**빅데이터응용보안**

**최종 보고서**

**「 이상거래 탐지 모델 」**

8조

사이버보안학과 201920666 송유래

수학과 201921218 김나연

사이버보안학과 201920675 문서연

**목차**

1. Introduction

1. Subject
2. Motivation
3. Environment
4. Process Architecture

2. Data set

1. Fraudulent Transactions Data
2. Data Distribution
3. Missing Values

3. Preprocessing

1. Feature Selection
2. One-Hot Encoder
3. Label Encoder
4. Scaling
5. Splitting Data Set
6. Undersampling
7. Learning Curve

4. Model Training

1. KNN(K-Nearest Neighbor)
2. Logistic Regression
3. SVM(Support Vector Machine)
4. Decision Tree
5. Random Forest
6. XGBoost(eXtreme Gradient Boosting)
7. FCN(Fully Connected Neural Network)
8. CNN(Convolutional Neural Network)

5. Evaluation

6. Improvement

7. Reference

1. Introduction

이 장에서는 주제와 선정 동기, 학습을 시킨 환경과 프로세스 아키텍처에 대한 내용을 설명한다.

a. Subject

본 프로젝트는 “이상거래 탐지 모델”로 전자금융거래에서 발생하는 이상금융거래를 탐지할 수 있도록 하는 예측 모델을 찾는 것이다. 사기, 부정 대출 등의 비정상적인 거래 데이터를 기반으로 발생 가능한 부정 거래를 예측 및 탐지하는 것을 목표로 한다.

b. Motivation

최근 현대 사회에서는 전자금융거래를 써보지 않은 사람들이 없을 정도로 자주 사용되고, 이를 사용하는 사용자들 또한 계속적으로 증가하고 있다. 이로 인해 자연적으로 악의적인 공격 행위가 증가하고 있으며, 또한 기술의 발달로 인해 공격 방식도 점점 세밀화, 고도화되고 있다. 따라서 이상금융거래로 인한 피해는 점점 심각해지고 있는 추세이다. 따라서 해당 프로젝트에서는 이상거래 탐지 모델을 주제로 선정하게 되었다.

c. Environment

테스트 환경은 Google에서 제공하는 Colaboratory를 사용했다.

d. Process Architecture

본 프로젝트의 수행과정은 먼저 Kaggle에서 제공하는 “Fraudulent Transactions Data”를 이용하여 Feature Selection을 처리해 필요한 특성을 선별한 후, 필요한 데이터에 One-Hot Encoding과 Label Encoding을 진행한다. 그 다음에는 Scaling을 처리한 후, 데이터 불균형을 해소하기 위해 Undersampling을 거치며 KNN, Logistic Regression 등의 여러 모델을 학습시킨다. 그리고 Classifier로 최종적으로 모델의 성능을 검증한다.

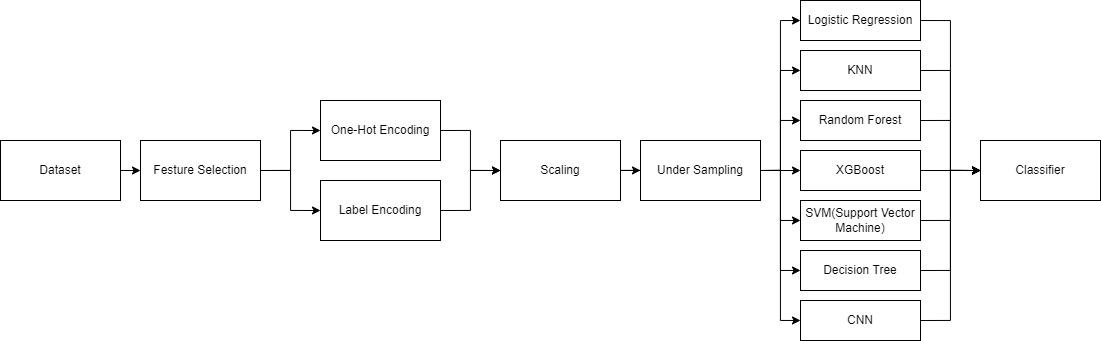


그림 1. Process Architecture

2. Data set

이 장에서는 학습을 시키기 위해 사용했던 데이터셋에 대해 설명한다

a. Fraudulent Transactions Data

본 프로젝트에서 사용한 데이터셋은 Kaggle에서 제공하고 있는 “Fraudulent Transactions Data”로 금융 회사의 사기거래 데이터들이 모여있다. 해당 데이터셋은 시간 단위를 나타내는 step, 거래 종류인 type, 거래 금액을 의미하는 amount, 거래 시작 고객명인 nameOrig, 거래 전 잔액인 oldbalanceOrg, 거래 후 잔액인 newbalanceOrig, 수신자명인 nameDest, 거래 전 최초 수취인 oldbalanceDest, 거래 후 수취인 newbalanceDest, 사기 거래 여부를 나타내는 isFraud, 대규모 거래 시도 여부를 나타내는 isFlaggedFraud로 총 11가지의 column으로 구성되어 있다.

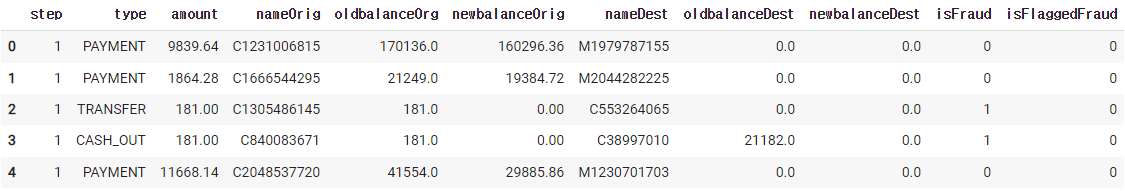


그림 2. Fraudulent Transactions Data

데이터 전체의 요약정보(데이터 수, 평균, 표준편차. 최소·최대값 등)는 다음과 같다.

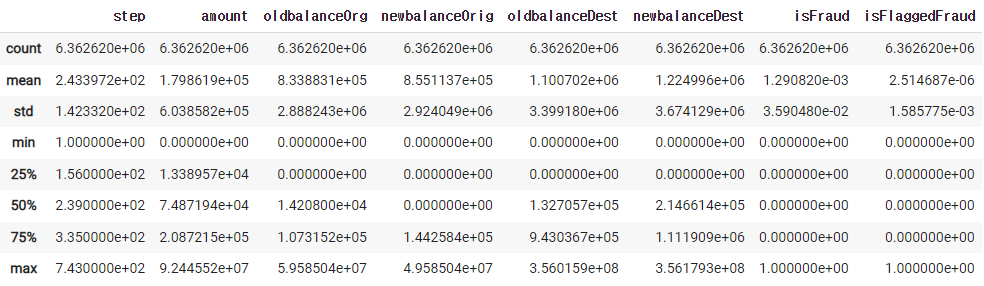


그림 3. 데이터 전체 요약정보

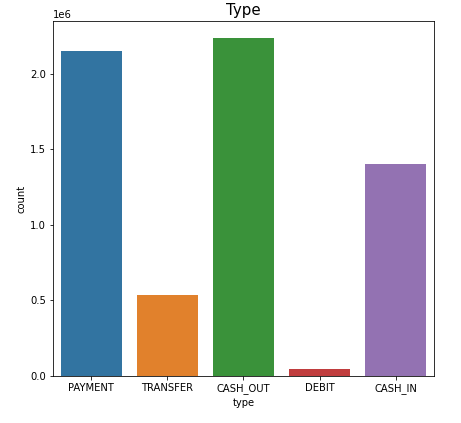


그림 4. 데이터 타입 정보

b. Data Distribution

해당 데이터는 총 6362620의 거래 데이터가 존재하며 Target column에서 0은 정상 거래, 1은 이상 거래를 나타낸다. 데이터셋의 분포를 살펴본 결과, 그 중 정상 거래는 6354407개, 이상 거래는 8213개로 99.87% : 0.13%의 비율을 차지하고 있는 것을 확인할 수 있다. 이 결과에서 확인할 수 있듯이, 정상거래에 비해 이상거래 데이터의 수가 적은 불균형한 상태이기 때문에 이를 해결하기 위해서 Undersampling 방법으로 처리하기로 했다.

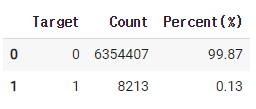


그림 5. 정상거래, 이상거래 비율

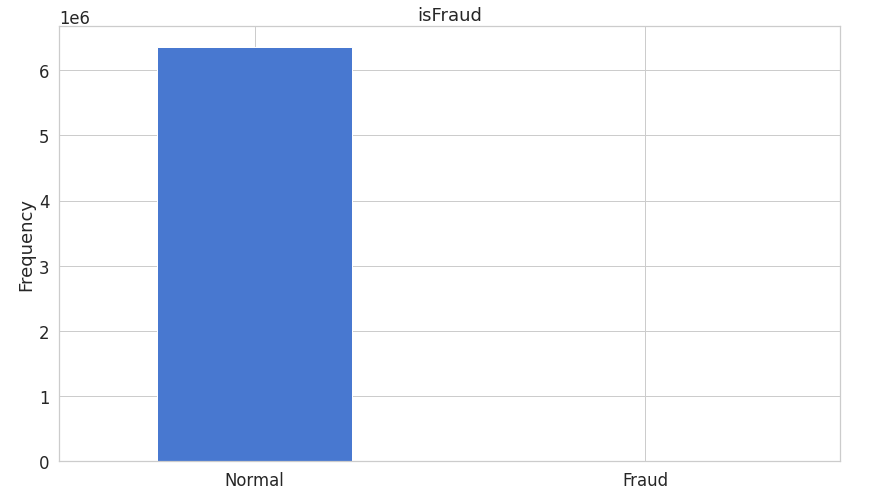


그림 6. 정상거래, 이상거래 분포

c. Missing Values

해당 데이터셋에서 결측값을 조사해본 결과, 결측값을 갖는 경우가 없었다 . 따라서 결측값에 대한 문제는 없는 것으로 확인되었다.

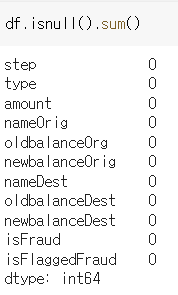


그림 7. 결측값 확인

3. Preprocessing

이 장에서는 전처리 과정에서 수행되었던 과정을 설명한다.

a. Feature Selection

데이터셋의 feature는 총 11가지가 존재하는데 그 중 시간 단위를 나타내는 step은 중요도가 낮은 feature라고 판단해 제거하였다.

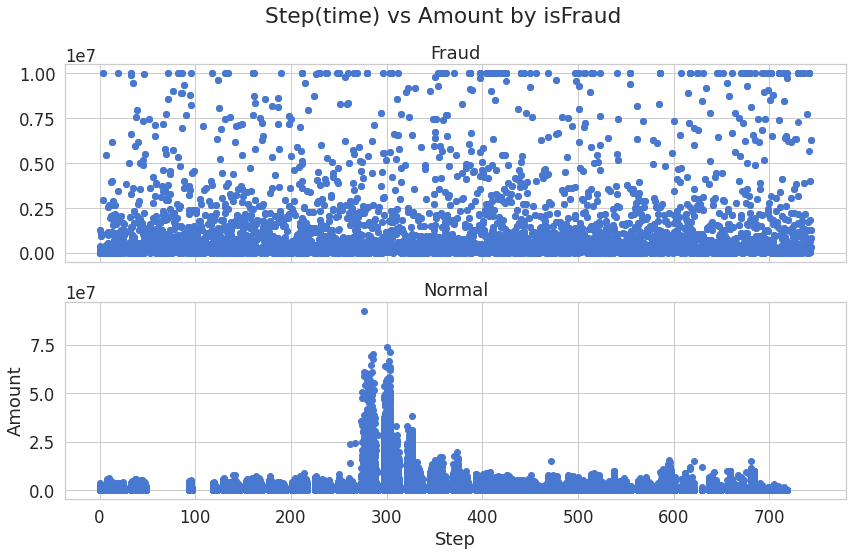


그림 8. 정상 거래와 이상거래의 step과 amount 관계

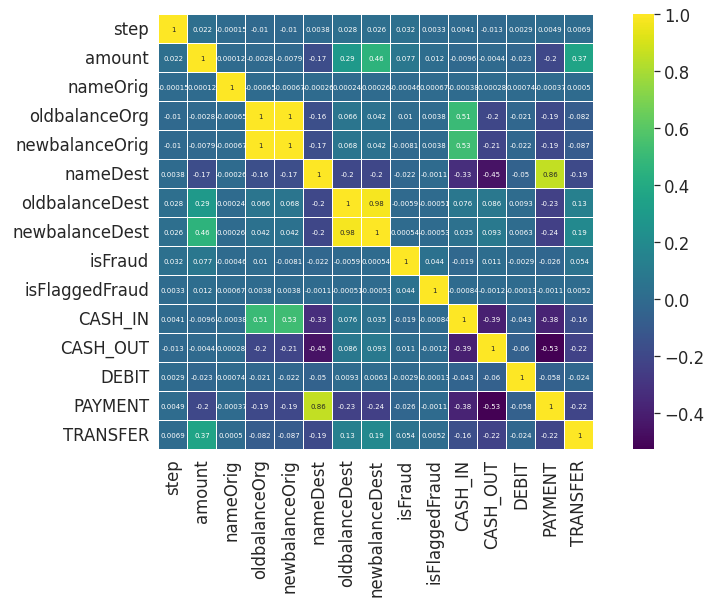


그림 9. heat map

b. One-Hot Encoder

거래 종류를 나타내는 type은 그림 4에서 살펴볼 수 있듯이 CASH\_IN, CASH\_OUT, DEBIT, PAYMENT, TRANSFER 총 5가지가 존재한다. 해당 데이터는 범주형 문자열 데이터이기 때문에 컴퓨터가 인식할 수 있도록 변형해준다. 이때, One-Hot Encoding을 거쳐서 고유값에 해당하는 column에만 1을 표시하고, 나머지 column에는 0을 표시하도록 해준다. 그 결과로 기존 type을 제거하고, 새로운 column 5개를 데이터셋에 추가해주었다.

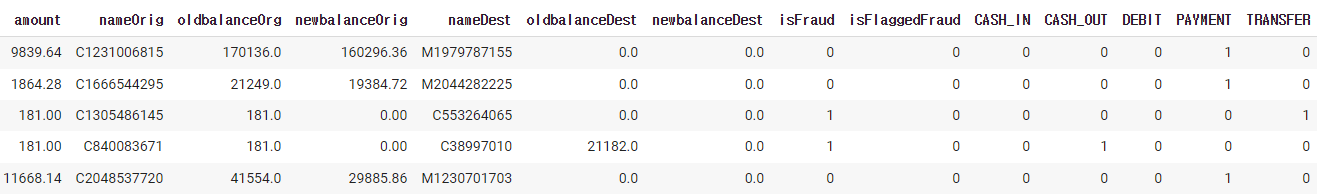


그림 10. One-Hot Encoding 처리 결과

c. Label Encoder

거래시작 고객명 nameOrig과 수신자명 nameDest 또한 문자열 데이터 이므로 수치형으로 변환해준다. nameOrig과 nameDest는 고유값의 개수가 많은 편이므로 Label Encoding을 거치도록 한다. 이때 scikit-learn에서 제공하는 LabelEncoder를 사용하여 데이터를 변환하였다.

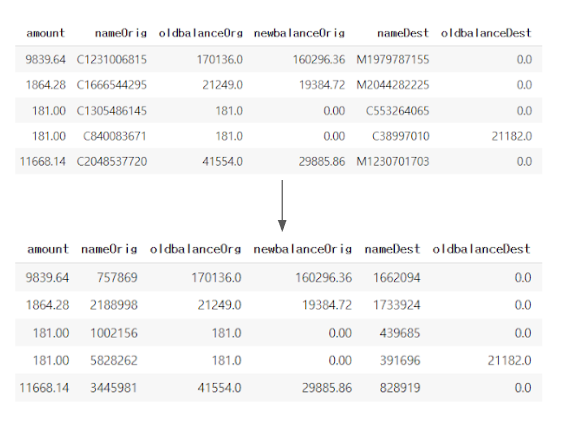


그림 11. Label Encoding 처리 결과

d. Scaling

만약 데이터 값의 범위가 차이가 많이 날 경우, 학습 성능이 떨어질 가능성이 크다. 따라서 이러한 상황을 막기 위해 표준화를 통해 데이터의 값들을 일정한 범위의 값으로 scaling 해준다. scikit-learn에서 제공하는 StandardScaler를 이용해 정규분포 표준화를 시켜주어서 평균이 0, 분산이 1인 값으로 변환해준다. scaling을 처리한 결과, 데이터들의 값이 일정 범위내로 변환된 것을 확인할 수 있었다.

또한, 과대적합을 방지하고 학습속도를 빠르게 하기 위해 normalizer를 사용하여 scaling을 처리하여 데이터 값을 원점으로부터 1만큼 떨어져 있는 범위 내로 변환시켜주었다.

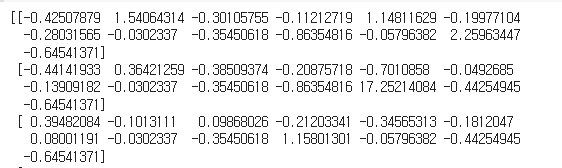


그림 12. Scaling 처리 결과

e. Splitting data set

트레이닝셋과 테스트셋의 비율은 각각 80퍼센트, 20퍼센트로 4:1의 비율로 나누었다.



그림 13. 트레이닝셋과 테스트셋 비율

f. Undersampling

주어진 데이터는 이상 거래 데이터가 정상 거래 데이터보다 많은 불균형 데이터이기 때문에 이를 해결하기 위해서는 데이터의 비율을 조절해주어야 한다. 이를 위해 Undersampling 기법을 활용하였는데, Undersampling은 높은 비율을 차지하는 클래스의 데이터의 개수를 줄이는 것이다. 이를 통해 데이터의 불균형을 어느정도 해소할 수 있지만 데이터를 제거하기 때문에 일부 정보가 사라지게 되는 것이 단점이다.

Undersampler 중 imblearn 모듈에서 데이터를 임의로 제거하는 RandomUnderSampler를 이용해 undersampling을 처리해주었다. Undersampling 이후, 정상 거래와 이상 거래의 비율이 같아진 것을 확인할 수 있다.

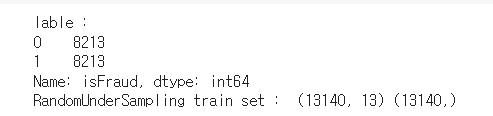


그림 14. UnderSampling 처리 결과

Oversampling 기법도 수행하였으나, 데이터의 크기가 과도하게 커져서 컴퓨터 RAM 성능상 oversampling이 제대로 진행되지 않았다.

g. Learning Curve

Learning curve는 훈련셋의 80퍼센트를 0.1, 0.325, 0.55, 0.775, 1의 비율로 학습시켰다.

train\_sizes, train\_scores, test\_scores = learning\_curve(estimator, x, y, cv=cv, n\_jobs=n\_jobs, train\_sizes=train\_sizes, scoring=s)

가로축은 training size를, 세로축은 loss를 표시하며, train score는 주황색, test score는 파란색 그래프이다.

4. Model Training

이 장에서는 학습을 위해 사용한 모델들을 설명하고 모델 별 학습 결과 값을 설명한다.

전처리 과정을 수행한 데이터들을 가지고 모델의 학습을 진행한다. 본 프로젝트에서 선정한 모델은 정답을 알려주고 학습시키는 방식인 지도학습 알고리즘 6개(KNN, Logistic Regression, SVM, Decision Tree, Random Forest, XGBoost), 여러층을 가진 인공신경망을 사용해 학습시키는 딥러닝 모델 2개(FCN, CNN)이다.

a. KNN(K-Nearest Neighbor)

KNN은 새로운 데이터가 주어지면 기존의 데이터들 가운데 가장 가까운 k개의 이웃의 정보로 새로운 데이터를 예측해내는 방식이다. 지도학습에서 데이터에 대해 여러가지의 값 중 하나의 답을 도출해내는 분류(classification)에 해당한다. KNN은 k값을 무엇으로 정하느냐에 따라 결과 값이 달라질 수 있으며, default값은 5이다. k값이 1일 경우, train 데이터가 예측에 주는 영향이 크기 때문에 불안정한 예측을 만들어낸다. 반대로 k값이 클 경우, train 데이터는 잘 맞지 않을 수 있지만 더 안정된 예측값을 얻을 수 있다. 데이터의 전처리 과정이 중요하며 모델을 별도로 구축하지 않아서 SVM이나 Linear Regression보다 빠르지만 훈련 세트가 클 경우 예측이 느리다는 점 등의 장단점이 있다.

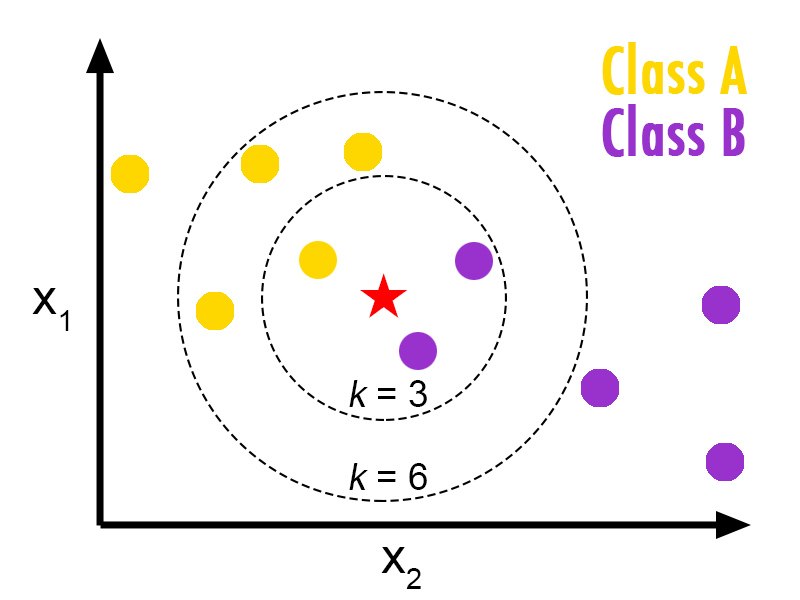


그림 15. KNN(K-Nearest Neighbor)

최적의 성능을 내는 KNN의 파라미터를 찾기 위하여, 파라미터 후보군들 조합에서 가장 성능이 좋은 것을 찾아내줄 수 있는 Grid Search를 이용했다. Grid Search를 통해 k : [3, 5, 7] 중 성능이 가장 좋았던 파라미터는 7이었으며, 하이퍼파라미터는 KNeighborsClassifier(n\_neighbors = 7)이었다.

Recall Score: 0.8784361637141112

Precision Score: 0.899874843554443

Accuracy Score: 0.8907486305538649

F1 Score: 0.8890262751159196

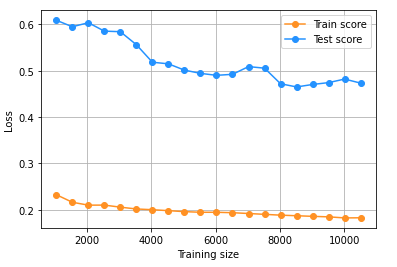


그림 16. KNN 모델 learning curve

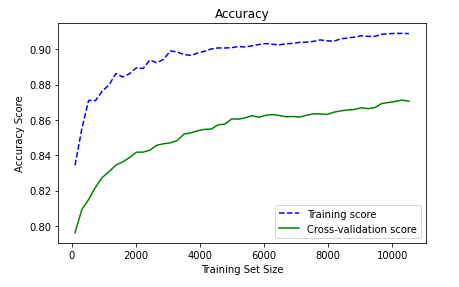


그림 17. KNN 모델 정확도

b. Logistic Regression

Logistic Regression은 샘플이 특정 class에 속할 확률을 추정해 더 가능성이 높은 class로 분류하는 방식으로 지도학습 중 데이터 분석을 통해 특징으로 답을 찾는 회귀(Regression)에 해당한다. 입력 특성의 가중치의 합과 편향을 더하여 결과값의 로지스틱을 출력한다. Logistic Regression에서는 시그모이드(Sigmoid) 함수를 이용해 t>0이면 σ(t)>0.5이 되므로 양성, 반대의 경우 음성이라고 예측한다.

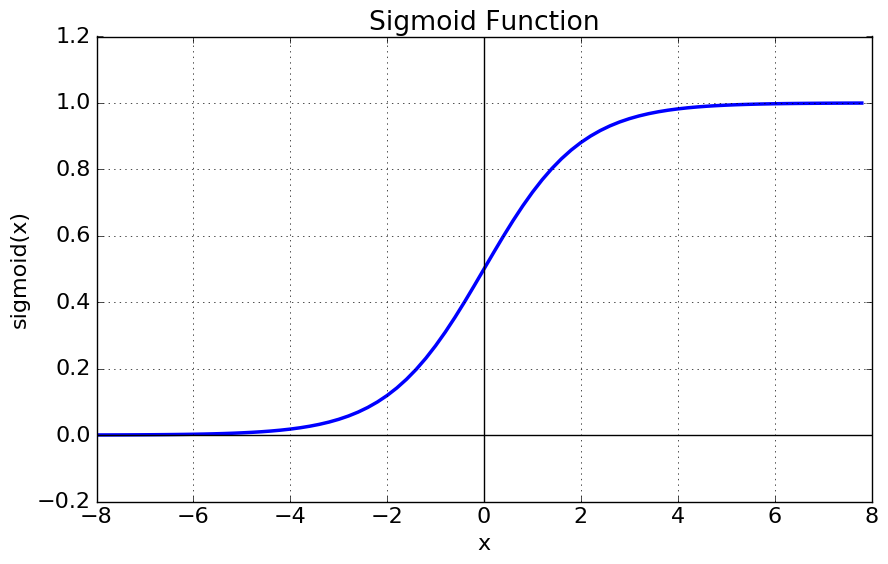


그림 18. Sigmoid 함수

Grid Search를 통해 C : [0.0001, 0.001, 0.01, 0.1, 0, 1, 10, 50, 100] 중 가장 좋은 파라미터로 100이 선정되었으며, Grid Search를 한 번 더 실행하여 C : [95~105] 중 가장 좋은 파라미터로 102가 선정되었다. 하이퍼파라미터는 LogisticRegression(C = 102, random\_state = RANDOM\_SEED)이었다.

Recall Score: 0.9902260232131949

Precision Score: 0.9518496770405167

Accuracy Score: 0.9701765063907486

F1 Score: 0.9706586826347307

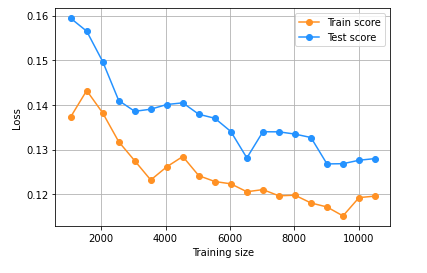
****

그림 19. Logistic Regression 모델 learning curve

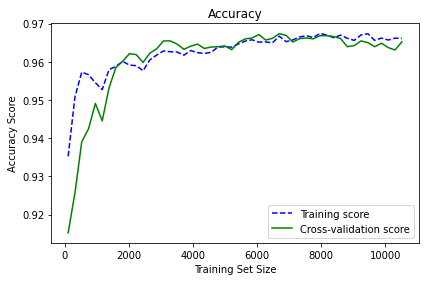


그림 20. Logistic Regression 모델 정확도

c. SVM(Support Vector Machine)

SVM은 데이터들과거리가 가장 먼 초평면을 선택해 분리하는 기계학습 분야 중 하나로, 주로 분류와 회귀 분석을 할 때 사용된다. SVM은 마진(margin)을 최대화 하도록 하는 알고리즘으로, SVM 분류기를 클래스 사이에 가장 폭이 넓은 것을 찾는다. 일반적으로 초고차원 공간에서 공간을 나누는 경계인 초평면을 사용해 양성 class와 음성 class를 구분한다.

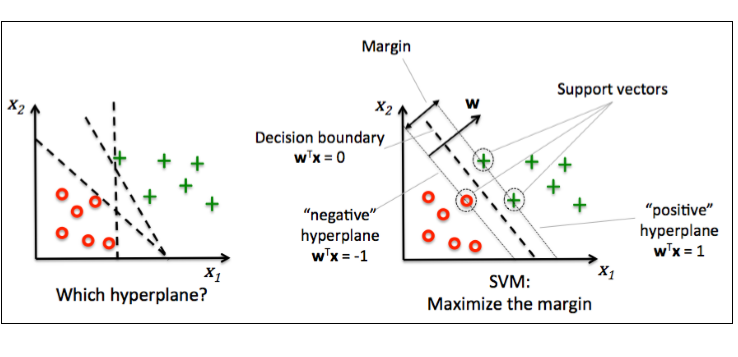


그림 21. SVM(Support Vector Machine)

StandardScaler를 사용했으며, Grid Search를 통해 C : [0.0001, 0.001, 0.01, 0.1, 0, 1, 10, 50, 100] 중 가장 좋은 파라미터로 10이 선정되었으며, Grid Search를 한 번 더 실행하여 C : [2~20] 중 가장 좋은 파라미터로 20이 선정되었다. 커널은 linear로 설정하여 학습시켰다. 하이퍼파라미터는 Pipeline(scaler, StandardScaler(), ‘linear\_svc’, LinearSVC(C=20))이었다.

Recall Score: 0.9743433109346366

Precision Score: 0.9562350119904077

Accuracy Score: 0.9650030432136336

F1 Score: 0.9652042360060515



그림 22. SVM 모델 정확도(C = 20)

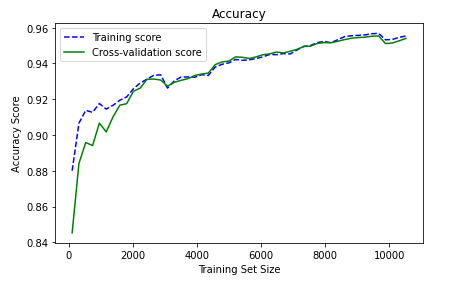


그림 23. SVM 모델 정확도(C = 13)

d. Decision Tree

Decision Tree는 Random Forest의 기본적인 구성 요소로 예/아니오의 질문을 이어가며 학습하면서, target값에 대해 가장 다수의 정보를 가진 것들을 선택하면서 학습을 한다. 샘플에 대한 leaf node를 찾기 위해 tree를 탐색하는데, 샘플이 특정 class k에 속할 확률을 추정할 수 있다. Train 데이터에 있는 작은 변화에도 민감하게 반응하는 경향이 있는 것이 특징이다.

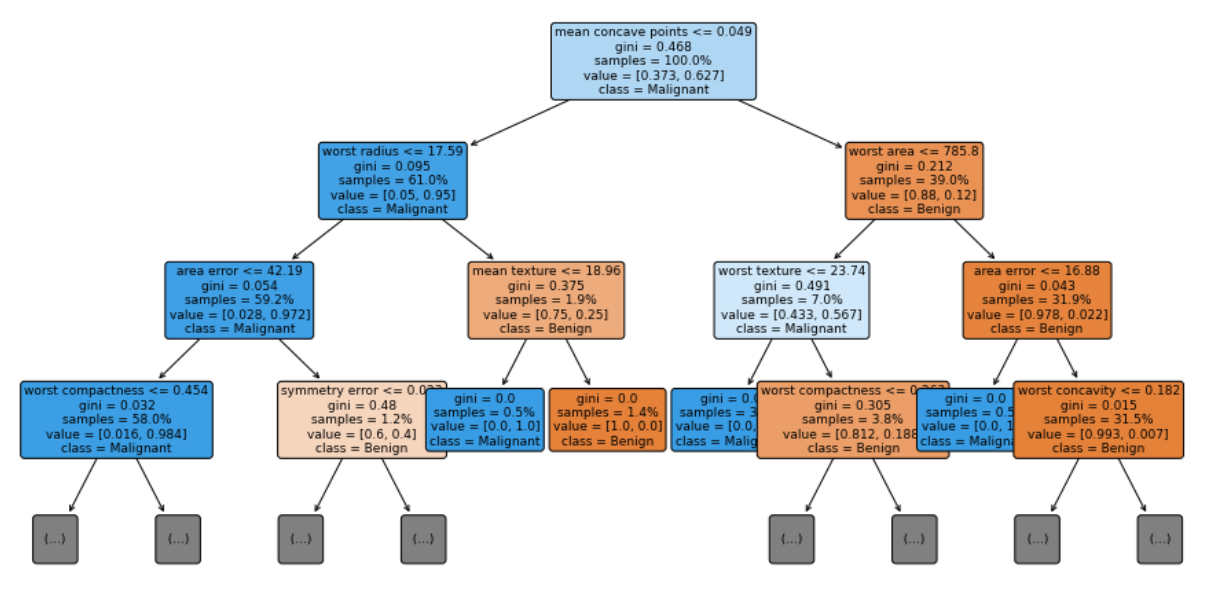


그림 24. Decision Tree

max\_leaf\_nodes : [10, 11, 12, 13, 14, 15] 중에서 14가 best parameter로 선정되었고, max\_ depth : [2, 3, 4, 5, 6, 7] 중에서 5가 best parameter로 선정되었다. 하이퍼파라미터는 DecisionTreeClassifier(max\_depth = 5, max\_leaf\_nodes = 14, random\_state = 0)이었다.

Recall Score: 0.9993891264508247

Precision Score: 0.4980213089802131

Accuracy Score: 0.497869750456482

F1 Score: 0.6647704185290532

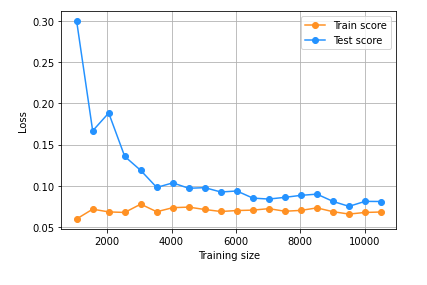


그림 25. Decision Tree 모델 learning curve

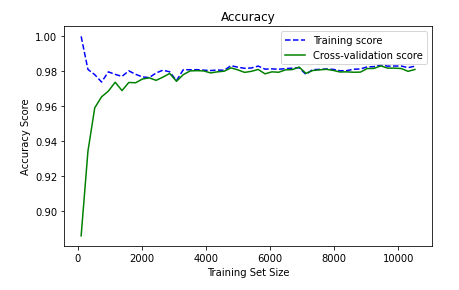


그림 26. Decision Tree 모델 정확도

e. Random Forest

Random Forest는 일반적으로 배깅이나 페이스팅을 적용한 Decision Tree의 앙상블이다. 트리의 노드를 분할할 때, 전체 특성 중 최선의 특성을 찾는 대신 무작위로 선택한 특성의 후보들 중, 최적의 특성을 찾는 방식으로 무작위성을 더 주입한다. 범위 외의 값에서는 정답률이 떨어지는 Decision Tree의 단점을 보완하기 위해 만든 것으로, 특성의 상대적 중요도를 측정하기 쉽다는 장점이 있다.

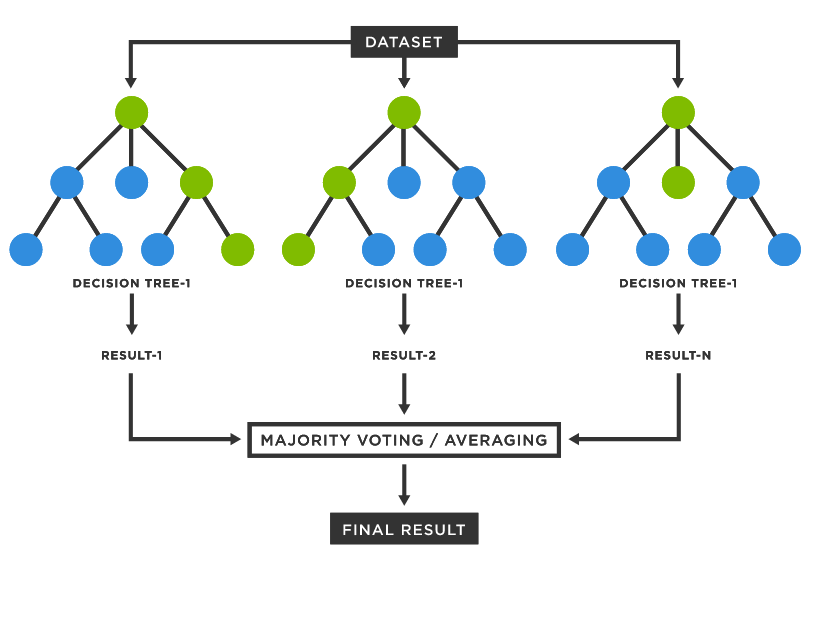


그림 27. Random Forest

Grid Search를 통해 max\_depth : [2, 3, 4, 5, 6, 7], n\_estimators: [20, 40, 60, 80, 100] 중 가장 좋은 파라미터로 max\_depth = 7, n\_estimators = 40이 선정되었다. 하이퍼파라미터는 RandomForestClassifier(n\_estimators = 40, max\_depth = 7, random\_state = RANDOM\_SEED)이었다.

Recall Score: 0.996334758704948

Precision Score: 0.9696789536266349

Accuracy Score: 0.9826536822884967

F1 Score: 0.9828261524555589

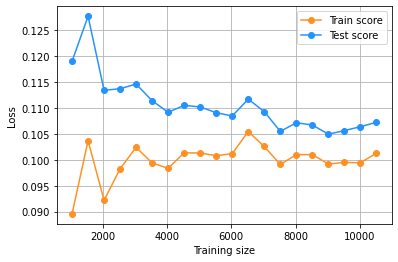


그림 28. Random Forest 모델 learning curve

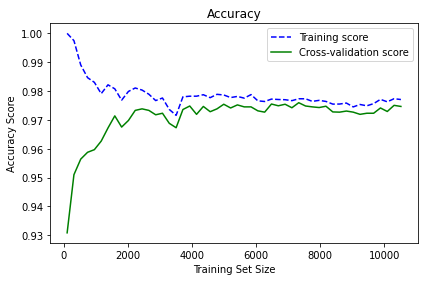


그림 29. Random Forest 모델 정확도

f. XGBoost(Extreme Gradient Boosting)

XGBoost는 여러개의 Decision Tree를 조합하여 사용하는 앙상블 알고리즘으로 분류, 회귀 등의 모델을 제공한다. 이전 모델에서의 실제값과 예측값의 loss를 train 데이터에 넣고, gradient를 이용하여 오류를 보완하는 방식으로 이루어진다. 잘못 분류된 데이터에 가중치를 두면서 순차적으로 학습하는 과정을 통해, 일반적으로 빠른 수행 시간과 뛰어난 예측 성능을 발휘하는 것이 특징이다.

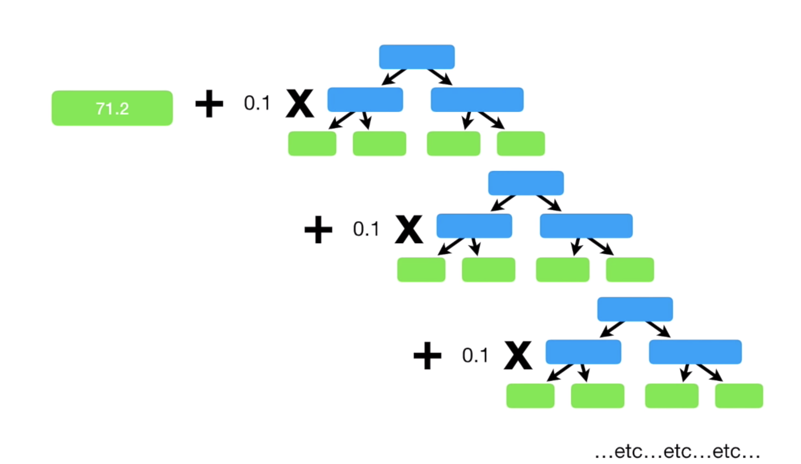


그림 30. XGBoost(Extreme Gradient Boosting)

Grid Search를 통해 max\_depth : [3, 4, 5, 6], n\_estimators : [30, 50, 70, 100], learning\_rate : [0.01, 0.1, 0.15, 0.2], gamma : [0, 0.5, 1] 중 가장 좋은 파라미터로 gamma는 1, learning\_rate는: 0.1, max\_depth는 5, n\_estimators는 100이 선정되었다. 하이퍼파라미터는 XGBClassifier(n\_estimators = 100, learning\_rate = 0.1, gamma = 1, max\_depth = 5, random\_state = RANDOM\_SEED)이었다.  
여기서 learning rate는 이전 결과를 반영하는 비율을 의미하고, gamma는 가지를 추가로 나눌 것인지 아닌지 결정하는 최소 손실 감소 값으로 정의된다.

Recall Score: 0.998167379352474

Precision Score: 0.9909035779260158

Accuracy Score: 0.9945222154595252

F1 Score: 0.9945222154595252

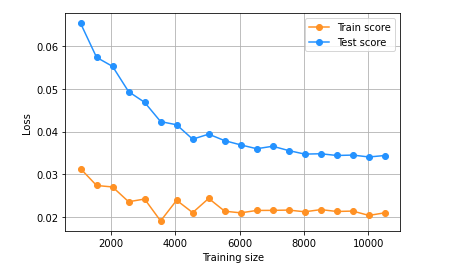


그림 31. XGBoost 모델 learning curve

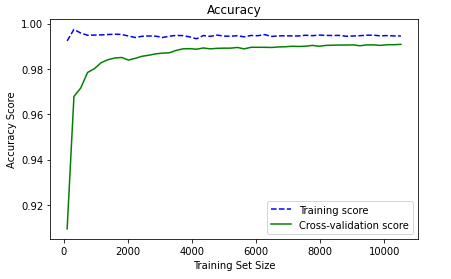


그림 32. XGBoost 모델 정확도

g. FCN(Fully Connected Neural Network)

완전 연결 신경망은 인공 신경망의 한 종류로, 다층 퍼셉트론을 의미한다. 완전 연결 신경망은 노드 간에 횡적으로, 혹은 종적으로 2차원적 연결을 이룬다. FCN은 각 계층간에 모든 뉴런들이 연결된 connected layer의 모임으로 구성되어 있으며, 입력에 대해 특별한 가정이 필요하지 않다는 점이 있어 광범위하게 적용할 수 있지만 특정 구조에 맞추어 형성된 신경망보다는 성능이 약하다는 약점이 있다.

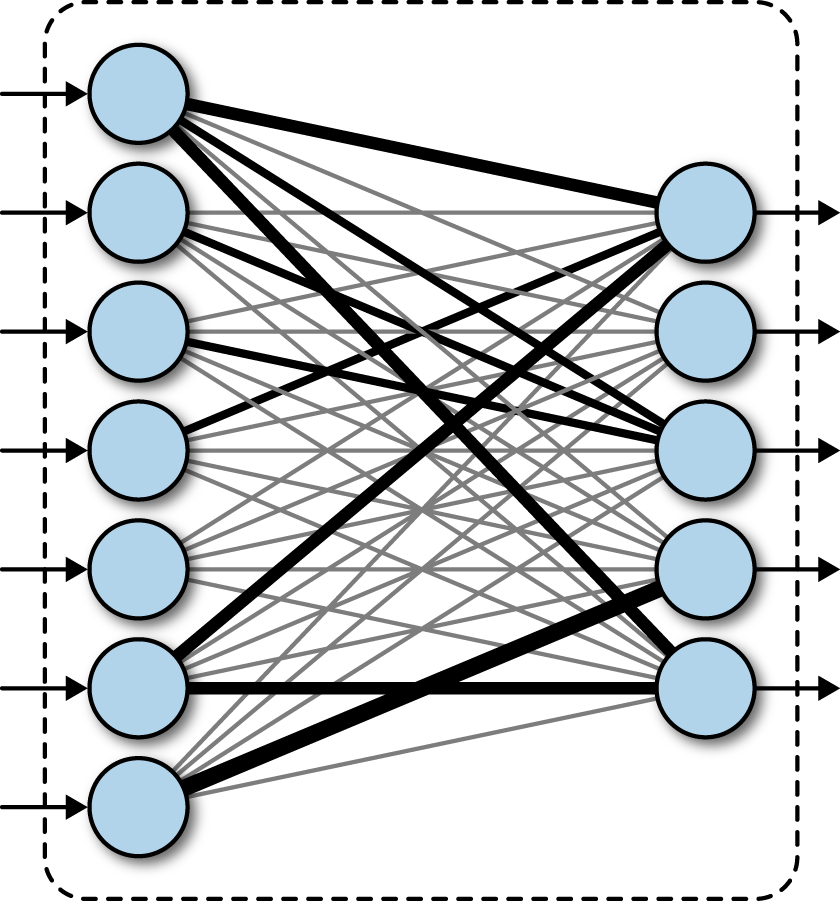


그림 33. Fully Connected Layer

FCN을 학습시키면서 epoch는 100, 배치 크기는 32, dropout 비율은 0.5로 설정하였다. 하이퍼파라미터는 optimizer = Adam(lr = 0.0001, loss = ‘binary\_crossentropy’, metrics = [‘accuracy’, precision, recall, f1score])이었다.

Recall Score: 0.8916

Precision Score: 0.8512

Accuracy Score: 0.8250

F1 Score: 0.8539

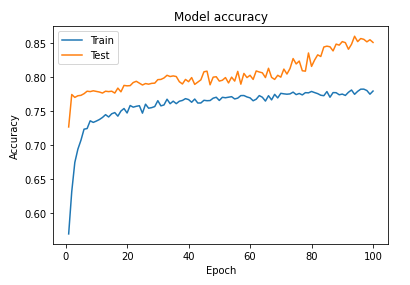


그림 34. 완전 연결 신경망 모델 정확도

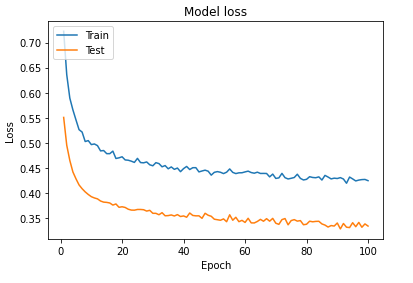


그림 35. 완전 연결 신경망 Loss

h. CNN(Convolutional Neural Network)

CNN은 이미지 분석을 위한 패턴을 찾는데에 유리한 알고리즘으로 행렬로 표현된 필터의 각 요소가 데이터 처리에 적합하도록 자동으로 학습되게 하는 것이다. CNN은 크게 input 계층, output 계층, convolution 계층, pooling 계층, FC 계층 등으로 구성된다.

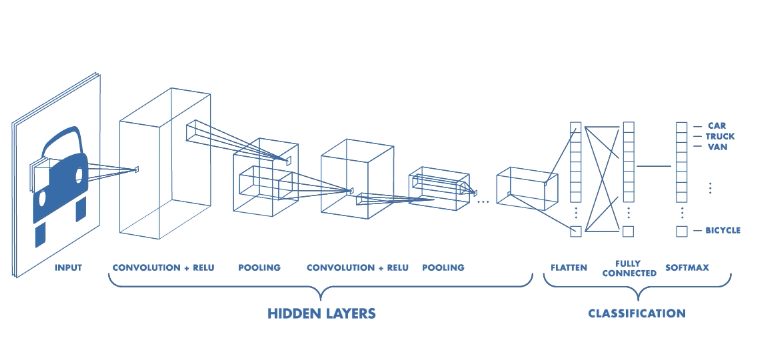


그림 36. CNN(Convolutional Neural Network)

하이퍼 파라미터는 optimizer = Adam(learning\_rate = 0.001, loss = ‘binary\_crossentropy’, metrics = METRICS, epoch = 60, batch size = 2048, activate = ‘relu’, ‘sigmoid’, dropout = 0.2)이었다.

Recall Score: 0.9981

Precision Score: 0.9945

Accuracy Score: 0.9909

F1 Score: 0.9945

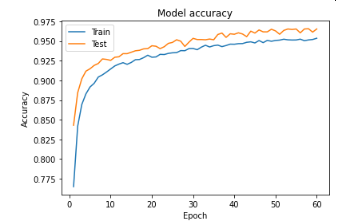


그림 37. CNN 모델 정확도

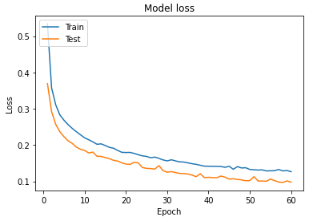


그림 38. CNN 모델 Loss

5. Evaluation

이 장에서는 성능 평가 지표의 종류와 성능 평가 결과를 정리한다.

a. 평가 지표

총 4가지의 지표를 활용하여 학습 결과를 평가하였다. True를 True로 판별한 비율인 정확도, 실제 True 비율인 정밀도, 실제 True 중 True로 예측된 비율인 재현율, 최종 성능 평가지표인 F1 score를 활용하였다.

b. 학습 결과

아래 표 1은 모델별 성능 평가 결과를 정리한 것이다.

학습 결과 가장 성능이 좋았던 모델은 XGBoost이며 , 가장 성능이 좋지 않았던 모델은 Decision Tree이다. 특히 Decision Tree 모델의 경우 재현율은 0.9981로 모든 모델을 통틀어 가장 높았지만, 다른 정확도, 정밀도, F1 score에서 낮은 성능을 보인다.

| Algorithm | Accuracy | Precision | Recall | F1 |
| --- | --- | --- | --- | --- |
| KNN | 0.8907 | 0.8998 | 0.8784 | 0.8890 |
| Logistic Regression | 0.9707 | 0.9518 | 0.9908 | 0.9706 |
| SVM | 0.9743 | 0.9556 | 0.9743 | 0.9649 |
| Decision Tree | 0.4978 | 0.4980 | 0.9993 | 0.6647 |
| Random Forest | 0.9826 | 0.9556 | 0.9963 | 0.9828 |
| XGBoost | 0.9945 | 0.9909 | 0.9981 | 0.9945 |
| FCN | 0.8512 | 0.8250 | 0.8916 | 0.8539 |
| CNN | 0.9665 | 0.9685 | 0.9657 | 0.9660 |

표 1. 모델별 성능 평가 결과

6. Improvement

이 장에서는 학습시켰던 결과를 바탕으로 개선할 부분을 설명한다.

Oversampling을 시도하였으나 데이터셋의 크기가 너무 커 RAM의 과부하로 인해 실행이 불가능하여 undersampling과의 비교를 하지 못했다. 또한 몇몇 모델에서의 overfitting 문제가 의심이 된다.

데이터셋 자체를 분할하여 여러 개의 case로 나눈 뒤 훈련셋과 테스트셋으로 나누어 학습했다면 oversampling을 하더라도 과부하를 줄일 수 있고 훈련셋과 테스트셋에 지나치게 적합해진 overfitting 문제를 어느정도 해결할 수 있었을 거라 판단한다.

또한, 정밀도 문제로 훈련셋에서만 undersampling을 하는 것이 이상적이지만 훈련셋과 테스트셋에서 모두 undersampling을 수행하였다. 그림 39와 그림 40은 훈련셋에만 undersampling을 했을 때와 훈련셋과 테스트셋 모두에 undersampling을 수행한 결과를 나타낸 것이다. 그림에서 확인할 수 있듯이 정밀도가 훈련셋에만 undersampling을 한 경우 0.0190 정도로 모두에 undersampling을 한 경우인 0.9531에 비해 정밀도가 급격히 떨어졌다.

이는 정상 데이터와 이상 데이터의 데이터 불균형으로 인한 문제로 추측된다. 조금 더 많은 이상 데이터를 수집하거나 oversampling으로 데이터 비율을 조금 더 조절한다면 더 나은 성능을 보일 수 있을 것이라 생각된다.

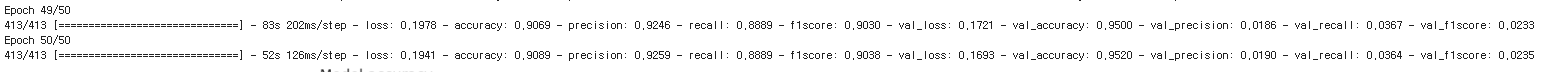


그림 39. 훈련셋에만 undersampling을 한 경우

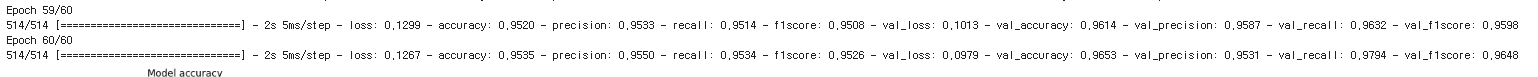


그림 40. 훈련셋과 테스트셋에 undersampling을 한 경우

7. Reference

1. K-Nearest Neighbor(KNN) Explained, https://www.pinecone.io/learn/k-nearest-neighbor/

2. Understanding Random Forest, https://towardsdatascience.com/understanding-random-forest-58381e0602d2

3. Complete Guide to Parameter Tuning in XGBoost with codes in Python, https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

4. XGBoost Part 1 (of 4): Regression, https://www.youtube.com/watch?v=OtD8wVaFm6E

5. [TensorFlow for Deep Learning by Bharath Ramsundar, Reza Bosagh Zadeh](https://learning.oreilly.com/library/view/tensorflow-for-deep/9781491980446/), https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html

6. convolutional neural network,

https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network