EDE',
'EDF', 'EDG', 'EEA', 'EEB', 'EEC', 'EED', 'EEE', 'EEF', 'EEG', 'EFA', 'EFB', 'EFC', 'EFD', 'EFE', 'EFF', 'EFG', 'EGA',
'EGB', 'EGC', 'EGD', 'EGE', 'EGF', 'EGG', 'FAA', 'FAB', 'FAC', 'FAD', 'FAE', 'FAF', 'FAG', 'FBA', 'FBB', 'FBC',
'FBD', 'FBE', 'FBF', 'FBG', 'FCA', 'FCB', 'FCC', 'FCD', 'FCE', 'FCF', 'FCG', 'FDA', 'FDB', 'FDC', 'FDD', 'FDE',
'FDF', 'FDG', 'FEA', 'FEB', 'FEC', 'FED', 'FEE', 'FEF', 'FEG', 'FFA', 'FFB', 'FFC', 'FFD', 'FFE', 'FFF', 'FFG', 'FGA',
'FGB', 'FGC', 'FGD', 'FGE', 'FGF', 'FGG', 'GAA', 'GAB', 'GAC', 'GAD', 'GAE', 'GAF', 'GAG', 'GBA', 'GBB', 'GBC',
'GBD', 'GBE', 'GBF', 'GBG', 'GCA', 'GCB', 'GCC', 'GCD', 'GCE', 'GCF', 'GCG', 'GDA', 'GDB', 'GDC', 'GDD', 'GDE',
'GDF', 'GDG', 'GEA', 'GEB', 'GEC', 'GED', 'GEE', 'GEF', 'GEG', 'GFA', 'GFB', 'GFC', 'GFD', 'GFE', 'GFF', 'GFG',
'GGA', 'GGB', 'GGC', 'GGD', 'GGE', 'GGF', 'GGG']

            a=changepattern(testX1)
            b=changepattern(testX2)
            newseq_x1=list(a)
            newseq_x1_weight=[0 for i in range(len(a))]
            newseq_x2=list(b)
            newseq_x2_weight=[0 for i in range(len(b))]
```

```
            prediction, bias, contributions = ti.predict(self.rf_clf, np.array([ctf(testX1)+ctf(testX2)]))
            if prn==True:
                print ("Prediction", prediction)
                print ("Bias (trainset mean)", bias)
                print ("Feature contributions:")
                print(contributions.shape)
                for c in range(len(contributions)):
                    print (c, -abs(contributions[c]))

                print ("-"*20)

            for i in range(0,len(a)-2):
                seqq=a[i:i+3]
                idx=ctf_list.index(seqq)
                newseq_x1_weight[i]+=abs(contributions[0][idx][0])
                newseq_x1_weight[i+1]+=abs(contributions[0][idx][0])
                newseq_x1_weight[i+2]+=abs(contributions[0][idx][0])

            for i in range(0,len(b)-2):
                seqq=b[i:i+3]
                idx=ctf_list.index(seqq)
                newseq_x2_weight[i]+=abs(contributions[0][idx+343][0])
                newseq_x2_weight[i+1]+=abs(contributions[0][idx+343][0])
                newseq_x2_weight[i+2]+=abs(contributions[0][idx+343][0])

            if type==0:
                plt.hist(range(len(a)), bins=len(a), weights=newseq_x1_weight, density=False, cumulative=False,
label='A', color='r', edgecolor='black', linewidth=1.2)

                plt.axvspan(inx[0], inx[1], facecolor='gray', alpha=0.5)
                plt.title('TreeInterpreter Weight', pad=10)
                plt.xlabel('Sequence', labelpad=10)
                plt.ylabel('Weight', labelpad=10)

                plt.minorticks_on()
                plt.tick_params(axis='both', which='both', direction='in', pad=8, top=True, right=True)
                plt.show()
            if type==1:
                plt.plot(newseq_x1_weight)
                plt.axvspan(inx[0], inx[1], facecolor='gray', alpha=0.5)
                plt.show()
```

이후 다음의 두 항원-항체 아미노산 서열 데이터를 이용해 test를 시도해보았다.

[데이터 1]

항원 단백질의 아미노산 서열 :

MGVFNYETETTSVIPAARLFKAFILDGDNLFPKVAPQAISSVENIEGNGGPGTIKKISFPEGFPFKYVKDRVDEVDHTNF
KYNYSVIEGGPIGDTLEKISNEIKIVATPDGGSILKISNKYHTKGDHEVKAEQVKASKEMGETLLRAVESYLLAHSDAYN

항체 단백질의 아미노산 서열 :

LESGGGLVQPGGSLKLSCVASGFDFNRYYMSWVRHTPGQGLEWIGEINPDSSTMNYTPSLKDKFIISRDNAKNTLYLQMSKVRSEDTALYYCVQRGLAYWGQGTLVTVSAAKTTPPSVYPLAPGSAAQTNSMVTLGCLVKGYFPEPVTVTWNSGSLSSGVHTFPAVLQSDLYTLSSSVTVPSSTWPSETVTCNVAHPASSTKVDKKIVPRDCTS

결합 부위에 해당하는 아미노산 서열 :

131~135번째 아미노산

[데이터 2]

항원 단백질의 아미노산 서열 :

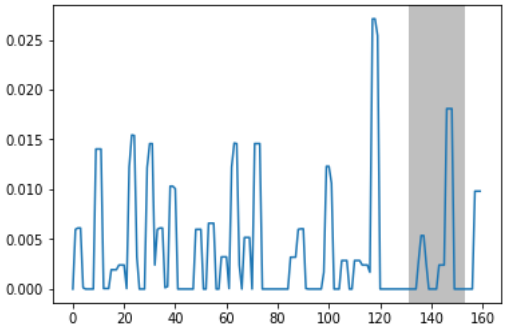
MASSSSVLLVVVLFAVFLGSAYGIPKVPPGPNITATYGDKWLDAKSTWYGKPTGAGPKDNGGACGYKDVDKPPFSGMTGCGNTPIFKSGRGCGSCFEIKCTKPEACSGEPVVVHITDDNEEPIAPYHFDLSGHAFGAMAKKGDEQKLRSAGELELQFRRVKCKYPEGTKVTFHVEKGSNPNYLALLVKYVNGDGDVVAVDIKEKGKDKWIELKESWGAIWRIDTPDKLTGPFTVRYTTEGGTKTEAEDVIPEGWKADTSYESK

항체 단백질의 아미노산 서열 :

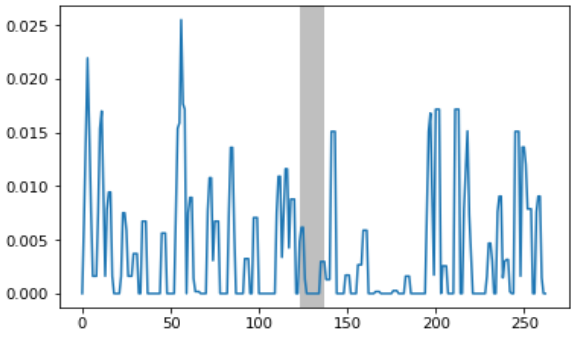
ELTQSPSSLSASVGDRVTITCRASQSIGNYLNWYQQKPGKAPNLLIYAASSLQSGVPSRFSGSGSGTDFTLTISSLQREDFATYYCQQSNRTPITFGQGTRLEIKG

결합 부위에 해당하는 아미노산 서열 :

123~137번째 아미노산



[Fig. 20] 데이터 1의 treeinterpreter 분석 히스토그램



[Fig. 16] 데이터 2의 treeinterpreter 분석 히스토그램

두 히스토그램(가로축 : 단백질 아미노산 서열 순서, 세로축 : 해당 아미노산에서의 feature 값의 합) 에서 회색으로 색칠되어 있는 부분은 실제 알려져 있는 결합 부위를 의미한다. 우리는 이 히스토그램에서 결합 예측 기여도가 높은 peak 부분과 회색으로 색칠되어 있는 부분이 어느 정도 일치할 것이라고 예측하였다. 그러나 데이터 1의 경우는 실제 결합 부위에서 조금 벗어났으며, 데이터 2의 경우는 peak 최하점이 실제 결합 부위인 것을 관찰할 수 있었다. 또한, 두 데이터의 경우 모두 결합 여부에 대한 prediction 값이 [0.6, 0.4], [0.64, 0.36]으로 실제 결합하는 항원-항체 단백질임에도 불구하고 False 값이 True 값보다 크게 나오게 되었다.

Ⅴ. 결론

1. 오차의 원인 분석

연구 결과 항원-항체 반응에서 결합 부위의 예측이 다소 정확하지 못했다. 이는 다양한 오차를 일으키는 원인이 존재했기 때문이라고 생각한다. 그 오차들의 원인은 다음과 같다.

먼저, CTF를 진행하는 과정에서 20가지의 아미노산을 7개의 카테고리로 분류하여 차원 축소를 진행하였다. 아미노산들의 상호작용으로 인해 이루어지는 단백질 간의 상호작용에서 아미노산들을 카테고리화한 것은 오차 발생의 원인이 될 수 있었다. 또한 단백질은 1차 구조뿐만 아니라 스스로 접히거나 꼬여 2-4차 구조를

형성하는데, 우리는 아미노산들이 일렬로 배열된 폴리펩타이드, 즉 1차 구조만을 고려하였기 때문에 단백질 간의 상호작용을 고려하기에는 어려웠을 가능성이 있다. 그리고 학습을 진행하는 과정에서 두 개의 단백질 아미노산 서열을 일렬로 배열하여 길이 686의 데이터들로 모델을 학습시켰다. 두 단백질의 상호작용에 대한 연구인데 두 개의 서열을 일렬로 이어 붙여서 학습을 진행시켰던 것은 다소 문제가 있었던 것으로 보인다. 추가적으로 인간의 체내에서 일어나는 항원-항체 반응을 효모 단백질 간의 상호작용 관련 데이터셋으로 연구를 진행했던 것도 오차 발생의 원인이 되었다고 생각한다.

2. 추후 연구 계획

추후의 위에서 언급한 여러 가지 오차 요인의 수정과 추가적 연구를 통해 예측의 정확도를 향상시키고자 한다.

- CTF를 20개의 아미노산에 대해서 진행하며, 벡터의 고정 길이를 3이 아닌 다른 길이로 변화시켜 보면서 진행해본다.
- 인간의 체내에서 일어나는 항원-항체 반응이기 때문에 Yeast-PPI 데이터가 아닌 인간의 단백질을 이용한 데이터로 수정하여 학습을 진행한다.
- 단백질의 단순 아미노산 서열뿐만 아니라 아미노산 서열에서 생성되는 입체 구조 등을 고려하여 예측을 진행한다.
- 모델 학습 과정에서 두 가지의 데이터를 일렬로 이어 붙이는 것이 아닌 다른 방법으로 학습을 진행할 수 있도록 한다.

3. 기대 효과

이번 연구를 통해 머신러닝・딥러닝 등의 컴퓨터과학 기술을 활용해 알레르기 반응에 대한 다양한 문제 해결에 대한 방향성을 제시하였다. 또한 미지 단백질이 체내에서 항원으로 작용할 수 있을지에 대한 예측 혹은 항원의 에피토프 부위 예측은 알레르기 반응 치료나 예방 등의 의학적 분야에서 원활하게 활용될 것으로 예상된다.

참고문헌

[1] PEDREGOSA, Fabian, et al. Scikit-learn: Machine learning in Python. Journal of machine learning research, 2011, 12.Oct, Pages 2825-2830.

[2] Yvan Saeys, Iñaki Inza, Pedro Larrañaga. A review of feature selection techniques in bioinformatics. Bioinformatics, Volume 23, Issue 19, 1 October 2007, Pages 2507-2517.

[3] Seonwoo Min, Byunghan Lee, Sungroh Yoon. Deep learning in bioinformatics. Briefings in Bioinformatics, Volume 18, Issue 5, September 2017, Pages 851-869.