Multivariate Statistics and Lab

Final Project

<미국 대학원 합격 확률 예측 데이터 분석>

오영민, 육현정, 최상진

Ⅰ. 서론

본 프로젝트가 대학생 신분으로 진행하는 것이기 때문에 관련 데이터를 분석해볼 수 있으면 좋겠다고 생각되어 해당 주제를 선정하게 되었다. 우리는 대학원 합격 여부가 개인의 학업 및 진로 선택에 있어 큰 영향을 미칠 수 있다고 생각하고, 그렇기에 많은 학생들이 관심 가지는 주제라고 판단하였다. 현재 대부분의 학생들이 불완전한 정보 또는 주관적 의견에 의존하여 대학원 지원 전략을 세우고 있다는 문제점이 존재한다는 생각이 들었다. 따라서 프로젝트의 목표를 "다양한 지원자의 데이터를 분석하여 미국 대학원 합격 확률 예측에 대한 모델을 만드는 것"으로 잡고 이에 대한 분석을 진행하고자 한다. 구체적으로 GRE Score, TOEFL Score, University Rating, SOP, LOR, CGPA, Research, Chance of Admit 변수를 종합적으로 고려한 예측 모델을 만듦으로써 "이 중 어떤 요소들이 합격에 가장 큰 영향을 미치는가?", "실제 데이터를 바탕으로 한 학생의 합격 확률 예측을 어떻게 할 수 있는가?" 등을 알아보고자 한다.

Ⅱ. 자료소개

해당 데이터는 <Mohan Sacharya, Asfia Armanan, Aneeta S Antony : 2019년 IEEE 국제 데이터 과학 분야 컴퓨터 지능 컨퍼런스의 대학원 입학 예측을 위한 회귀 모형 비교> 에서 사용된 데이터로, 우리 팀은 https://www.kaggle.com/ 사이트에서 해당 데이터를 얻었다.

데이터는 공과대학 학생들을 대상으로 하여 수집되었고, 400명의 데이터를 9가지 요소로 나타내었는데, 데이터 중 1열은 Serial No. (일련번호)로, 분석 목표와 관련 없는 열이기 때문에 데이터 분석 과정에서 사용하지 않을 것이다. 미국 대학 입학은 전인적 평가가 원칙이기 때문에 다양한 요소를 고려하는 특징이 있는데, 그러한 요소들은 다음과 같다.

1. GRE(Graduate Record Examination)

GRE는 미국 학부 과정의 SAT에 대응되는 미국의 대학원 수학 자격시험이다. GRE는 크게 General Test(작문, 언어, 수리)와 Subject Tests(각종 전공 분야, 현재 4개 과목)로 나뉜다. 보통 GRE라고 하면 전자를 지칭한다. 과목으로는 verbal reasoning(언어논증), quantitative reasoning(수리논증), analytical writing(분석적 작문)의 3가지가 있으며, 버벌과 퀀트는 만점 170점, 최하점 130점으로 150점을 중앙에 두는 표준 분포를 이루며, 작문은 0~6점 스케일에 한 구간은 0.5점이다. 본 데이터에서는 340점 만점 기준으로 해당 학생의 점수를 나타낸다.

2. TOEFL

토플은 외국어로서의 영어 시험(Test Of English as a Foreign Language)의 약자로서, 미국 ETS의 주관 하에 시행되는 영어 능력 시험이다. TOEFL은 영어를 모국어로 하지 않는 사람들을 대상으로 응시자가 영어권의 대학 등 교육, 학술기관에서 수학할 수 있는지를 판단하는 중요한 척도로 활용된다. 주로 토플은 각 대학원 또는 학과에서 요구하는 기본적인 입학 요건을 충족하는데

목표가 있어, 이를 충족한 이후에는 비교적 덜 중요한 요소이다. 시험 구성은 Reading, Listening, Writing, Speaking 4가지 영역으로 구성되어 있으며 총 120점 만점이다. 본 데이터에서는 120점 만점 기준 해당 학생의 점수를 나타낸다.

3. University Rating

졸업한 학부 대학 순위는 대학원 진학 시, 명문 대학 출신의 경우 해당 대학의 평판과 교육의 질이 입학 사정관들에게 긍정적인 인상을 줄 수 있으나 다른 요소들에 비해 상대적으로 덜 중요할 수 있다. 본 데이터에서는 해당 학생들이 생각하는 본인의 출신 학부 대학을 5점 만점 기준으로 측정하였다.

4. SOP(Statement of Purpose)

SOP는 직역하면 '목적에 대한 서술'로, 특정 분야에서 어떠한 학업/연구를 왜 하려고 하는지에 대한 목적을 기술하는 것이다.

실제로 학교에서 지원자를 평가할 때 가장 중요하게 고려하는 항목이 이 SOP라고 하는데. 지원자의 특정 분야 및 주제에 대한 생각과 지식의 깊이, 학업에 대한 태도를 판단할 수 있는 근거가 되기 때문이다. 이를 바탕으로 지원자가 해당 학교 프로그램에서 성공적으로 학업을 마칠 수 있는지, 그리고 장기적으로는 동 분야에서 지속적으로 기여할 수 있는 인재가 될 수 있는지를 판단하게 된다.

지원자와 학교 프로그램 간의 목적이 일치하는가와도 관련이 있는데. 학교는 중점 연구 분야를 계속해서 강화할 수 있고, 학생은 학업 목적을 달성할 수 있어야 하기 때문이다. 그래서 지원하려는 학교 프로그램 및 교수진의 주요 연구 분야와 지원자 본인이 심도 있게 공부하고자 하는 연구 주제 및 목적이 일치해야 한다. 본 데이터에서는 SOP 영향력을 5점 만점 기준으로 측정하였다.

5. LOR(Letters of Recommendation)

미국은 우리나라 대학 입시처럼 전공 필기시험을 보지 않기 때문에 미국 대학원 입학처에서 지원한 학생의 능력에 대해 객관적으로 3자의 의견을 확인할 수 있는 방법이다. 대학원 지원 과정에서는 추천서가 3장 필요한데, 특히 석사 과정을 마치고 박사과정에 지원할 경우 석사 지도교수님의 추천서가 필수라고 볼 수 있다. 공통적으로 학생과의 관계, 지원자가 해당 전공을 수학하기에 알맞은 사람이라 생각하는 이유, 학생의 역량을 입증할 수 있는 예시와 같은 사항이 포함되어야 한다. 본 데이터에서는 LOR 영향력을 5점 만점 기준으로 측정하였다.

6. CGPA

CGPA(Cumulative Grade Point Average)는, 누적 평균 이수 학점을 뜻한다. 미국 입시를 포함한 글로벌 입시에서 가장 기본이면서도 중요한 부분으로, 특히 미국 대학 진학에서 그 비중이 매우 높다. CGPA는 지원자가 학부과정에서 얼마나 성공적으로 학업을 수행했는지를 보여주기에 높은 CGPA는

지원자가 강한 학문적 능력을 가지고 있으며, 대학원 과정에서도 성공할 가능성이 높다는 것을 입증하는 지표이기도 하여 높은 CGPA는 다른 지원자들 간에 경쟁에서 우위를 점할 수 있는 중요한 요소 중 하나이다. 본 데이터에서는 10점 만점으로 지원자 본인의 학부 CGPA를 나타낸다.

7. **Research**

연구 경험은 대학원 진학 시 매우 중요한 요소 중 하나이다 . 이를 통해 얻은 문제 해결 능력, 비판적 사고, 독립적 연구 수행 능력 등은 대학원에서 요구되는 핵심능력이기에 이러한 능력을 갖추고 있다는 것을 보여줄 수 있다. 본 데이터에서는 연구 경험이 있는 학생의 경우1, 없는 경우 0으로 표시하였다.

8. **Chance of Admit :**학생들이 예측한 대학원 합격 확률을 나타내는 지표이다.

III. 분석 방법 및 결과

전체적인 분석과정은 아래 도식 Figure 1과 같다. 먼저, 데이터를 학습 데이터와 테스트 데이터로 분리하여, 학습데이터를 통해 사EDA를 진행한다. 다시 학습 데이터를 학습데이터와 검증 데이터로 나누어, 어떠한 모델을 사용할 것인지 후보군을 정하고, 가장 좋은 모델을 선정한다. 최종적으로 테스트 데이터를 통해 모델의 성능을 확인하고, 실제 값과의 어떠한 차이가 있는지를 알아 볼 것이다.

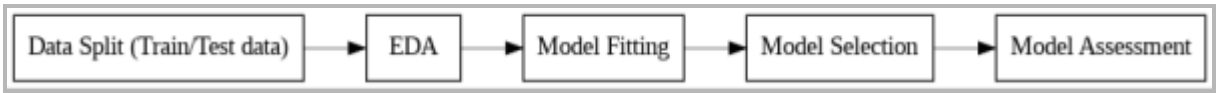

Figure 1.

데이터의 평가 지표는 차이를 한눈에 알아보기 쉬운 MAE를 이용하여 평가를 진행할 것이다. 우리의 분석 목표는 대학원 합격 확률을 예측하는 데에서 멈추지 않고, 지원자의 기존 스펙에서 타 지원자보다 부족한 부분을 찾아 솔루션을 지원하는 것이 최종 목표이기에, 모델의 정확도와 해석력 두 가지를 평가지표로 고려할 것이다.

먼저 가장 좋은 모델을 찾되, 해당 모델이 복잡하여 해석을 하는데에 어려움이 있다면, 더 단순하면서 에러가 비교적 작은 모델을 가장 좋은 모델로 선정할 것이다. 모델 선택 방법은 Cross validation 방법을 이용하여 각 모델 별 최적의 하이퍼 파라미터를 찾아, 가장 좋은 모델을 선정할 것이다.

Figure 2는 종속변수인 Chance of Admit을 y축으로, 나머지 변수들을 각각 x축으로 하는 산점도를 그린 그래프이다. 각각의 모든 독립변수가 종속변수와 선형관계를 나타내고 있음을 알 수 있다. 따라서 선형 모델을 가정하여 모델링을 진행하는 것이 합리적으로 보인다.

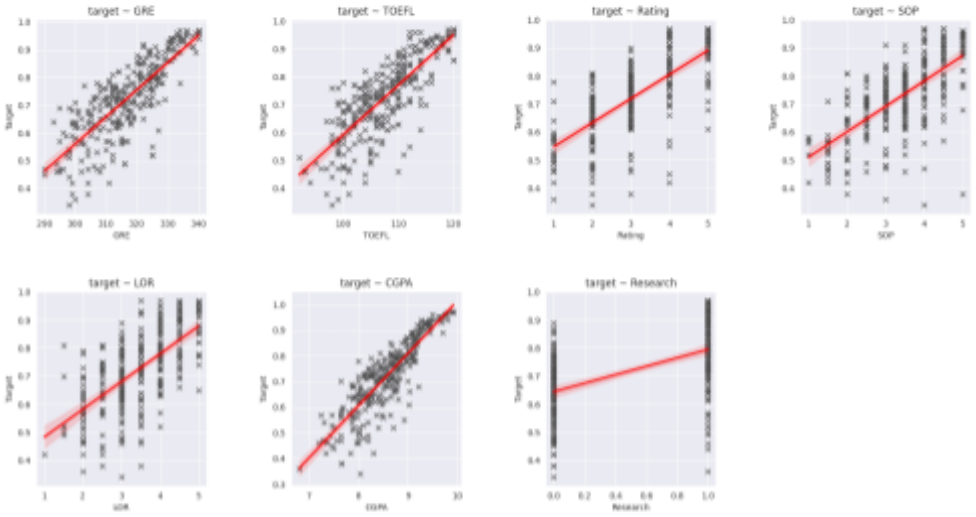

Figure 2.

Figure 3는 모든 변수간의 상관관계를 나타낸 히트맵 그림이다. TARGET인 Chance of Admit의 상관관계는 CGPA, GRE, TOEFL 순으로 높다.또한 CGPA는 GRE, TOEFL과 상관관계가 높고, GRE와 TOEFL도 상관관계가 높다. 따라서 종속변수와 독립변수와의 상관관계가 높아 선형 모델이 좋은 성능을 보일 것이라 기대가 되지만, 서로 상관관계가 높은 변수들이 존재하기에 다중 공선성을 우려해보아야 할 것이다. GRE, TOEFL, CGPA 세 지표의 상관관계가 서로 높은데, 이들은 모두 명확한 수치로 제시되는 정량적이며 객관적인 지표로 해석해 볼 수 있고, 반대로 SOP, LOR, Rating은 자기소개서, 추천서, 학부 순위로 정성적이며 다소 주관적인 지표로 해석해 볼 수 있을 것이다. 따라서 독립변수들 내에서 정량적 지표와 정성적 지표 2개의 잠재변수가 있을 것이라 가정하는 것은 합리적으로 보이기에, 요인을 2개로 설정하여 varimax rotaion을 적용하여 인자분석을 실시해 주었다.

Figure 3.

다음은 독립변수들에 대하여 인자분석을 진행한 결과를 나타낸 표이다.Factor 1의 로딩의 절대값이 GRE, TOEFL, CGPA, Research 순으로 높았고, Factor 2에 대해서는 SOP,LOR, Rating 순으로

높게 나왔다. 즉, 예상한 바와 같이 Factor 1이 정량적인 지표를, Factor 2가 정성적인 지표를 나타낸다고 해석해 볼 수 있다.[1] PCA결과 고유값도 2.20, 0.87, 0.73으로 1~2개의 요인 수가 적절해보인다.

Loadings:	Factor 1 (정량적 지표)	Factor 2 (정성적 지표)
GRE	-0.883	0.324
TOEFL	-0.790	0.414
Rating	-0.531	0.653
SOP	-0.531	0.800
LOR	-0.337	0.736
CGPA	-0.746	0.540
Research	-0.550	0.244

Figure 4.

즉, 다중 회귀 모형을 사용하여 예측을 진행하는 것이 정확도와 모델의 해석적인 측면에서도 좋을 것으로 보이기에 선형모델들을 후보군으로 정하여, 모델을 선정할 것이다. EDA 단계에서 알아보았듯이, 독립변수간의 상관관계가 높기에, 이에 대한 제약조건이 있는 모델을 선정하는 것이 타당할 것이다. 또한, 모델의 복잡성을 포함할 수 있도록 적절한 비선형성을 추가해주기 위하여 linear basis expansion을 적용한 모델을 사용할 것이다. 이들을 고려한 모델 후보들은 다음과 같다. 전체적인 모델 파이프라인은 Figure 5와 같다.

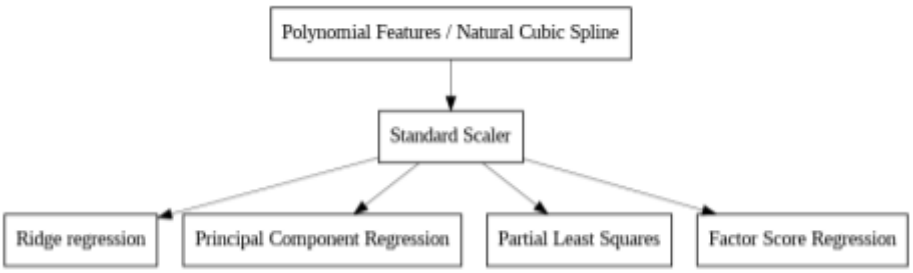

Figure 5.

1. Linear Basis Expansion[2]

먼저, 모델의 복잡성, 즉 분산을 증가시켜 편향을 줄이고, 이후에 regularization을 통해 적절한 모델을 찾기 위하여, 다음과 같이 true model을 가정할 것이다.

$$f(X) = \sum_{m=1}^{M} \beta_m h_m(X)$$

위의 $h_m(X)$에 대해서 polynomial 변환과, natural cubic spline 두가지 방법을 이용하여 모델의 복잡성을 더해줄 것이다. polynomial 변환의 차수는 3-way interaction 을 넘어간 교호작용은 해석이 어려워 보이기에 최대 차수는 3으로 제한하였다. natural cubic spline의 knot의 수는 종속변수와 독립변수간의 관계가 대부분 선형에 가까우므로 최대 3개까지로 제한하여 grid search를 통해 최적의 파라미터를 찾아보았다.

2. Ridge Regression[3]

Ridge regression는 shrinkage method중 하나로, 불편추정량에 규제항을 더하여, 편향을 어느정도 희생시키면서, 분산을 줄여주기에 prediction error를 줄이는 방법이다. 릿지 회귀의 해는 closed form으로 $\hat{\beta} = (X^T X + \lambda I)^{-1} X^T y$임이 알려져 있다. 이를 $X = UDV^T$의 특이값 분해를 이용하여 나타내면 다음과 같다.

$$\hat{f}(X) = X(X^T X + \lambda I)^{-1} X^T y = UD(D^2 + \lambda I)^{-1} DU^T y = \sum_{i=1}^{p} u_j \frac{d_j^2}{d_j^2+\lambda} u_j^T y$$

여기서 u_j는 U의 j번째 열벡터이고,d_j는 D의 j번째 대각 성분이다. 즉 d_j가 작아지면, 규제의 강도가 강해짐을 확인할 수 있다. 또한, 표본 공분산 행렬이 $S = X^T X/N = VD^2 V/N$ (N은 데이터셋의 수)이기에, 첫번째 주성분 방향에 대하여 z_1의 표본분산을 다음과 같이 계산할 수 있다.

$$Var(Z_1) = Var(Xv_1) = \frac{1}{N}\sum_{i=1}^{N}(x_i v_1)^2 = \frac{d_1^2}{N}||v_1||^2 = \frac{d_1^2}{N}$$

즉, 작은 d_j는 작은 표본 분산을 가지는 방향이므로, 릿지 회귀는 변동성이 작은 방향들의 영향을 줄어들기에, 독립변수간의 공선성이 있을 경우, 좋은 성능을 보인다.

먼저, linear basis expansion을 통해 복잡성을 더하여, 최적의 λ는 각 모델별로 Grid search를 통하여 선정하였다.

3. Principal Component Regression[4]

Principal component regression은 주성분 분석을 통해 유도된 선형결합인 Z_m 을 독립변수로 회귀분석을 진행하는 것이다. 즉, 모든 주성분을 사용할 경우 단순 선형회귀 모형과 같다. 주성분의 개수는 모델의 해석력을 위하여 최대 3개까지로 제한하여 최적의 파라미터를 찾아볼 것이다.

4. Partial Least Squares[5]

Principal component regression의 경우, 단순히 독립변수들의 분산을 최대로 보존하는 축을 찾아, 해당 축을 이용하여 회귀분석을 진행하였다. 다시 말해서, m번째 주성분 v_m 은 다음을 만족하는 해이다.

$$max_{\alpha} Var(X\alpha)\ subject\ to\ ||\alpha||\ =\ 1,\ \alpha^T S v_l\ =\ 0,\ l\ =\ 1,\ldots,m\ -\ 1$$

반면에 Partial Least Squares의 경우, 종속변수와의 상관관계까지 고려하여 다음을 만족하는 m번째 component $\hat{\varphi}_m$를 찾는 해이다.

$$max_{\alpha} Corr^2(X\alpha,\ y)Var(X\alpha)\ \ subject\ to\ ||\alpha||\ =\ 1,\ \alpha^T S\hat{\varphi}_l\ =\ 0,\ l\ =\ 1,\ldots,m\ -\ 1$$

PLS의 방향 축은 최대 3개까지로 제한하여 최적의 파라미터를 찾아보았다.

5. Factor Score Regression[6]

PCA를 이용하여 구한 로딩을 통한 factor score를 이용하여 회귀분석을 진행하였다. 최대 component의 수는 3개로 제한하여 최적의 파라미터를 찾아보았다.

3-Fold Cross Validation을 이용하여 진행한 결과는 다음과 같다. 학습데이터를 다시 학습데이터와 검증데이터로 나누어, 학습데이터를 이용해 cross validation을 진행하여 최적의 파라미터를 찾고, 이를 통해 검증데이터로 MAE를 계산하였다.[7]

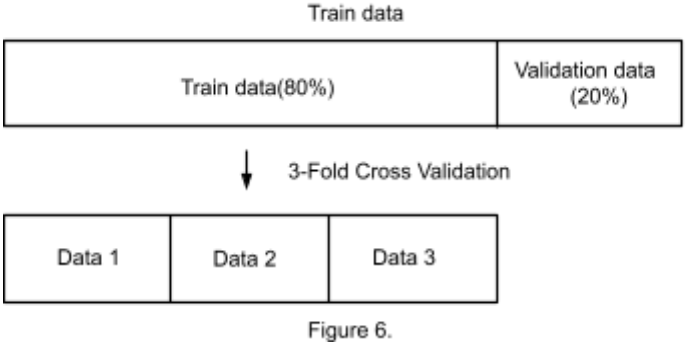

Figure 6.

	MAE	best_parameters
Polynomial + Ridge	0.05513	λ = 7 degree = 1
Polynomial + PCR	0.05529	n_components = 2 degree = 1
Polynomial + PLS	0.05442	n_components = 3 degree = 2
Polynomial + FSR	0.05324	n_components = 1 degree = 1
Spline + Ridge	0.05500	λ = 9 n_knots = 1
Spline + PCR	0.05529	n_components = 2 n_knots = 1
Spline + PLS	0.05456	n_components = 3 n_knots = 1
Spline + FSR	0.05361	n_components = 3 n_knots = 1

분석 결과, Factor score를 이용한 회귀 모델이 두 linear basis expansion 방법 모두에서 가장 좋은 성능을 보였다. 다음은 가장 좋은 성능을 보인 Polynomial + FSR의 최적의 파라미터인 요인수가 1인 인자분석 결과를 나타낸 표이다.[8]

Loadings:	Factor 1
GRE	-0.881
TOEFL	-0.879
Rating	-0.815
SOP	-0.778
LOR	-0.710
CGPA	-0.932
Research	-0.578

요인 분석 결과, 위의 표와 같이 특정한 내재된 변수를 해석하기가 어려워 보이기에 위 EDA결과에서 제시된 요인이 2개인 FSR을 최종적인 모델로 선정하였다. 해당 모델에 대해 검증 데이터로 MAE를

계산한 결과, 0.05357이 나왔고, 이는 최적의 모델과 0.0003차이로 성능 차이가 미미하고, 해석력이 더 강하다는 이점이 있다. 또한 타 모델보다도 성능이 여전히 좋으므로, 해석이 가능한 변수가 2개인 해당 모델이 가장 적합하다고 결론을 내렸다. 해당 모델의 첫번째 독립변수는 위의 EDA의 결과와 같이 Factor 1인 정량적인 지표이고, 두번째는 Factor 2는 정성적인 지표로 각각의 회귀계수 추정량은 -0.0944, 0.0619이다. 즉, 정량적인 지표가 한 단위를 올릴때마다 합격 확률을 더 높게 예측한다는 것을 알 수 있다. Figure 7은 최종적인 모델의 파이프라인이다.[9]

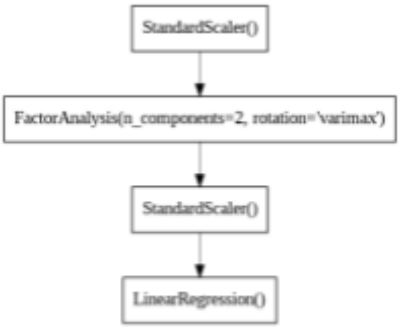

Figure 7.

IV. 결론

Figure 8은 최종적으로 테스트 데이터에 대해서 예측을 수행한 결과이다.[10] 왼쪽 그림은 x축을 종속변수로 y축을 실제 값과 예측값의 차의 절대값을 나타낸 그림이고, 오른쪽 그림은 x축을 실제 종속 변수 값, y축을 예측값으로 산점도를 그린 그림이다. 지원자의 실제 합격확률이 낮을수록 합격 확률을 과대 추정하는 경향이 있다는 것을 알 수 있다. 테스트 데이터와의 MAE는 0.05237이다. 해당 모델은 지원자의 스펙을 정성적, 정량적 평가의 두 가지 요인으로, 각각의 회귀계수를 이용하여 현재 보완하고 싶은 부분을 파악하고, 어느정도의 합격확률을 올려줄 수 있을지를 수치적으로 제시해준다. 특히, GRE, TOEFL, SOP, LOR등은 입시 준비 기간에 대비가 가능하므로, 지원자 개개인의 성향에 따라 각각의 준비를 배분해줄 수 있다는 점에서도 의의가 있다. 해당 모델을 활용하여 미국 대학원 입시 정보가 부족한 학부생들에게 많은 도움이 될 것이라 기대가 된다.

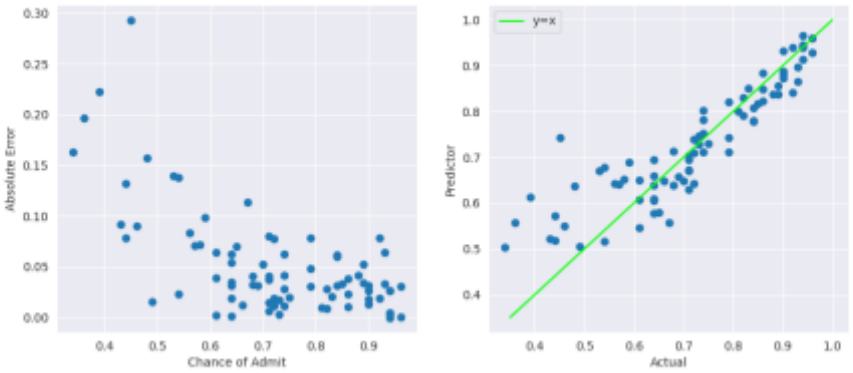

Figure 8.

Ⅴ. Appendix

1.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import matplotlib.gridspec as gridspec

from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import StandardScaler, FunctionTransformer

from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.decomposition import PCA,FactorAnalysis
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.metrics import mean_absolute_error
from sklearn.cross_decomposition import PLSRegression
from sklearn.base import BaseEstimator, TransformerMixin



%matplotlib inline
sns.set_style('darkgrid')
warnings.filterwarnings('ignore')

df = pd.read_csv("/content/drive/MyDrive/CS/24-1/다변량/Dataset/train.csv")
df.columns = ['id','GRE','TOEFL','Rating','SOP','LOR','CGPA','Research','Target']
X = df[['GRE', 'TOEFL', 'Rating', 'SOP', 'LOR', 'CGPA', 'Research']]
y = df['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)


X_scaled = scaler.fit_transform(X)
fa = FactorAnalysis(rotation='varimax', n_components=2)
fa.fit(X_scaled)

fa_scores = fa.transform(X)
fa_scores_scaled = scaler.fit_transform(fa_scores)
loadings =  fa.components_

df_fa = pd.DataFrame({'Feature' : ['GRE', 'TOEFL', 'Rating', 'SOP', 'LOR',
'CGPA','Research'], 'Factor1': loadings[0,:], 'Factor2':loadings[1,:]})
df_fa
```

2.

```
class NaturalCubicSplineFeatures(BaseEstimator, TransformerMixin):
    def __init__(self, knots=None, df=None):
        self.knots = knots
        self.df = df # (df+1)-knots calculated at uniform quantiles

    def fit(self,X,y=None):
        if self.df is not None:
            quantiles = np.linspace(0,1,self.df + 1)
            self.knots = []
            for i in range(X.shape[1]):
                self.knots.append(np.unique(np.quantile(X[:,i], quantiles)))


        self.dfs_ = np.array([len(k)-1 for k in self.knots])
        dfs_cumsum = [0] + list(np.cumsum(self.dfs_))
        self.positions_ =  [(dfs_cumsum[i-1], dfs_cumsum[i]-1) for i in range(1,
len(dfs_cumsum))]
        return self

    def transform(self,X):
        features_basis_splines = []
        for i in range(X.shape[1]):
            features_basis_splines.append(
                self.__expand_natural_cubic(X[:, i:i+1], self.knots[i]))

        return np.hstack(features_basis_splines)


    @staticmethod
    def __dk(X,k,k_last):
        return (X-k).clip(0) **3 / (k_last - k)

    @staticmethod
    def __expand_natural_cubic(X,ks):
        basis_splines = [X]
        dki_last = NaturalCubicSplineFeatures.__dk(X,ks[-2],ks[-1])
        for knot in ks[:-2]:
            dki = NaturalCubicSplineFeatures.__dk(X,knot,ks[-1])
            basis_splines.append(dki - dki_last)

        return np.hstack(basis_splines)
add_intercept = FunctionTransformer(
    lambda X: sm.add_constant(X), validate=True)

3.
poly_ridge_pipeline = Pipeline([
    ('poly', PolynomialFeatures()),
    ('scaler', StandardScaler()),
    ('ridge', Ridge())
])
spline_ridge_pipeline = Pipeline([
                        ('scale_features', StandardScaler()),
                        ("expand_basis",NaturalCubicSplineFeatures()),
```

```
                        ('scale_basis',StandardScaler()),
                        ('add_intercept_column', add_intercept),
                        ('ridge', Ridge())
                       ])
param_poly_ridge = {
    'poly__degree': [1, 2, 3],
    'ridge__alpha': [3,5,7]
}
param_spline_ridge = {
    'expand_basis__df' : [1,2,3],
    'ridge__alpha' : [7,9,11,13]
}
```

4.

```
poly_pcr_pipeline = Pipeline([
    ('poly', PolynomialFeatures()),
    ('scaler1', StandardScaler()),
    ('pca', PCA()),
    ('scaler2', StandardScaler()),
    ('lr', LinearRegression())
])
spline_pcr_pipeline = Pipeline([
                        ('scale_features', StandardScaler()),
                        ("expand_basis",NaturalCubicSplineFeatures()),
                        ('scale_basis',StandardScaler()),
                        ('add_intercept_column', add_intercept),
                        ('pca', PCA()),
                        ('scaler', StandardScaler()),
                        ('lr', LinearRegression())
                       ])
param_poly_pcr = {
    'poly__degree': [1, 2, 3],
    'pca__n_components': [2,3]
}
param_spline_pcr = {
    'expand_basis__df' : [1,2,3],
    'pca__n_components': [2,3]
}
```

5.

```
poly_pls_pipeline =  Pipeline([
```

```
    ('poly', PolynomialFeatures()),
    ('scaler', StandardScaler()),
    ('pls', PLSRegression()),
])
spline_pls_pipeline = Pipeline([
                        ('scale_features', StandardScaler()),
                        ("expand_basis",NaturalCubicSplineFeatures()),
                        ('scale_basis',StandardScaler()),
                        ('add_intercept_column', add_intercept),
                        ('pls', PLSRegression())
                       ])

param_poly_pls = {
    'poly__degree': [1, 2, 3],
    'pls__n_components': [2,3]
}

param_spline_pls = {
    'expand_basis__df' : [1,2,3],
    'pls__n_components': [2,3]
}

6.
poly_fa_pipeline = Pipeline([
    ('poly', PolynomialFeatures()),
    ('scaler1', StandardScaler()),
    ('factor', FactorAnalysis(rotation='varimax')),
    ('scaler2', StandardScaler()),
    ('lr', LinearRegression())
])
spline_fa_pipeline = Pipeline([
                        ('scale_features', StandardScaler()),
                        ("expand_basis",NaturalCubicSplineFeatures()),
                        ('scale_basis',StandardScaler()),
                        ('add_intercept_column', add_intercept),
                        ('factor', FactorAnalysis(rotation='varimax')),
                        ('scaler', StandardScaler()),
                        ('lr', LinearRegression())
                       ])
param_poly_fa = {
    'poly__degree': [1, 2, 3],
    'factor__n_components': [1, 2, 3]
}

param_spline_fa = {
    'expand_basis__df' : [1,2,3],
    'factor__n_components': [2, 3]
}
```

7.

```
cv = KFold(n_splits=3, random_state=2, shuffle=True)

grid_poly_ridge = GridSearchCV(poly_ridge_pipeline, param_poly_ridge, cv=cv,
scoring='neg_mean_absolute_error')
grid_poly_pls = GridSearchCV(poly_pls_pipeline,param_poly_pls, cv=cv,
scoring='neg_mean_absolute_error')
grid_poly_pcr = GridSearchCV(poly_pcr_pipeline, param_poly_pcr, cv=cv,
scoring='neg_mean_absolute_error')
grid_poly_fa = GridSearchCV(poly_fa_pipeline, param_poly_fa, cv=cv,
scoring='neg_mean_absolute_error')

grid_spline_ridge = GridSearchCV(spline_ridge_pipeline, param_spline_ridge, cv=cv,
scoring='neg_mean_absolute_error')
grid_spline_pls = GridSearchCV(spline_pls_pipeline,param_spline_pls, cv=cv,
scoring='neg_mean_absolute_error')
grid_spline_pcr = GridSearchCV(spline_pcr_pipeline, param_spline_pcr, cv=cv,
scoring='neg_mean_absolute_error')
grid_spline_fa = GridSearchCV(spline_fa_pipeline, param_spline_fa, cv=cv,
scoring='neg_mean_absolute_error')
grids_poly = [grid_poly_ridge, grid_poly_pls, grid_poly_pcr , grid_poly_fa]
grids_spline = [grid_spline_ridge, grid_spline_pls, grid_spline_pcr , grid_spline_fa]
for pipe in grids_poly:
   pipe.fit(X_train,y_train)
for pipe in grids_spline:
   pipe.fit(X_train,y_train)
poly_dict = {
          0: 'Polynomial Regression',
          1: 'Polynomial Partial Least Squares',
          2: 'Polynomial Principal Components Regression',
          3: 'Polynomial Factor Scores Regression'
          }

spline_dict = {
          0: 'Spline regression',
          1: 'Spline Partial Least Squares',
          2: 'Spline Principal Components Regression',
          3: 'Spline Factor Scores Regression'
          }

for i, model in enumerate(grids_poly):
   print('{} Test Accuracy: {}'.format(poly_dict[i],
   -model.score(X_test,y_test)))
   print('{} Best Params: {}'.format(poly_dict[i],        model.best_params_))

print('================================================================
============================')
```

```
for i, model in enumerate(grids_spline):
    print('{} Test Accuracy: {}'.format(spline_dict[i],
    -model.score(X_test,y_test)))
    print('{} Best Params: {}'.format(spline_dict[i],          model.best_params_))

print('================================================================
============================')
```

8.

```
X_scaled = scaler.fit_transform(X)
fa = FactorAnalysis(rotation='varimax', n_components=1)
fa.fit(X_scaled)

fa_scores = fa.transform(X)
loadings =  fa.components_
loadings
```

9.

```
param_poly_fa = {
    'poly__degree': [1,2,3],
    'factor__n_components': [2]
}

grid_poly_fa = GridSearchCV(poly_fa_pipeline, param_poly_fa, cv=cv,
scoring='neg_mean_absolute_error')
grid_poly_fa.fit(X_train, y_train)
grid_poly_fa.best_params_
-grid_poly_fa.score(X_test, y_test)
fin_model_pipeline = Pipeline([
    ('scaler1', StandardScaler()),
    ('factor', FactorAnalysis(rotation='varimax', n_components=2)),
    ('scaler2', StandardScaler()),
    ('lr', LinearRegression())
])
fin_model_pipeline.fit(X, y)
fin_model_pipeline.named_steps['lr'].coef_
```

10.

```
test_df = pd.read_csv("/content/drive/MyDrive/CS/24-1/다변량/Dataset/test.csv")
test_df.columns = ['id','GRE','TOEFL','Rating','SOP','LOR','CGPA','Research','Target']
test_df.head()
X_test = test_df[['GRE', 'TOEFL', 'Rating', 'SOP', 'LOR', 'CGPA', 'Research']]
y_test = test_df['Target']
test_df['Target_est'] = fin_model_pipeline.predict(X_test)
test_df['Target']
test_df['absolute_est_error'] = test_df['Target'] - test_df['Target_est']
mean_absolute_error(test_df['Target'],  test_df['Target_est'])
plt.figure(figsize=(12,5))
```

```
plt.subplot(121)
plt.scatter(test_df['Target'], abs(test_df['absolute_est_error']))
plt.xlabel("Chance of Admit")
plt.ylabel("Absolute Error")

plt.subplot(122)
xx = np.linspace(0.35, 1, 100)
plt.scatter(test_df['Target'], test_df['Target_est'])
plt.plot(xx, xx,color='#00FF00', label='y=x')
plt.xlabel("Actual")
plt.ylabel("Predictor")
plt.legend()

plt.savefig('/content/drive/MyDrive/CS/24-1/다변량/result.png')

plt.show()
```