

PROJECT

인공지능과 정보보호 기말 프로젝트

The UNSW-NB15 Dataset

2022111319
2022*****

박정은
이성민

CONTENT

01 | 데이터셋 소개

02 | 전처리

03 | 머신러닝

04 | 딥러닝

05 | 성능평가

06 | 결론

UNSW-NB15

네트워크 트래픽의 정상 및 비정상 패턴을 학습 및 예측하는 데 사용되는 공개 데이터셋으로,
특히 침입 탐지 시스템(IDS) 개발과 성능 평가에 널리 활용됨

주요 특징

- 네트워크 패킷 및 연결 정보를 포함
- 다양한 네트워크 트래픽 시나리오(정상/비정상)
제공
- 공격 유형이 세분화되어 있으며, 실제 네트워크
환경에서 발생할 수 있는 다양한 공격을 반영

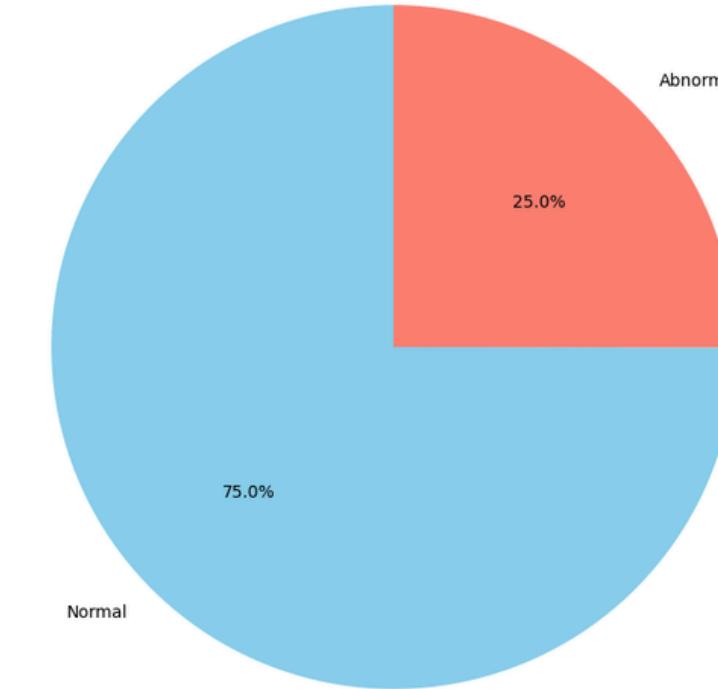
선정 이유

- 최신의 네트워크 트래픽 패턴과 9가지 유형의 공격
포함
- 현재 네트워크 환경 반영
- 다양한 특성 포함
- 네트워크 트래픽의 다양성을 포함하며, 실제와 유사
한 정상 및 공격 시나리오를 포함

기본 데이터셋 정보

- 샘플 수: 257,672
- 특성 수: 45
 - 41개의 정수형 또는 실수형 데이터
 - 4개의 범주형 데이터 (proto, service, state, attack_cat)

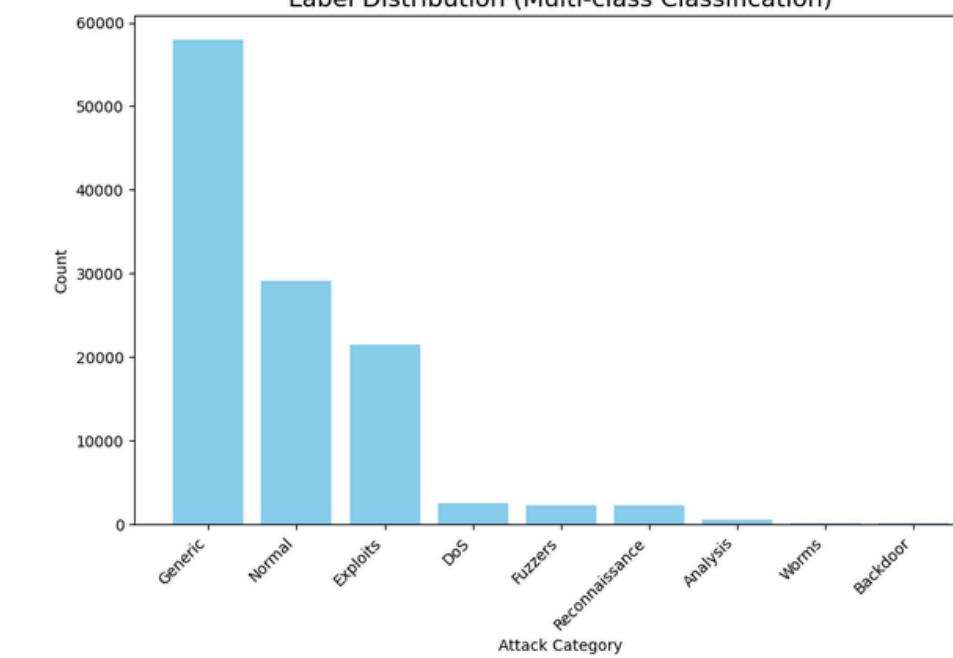
Label Distribution (Binary Classification)



클래스 분포

- 정상(Normal) 데이터와 9개 공격 유형으로 구성
 - 정상(Normal) 트래픽: 75%
 - 비정상 트래픽: 25%
- 소수 클래스(Backdoor, Worms)의 비율이 매우 낮음

Label Distribution (Multi-class Classification)



전처리

01

결측치 처리

데이터의 구조와 분포 확인 후 결측치 및 이상치 확인
결측치 제거, 필요 없는 컬럼 삭제

02

범주형 데이터 처리

레이블 인코딩으로 범주형 데이터를 숫자로 치환
원-핫 인코딩으로 여러 값 중 하나만 활성화

03

정규화

StandardScaler, MinMaxScaler 활용
서로 다른 피처 값의 범위가 일치하도록 조정

04

클래스 불균형 해결

SMOTE 활용
소수 클래스 새로운 합성 샘플을 생성

05

주요 특성 선택

Random Forest, Decision Tree, SelectKBest 활용
필요한 특징만을 선택

전처리

결측치 처리

```
# 불필요한 컬럼 제거  
data.drop(['id'], axis=1, inplace=True)
```

결측 데이터는 모델 훈련에 영향을 미칠 수 있어 처리 필요
service 컬럼에서 값이 '-'로 나타나는 결측 데이터를 확인

```
data[data['service']=='-']
```

	dur	proto	service	state	:
0	0.121478	tcp	-	FIN	
1	0.649902	tcp	-	FIN	
2	1.623129	tcp	-	FIN	

범주형 데이터 처리

범주형 데이터를 수치형 데이터로 변환
해당 범주에 속하면 True(1) 그렇지 않으면 False(0)

```
data_cat = pd.get_dummies(data_cat,columns=cat_col)
```

```
data_cat.head()
```

```
proto_tcp proto_udp service_dhcp service_dns
```

3	True	False	False	False
11	True	False	False	False
15	False	True	False	False
17	True	False	False	False
21	True	False	False	False

전처리

정규화

훈련 시 스케일 차이로 인해 가중치가 왜곡되는 문제를 방지
데이터셋에서 수치형 컬럼만 선택
타겟 변수 label은 정규화 대상에서 제외

```
num_col = list(data.select_dtypes(include='number').columns)
num_col.remove('label')
print(num_col)
```

```
['dur', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'rate', 'sttl',
```

평균이 0, 표준편차가 1인 정규분포로 데이터를 변환

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
data[num_col] = scaler.fit_transform(data[num_col])
```

다중 분류에서 Normal 클래스 제거 및 레이블 인코딩
데이터 분할 (Train 80% Test 20%)

```
# 다중분류에서 Normal 클래스 제거
data_multi = data[data['attack_cat'] != 'Normal']

# 다중분류 라벨 인코딩 (attack_cat)
label_encoder = LabelEncoder()
data_multi['attack_cat_encoded'] = label_encoder.fit_transform(data_multi['attack_cat'])

# 이중분류 타겟 (label: 0 -> 정상, 1 -> 비정상)
y_binary = data['label']

# 다중분류 타겟 (공격 유형 포함)
y_multi = data_multi['attack_cat_encoded'] # 다중분류는 data_multi를 기반으로 사용

# 입력 데이터에서 타겟 컬럼 제거
X_binary = data.drop(['label', 'attack_cat'], axis=1) # 이중분류용
X_multi = data_multi.drop(['label', 'attack_cat', 'attack_cat_encoded'], axis=1) # 다중

# 데이터 분할 (이중분류)
X_train_binary, X_test_binary, y_train_binary, y_test_binary = train_test_split(
    X_binary, y_binary, test_size=0.2, stratify=y_binary, random_state=42
)

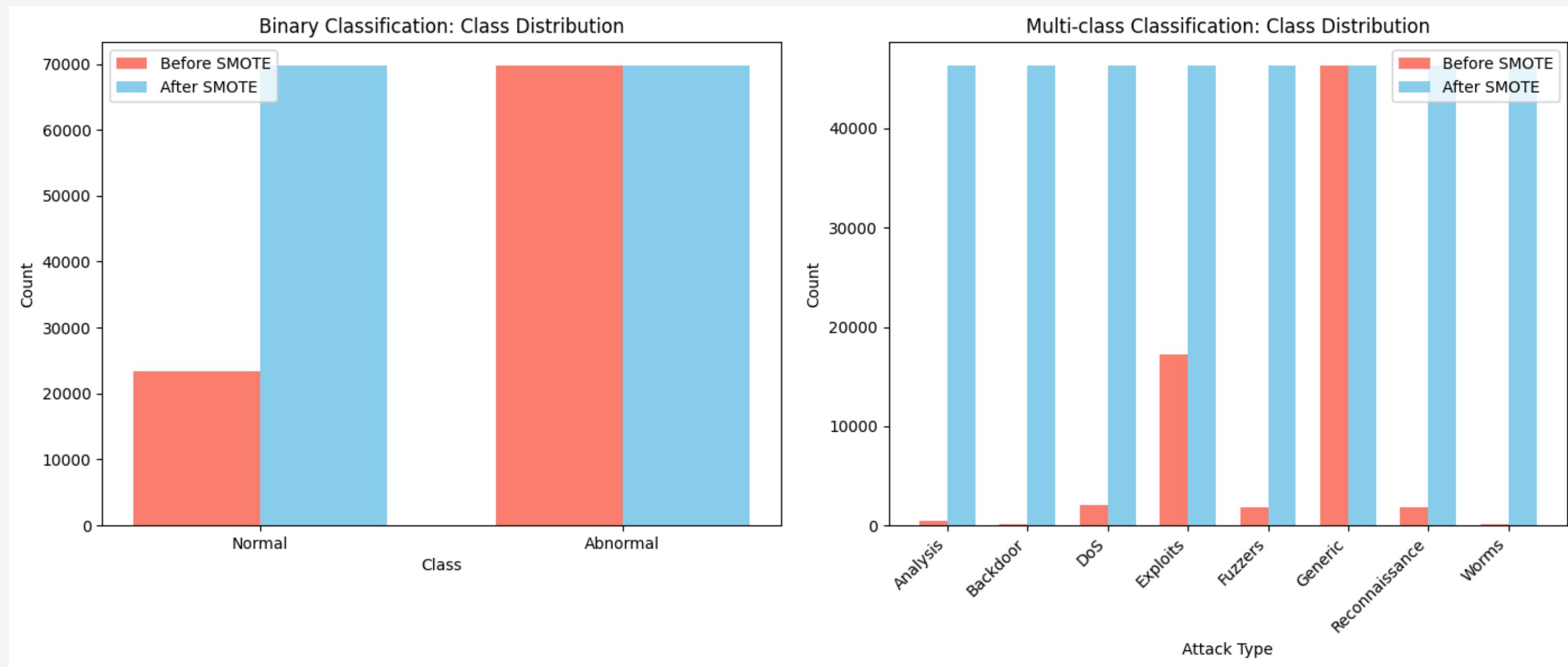
# 데이터 분할 (다중분류)
X_train_multi, X_test_multi, y_train_multi, y_test_multi = train_test_split(
    X_multi, y_multi, test_size=0.2, stratify=y_multi, random_state=42
)
```

전처리

SMOTE (Synthetic Minority Over-sampling Technique)

데이터 불균형 문제를 해결하기 위한 오버샘플링 기법

작은 샘플(소수 클래스)의 데이터를 단순 복제하지 않고, 새로운 데이터를 생성하여 클래스 간 샘플 균형을 맞춤

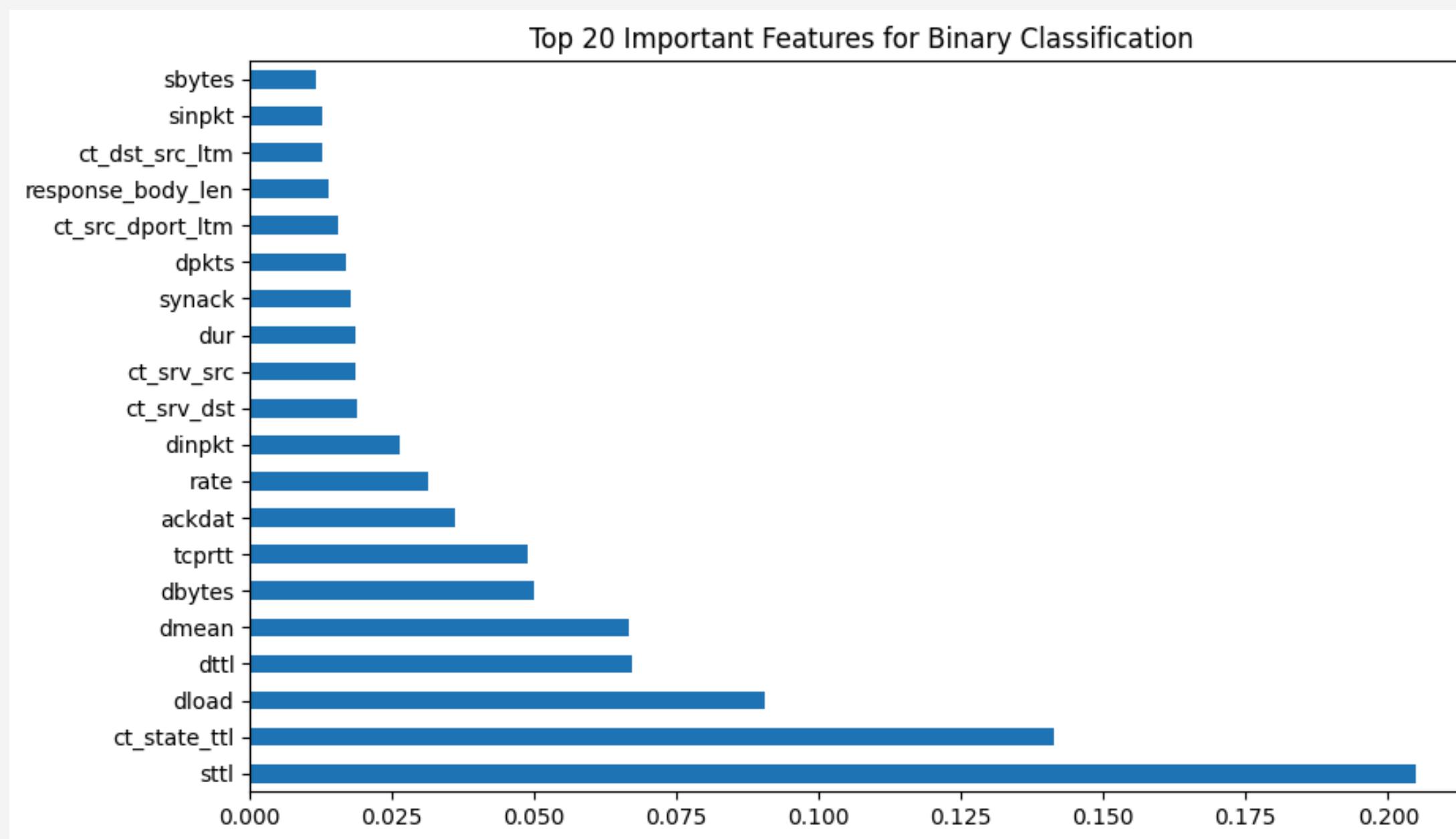


전처리

Feature Selection - Binary Classification

TTL(sttl, dttl), 데이터 크기(dload, dmean), 연결 상태(ct_state_ttl), 패킷 간 간격(sinpkt, dinpkt)

=> 공격 시 네트워크 동작이 변형되기 때문에 비정상 트래픽 탐지에 핵심적인 역할

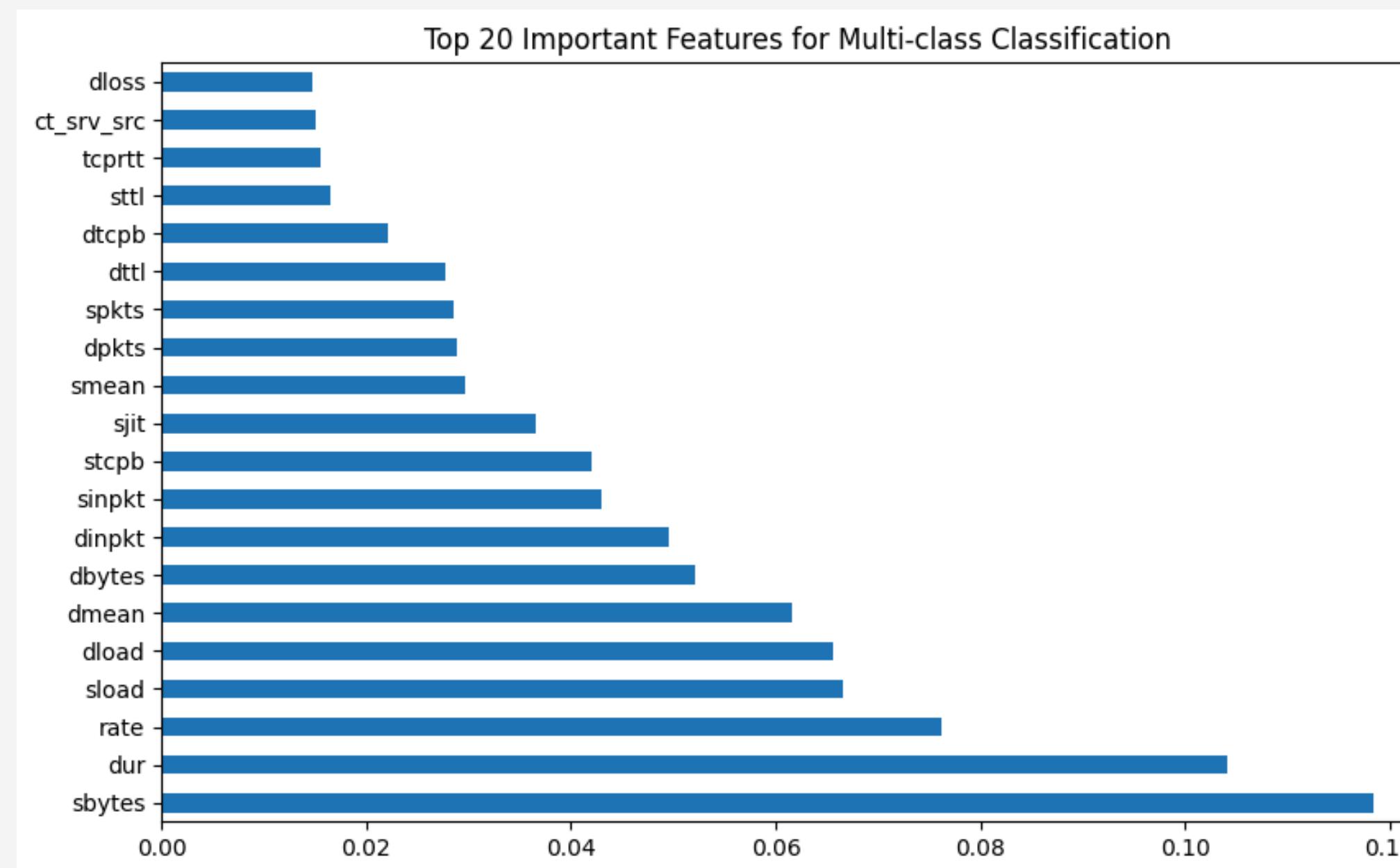


전처리

Feature Selection - Multi-class Classification

발신/수신 바이트(sbytes, dbytes), 부하(sload, dload), 연결 시간(dur), 전송 속도(rate)

=> 트래픽 특성을 기반으로 공격 유형을 구분하는 데 핵심적인 역할



선정한 모델

Decision Trees

- 클래스 불균형 문제 해결에 효과적
- 특성 중요도 분석 가능
 - 45개의 특성 중 중요한 특성을 파악하여 모델 성능 최적화 가능

Random Forest

Extra Trees

XGBoost

- 네트워크 침입 탐지에서 중요한 높은 정확도와 낮은 오탐률을 만족
- 대규모 데이터셋 처리에 효율적
- 다양한 공격 유형 탐지에 적합

선정한 모델

LSTM

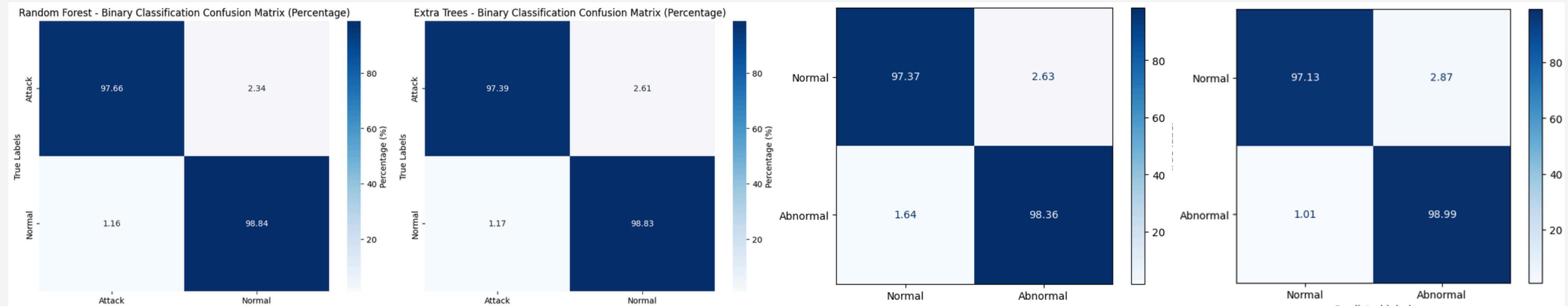
GRU

- 시간 의존성 학습
 - 네트워크 트래픽 데이터의 시계열적 특성을 효과적으로 포착
 - 장기 의존성 파악
 - 복잡한 공격 패턴의 시간적 연관성을 학습 가능
-

CNN

- 지역적 패턴 탐지
 - 네트워크 트래픽의 특정 프로토콜이나 연결 패턴과 같은 지역적 특징을 효과적으로 학습
- 특성 간 상호작용 학습

머신러닝 - 이진 분류

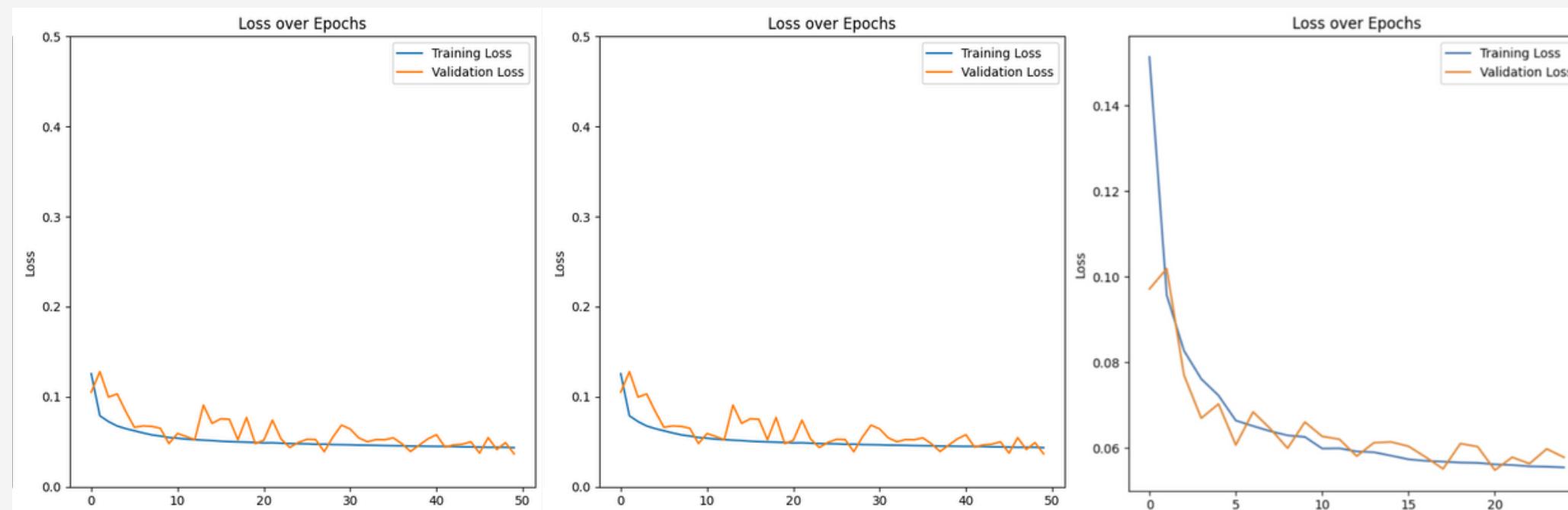


Random Forest

Extra Trees

Decision Trees

XGBoost



LSTM

GRU

CNN

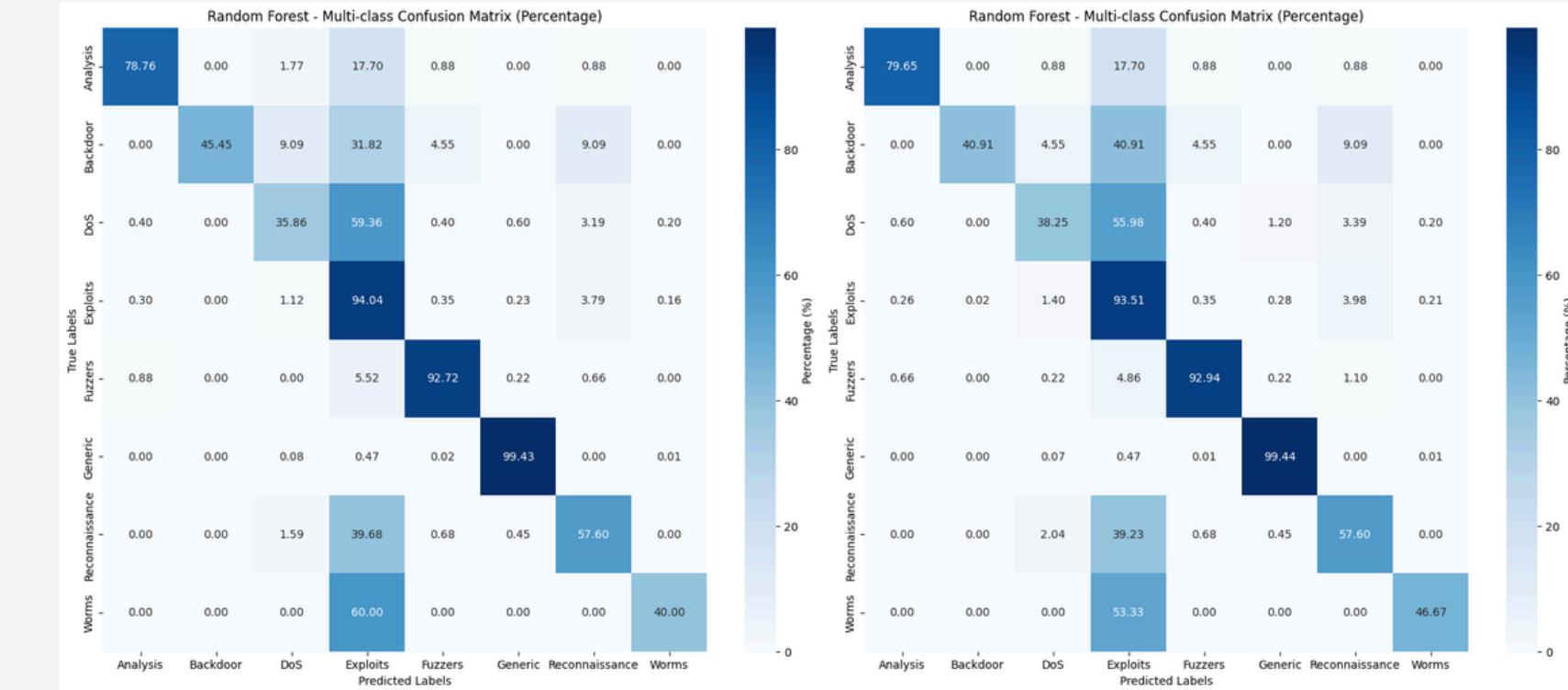
- 이진 분류 결과는 Precision, Recall, F1-Score 모두 약 97~99%의 높은 성능을 기록
- Training Loss와 Validation Loss가 지속적으로 감소하며, 안정적으로 수렴

머신러닝 - 다중 분류

Random Forest

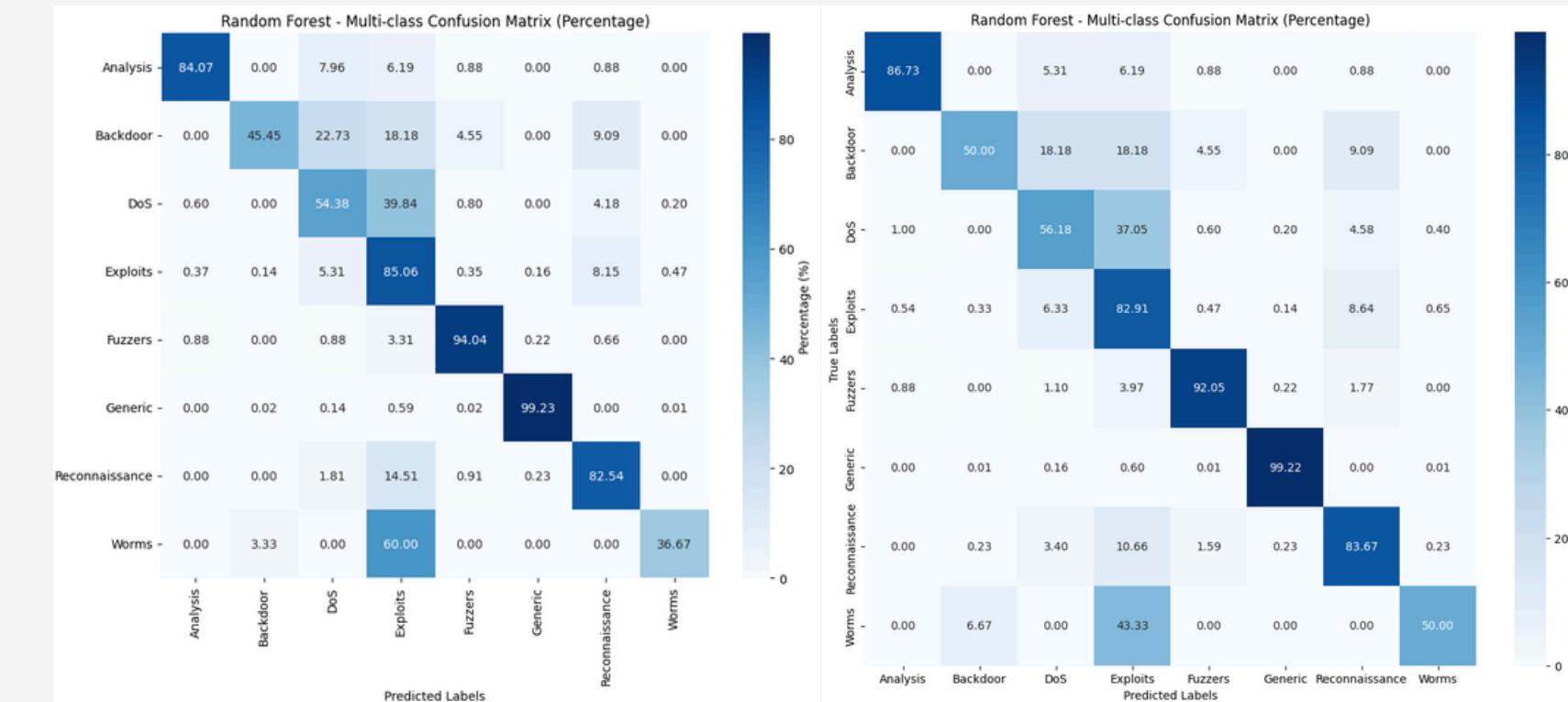
V4 분석

- 소수 클래스 탐지 성능
 - Backdoor +4.55%, Reconnaissance +26.07%, Worms +10.00%로 대폭 개선
- 주요 클래스(Generic): 99.22% 성능 유지.



V1 전처리만

V2 특성추출



V3 SMOTE

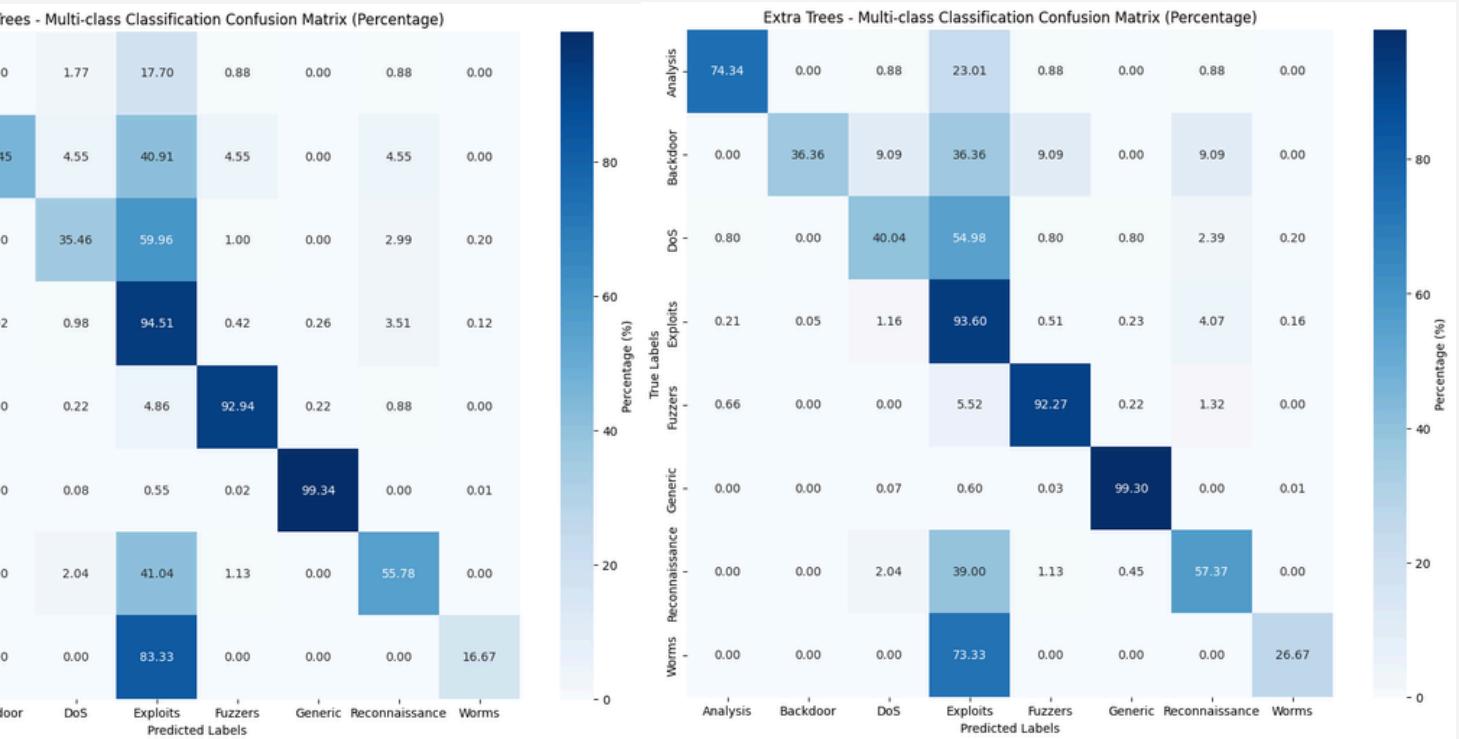
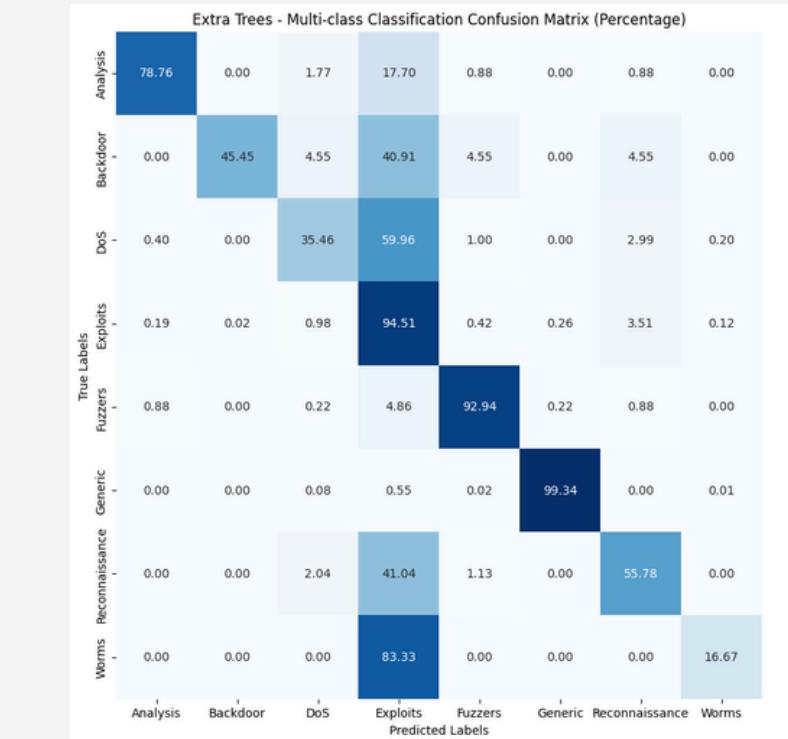
V4 특성추출 + SMOTE

머신러닝 - 다중 분류

Extra Trees

V4 분석

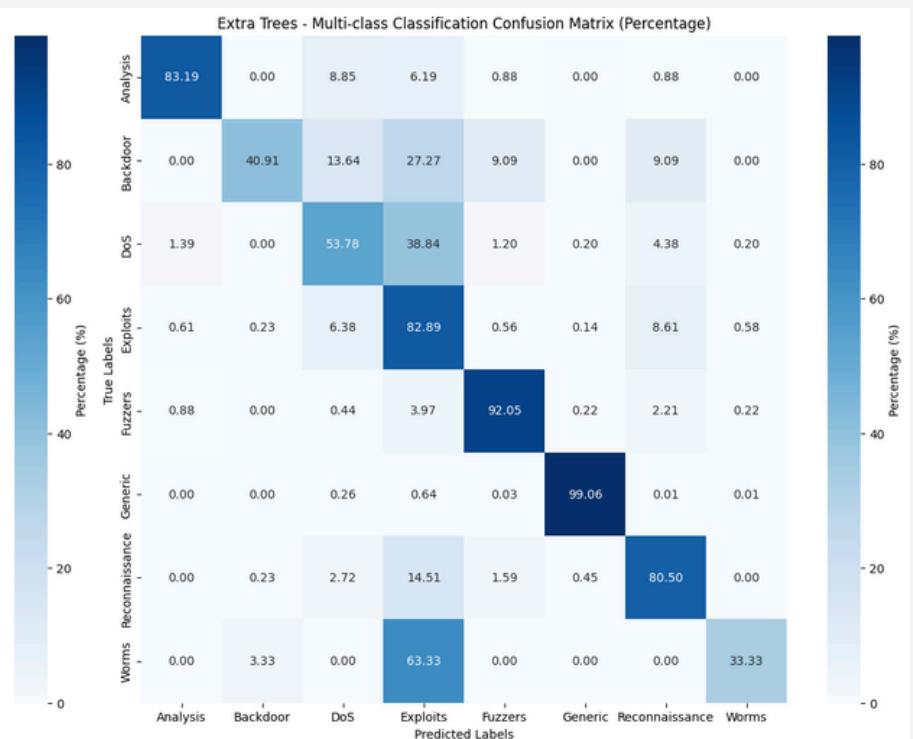
- 소수 클래스 탐지 성능:
 - Backdoor +9.09%, Reconnaissance +24.72%, Worms +16.66%로 대폭 개선.
- 주요 클래스(Generic, Exploits):
 - Generic 99.34%로 안정적 유지, Exploits +11.62% 개선.



V1 전처리만



V3 SMOTE



V4 특성추출 + SMOTE

머신러닝 - 다중 분류

하이퍼파라미터 튜닝

Decision Trees

```
# 1. 하이퍼파라미터 범위 설정
param_dist = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5, 10],
    'max_features': [None, 'sqrt', 'log2']
}

# 2. RandomizedSearchCV 객체 생성
random_search = RandomizedSearchCV(
    estimator=DecisionTreeClassifier(random_state=42),
    param_distributions=param_dist,
    n_iter=20, # 샘플링 횟수
    scoring='f1',
    cv=3, # 교차 검증 폴드 수
    verbose=1,
    random_state=42
)
```

Decision Tree의 분할 기준과 복잡도를 조정하여
최적의 성능 탐색

XGBoost

```
# 1. 하이퍼파라미터 범위 설정
param_dist = {
    'learning_rate': [0.01, 0.1, 0.2, 0.3],
    'n_estimators': [50, 100, 200, 300],
    'max_depth': [3, 5, 7, 10],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'reg_alpha': [0, 0.1, 1, 10],
    'reg_lambda': [1, 10, 50, 100]
}

# 2. RandomizedSearchCV 객체 생성
random_search = RandomizedSearchCV(
    estimator=XGBClassifier(use_label_encoder=False,
                           eval_metric='logloss', random_state=42),
    param_distributions=param_dist,
    n_iter=20, # 샘플링 횟수
    scoring='f1', # 이진분류에서 F1 점수를 기준으로 최적화
    cv=3, # 교차 검증 folds
    verbose=1,
    random_state=42
)
```

XGBoost의 학습률, 부스팅 반복 횟수, 정규화를
포함한 주요 파라미터를 설정

머신러닝 - 다중 분류

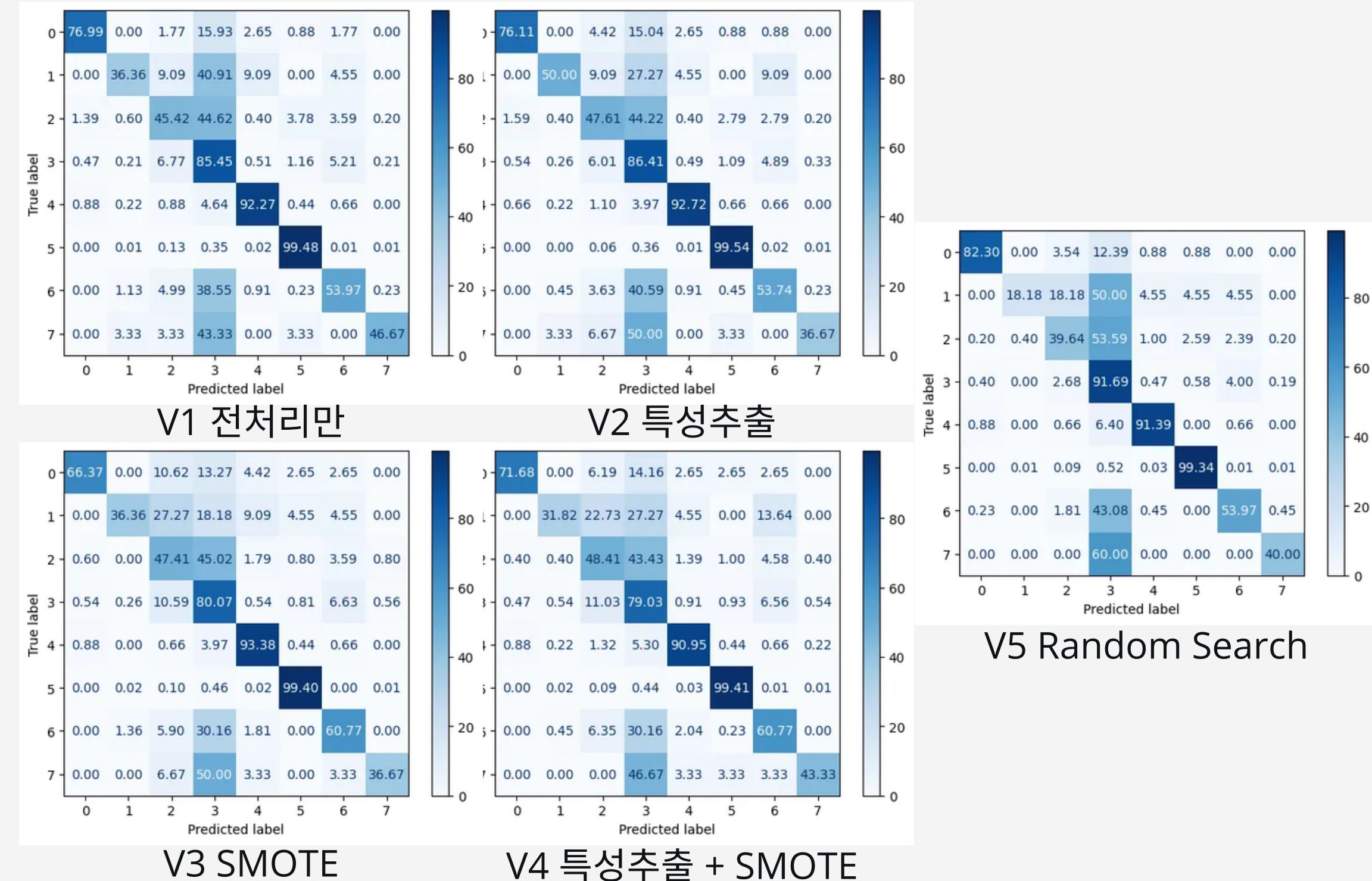
Decision Tree

소수 클래스(공격 유형) 탐지

- 특성 추출 + SMOTE 모델이 소수 클래스 (Backdoor, DoS)에서 재현율과 F1 점수를 개선
- 다수 클래스에서도 적절한 성능을 유지하여 침입 탐지 시스템(IDS)에 적합

전반적인 정확도

하이퍼파라미터 튜닝을 적용한 모델이 가장 높은 정확도(0.94)로 다수 클래스에서 성능이 우수



*Class Mapping
0: Analysis, 1: Backdoor, 2: DoS, 3: Exploits,
4: Fuzzers, 5: Generic, 6: Reconnaissance, 7: Worms

머신러닝 - 다중 분류

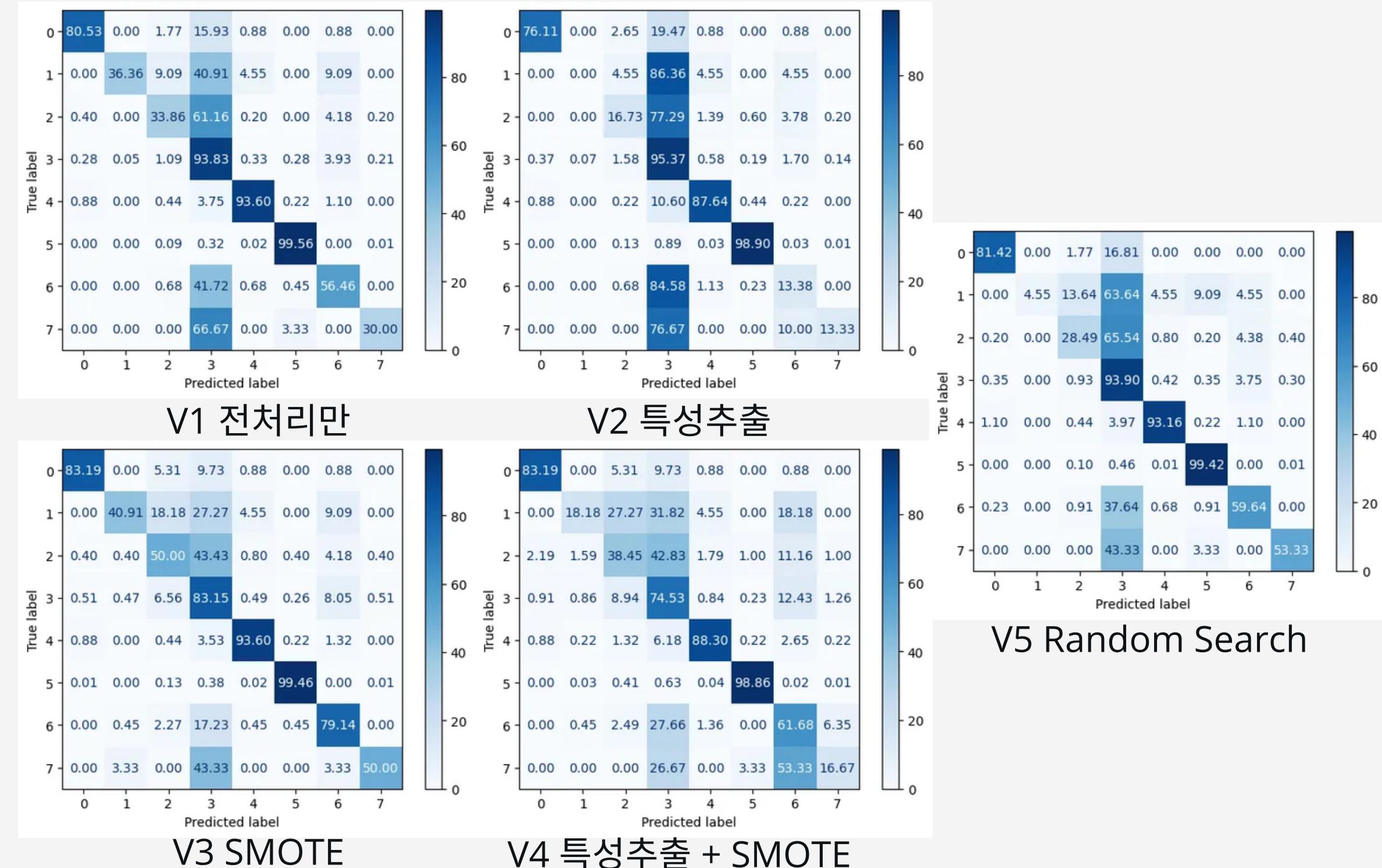
XGBoost

소수 클래스(공격 유형) 탐지

- SMOTE 적용 모델이 소수 클래스 (Reconnaissance, Worms)에서 재현율과 F1 score 개선
- 데이터 불균형 문제를 효과적으로 해결 가능

전반적인 정확도

기본 모델 또는 하이퍼파라미터 튜닝을 적용한 가장 높은 정확도(0.95)로 다수 클래스에서 가장 높은 정확도와 성능을 유지



*Class Mapping

0: Analysis, 1: Backdoor, 2: DoS, 3: Exploits,
4: Fuzzers, 5: Generic, 6: Reconnaissance, 7: Worms

딥러닝 - 다중 분류

LSTM

```
# 데이터 3D로 변환
X_train_binary_lstm = X_train_binary_balanced.values.reshape(X_train_binary_balanced.shape[0], 1, X_train_binary_balanced.shape[1])
X_test_binary_lstm = X_test_binary_selected.values.reshape(X_test_binary_selected.shape[0], 1, X_test_binary_selected.shape[1])

# 모델 훈련 및 성능 평가
start_train = time.time()

# LSTM 모델 정의
lstm_binary = Sequential([
    LSTM(64, input_shape=(1, X_train_binary_balanced.shape[1]), return_sequences=True),
    LSTM(32),
    Dense(1, activation='sigmoid')
])

# 모델 컴파일
lstm_binary.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# 훈련 (검증 세트를 사용하여 성능 평가)
history = lstm_binary.fit(
    X_train_binary_lstm, y_train_binary_balanced,
    epochs=50, batch_size=64,
    validation_split=0.2, # 20%의 데이터를 검증 세트로 사용
    verbose=1
)

end_train = time.time()
```

데이터 3D로 변환

- 입력 데이터는 3D 텐서 형태를 필요
- 데이터를 (샘플 수, 시간 스텝, 특성 수)로

모델 정의

첫 번째 레이어

- 64개 유닛

두 번째 레이어

- 32개 유닛, 단일 벡터 생성

Dense 레이어

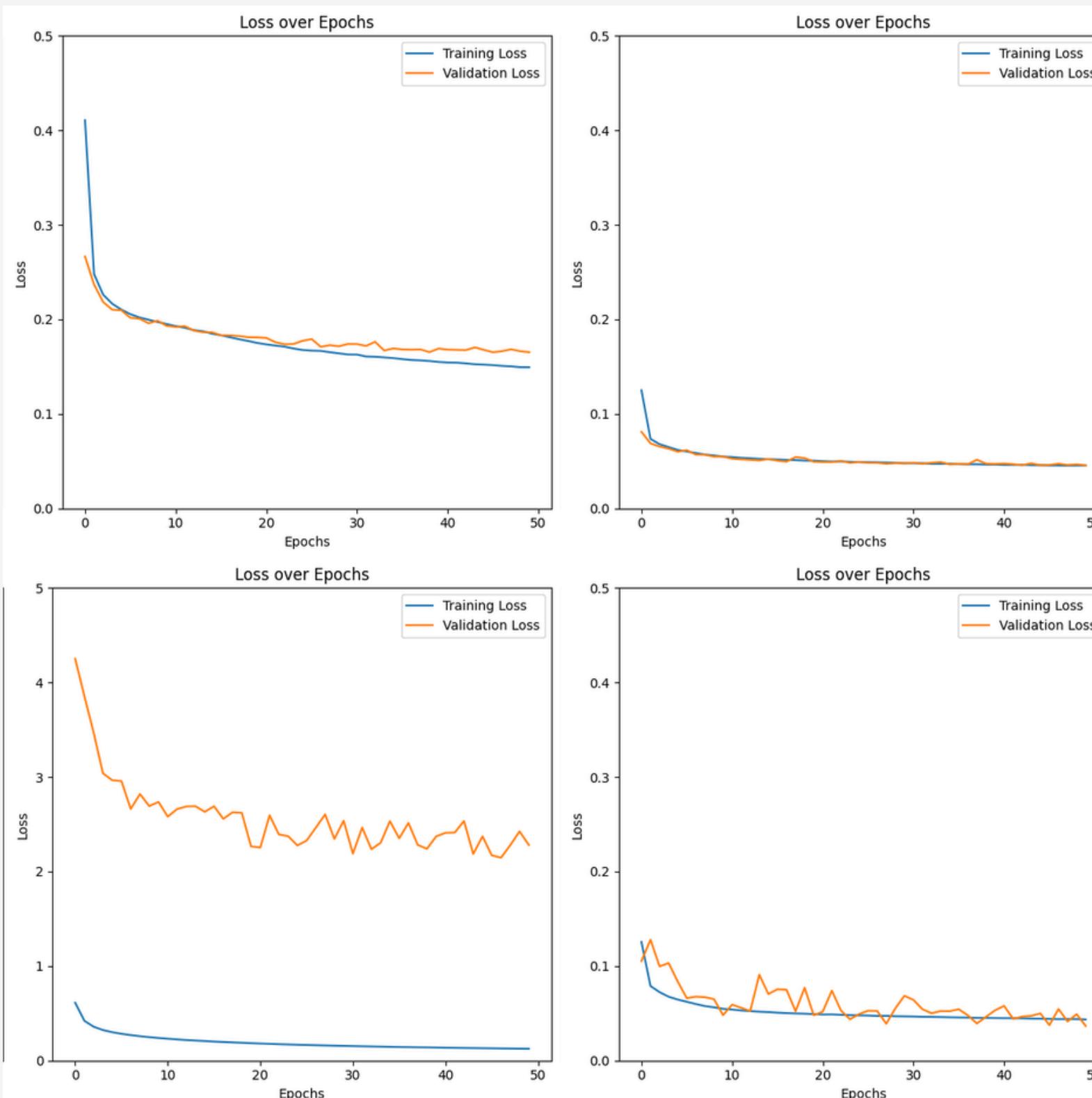
- 클래스 수만큼 출력 노드 생성
- softmax 활성화 함수 사용

컴파일 및 훈련

- 옵티마이저 adam(적응형 학습률)
- 에포크 50
- 배치 64

딥러닝 - 다중 분류

LSTM



V2(특성 추출)

- 훈련 손실과 검증 손실이 모두 낮은 수준으로 안정적으로 감소하며, 최종적으로 약 0.05~0.1에서 수렴.
- 훈련 데이터와 검증 데이터 간 손실 차이가 적어 과적합이 나타나지 않음.

V3(SMOTE)

검증 손실의 변동 폭이 크고, 훈련 손실과의 차이가 커 과적합이 심각하게 발생.

V4(특성 추출 + SMOTE)

- 훈련 손실과 검증 손실이 모두 낮은 값에서 시작하며, 최종적으로 약 0.05~0.1에서 수렴.
- 검증 손실이 초기 학습 단계에서 변동이 크지만, 에포크가 진행됨에 따라 점차 안정화.

딥러닝 - 다중 분류

GRU

```
# 데이터 3D로 변환
X_train_binary_gru = X_train_binary_balanced.values.reshape(X_train_binary_balanced.shape[0], 1, X_train_binary_balanced.shape[1])
X_test_binary_gru = X_test_binary_selected.values.reshape(X_test_binary_selected.shape[0], 1, X_test_binary_selected.shape[1])

# GRU 모델 정의
start_train = time.time()
gru_binary = Sequential([
    GRU(64, input_shape=(1, X_train_binary_balanced.shape[1]), return_sequences=True),
    GRU(32),
    Dense(1, activation='sigmoid')
])

# 모델 컴파일
gru_binary.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# 훈련
gru_binary.fit(X_train_binary_gru, y_train_binary_balanced, epochs=50, batch_size=64, verbose=1)

end_train = time.time()
```

데이터 3D로 변환

- 입력 데이터는 3D 텐서 형태를 필요
- 데이터를 (샘플 수, 시간 스텝, 특성 수)로

모델 정의

첫 번째 레이어

- 64개 유닛

두 번째 레이어

- 32개 유닛, 단일 벡터 생성

Dense 레이어

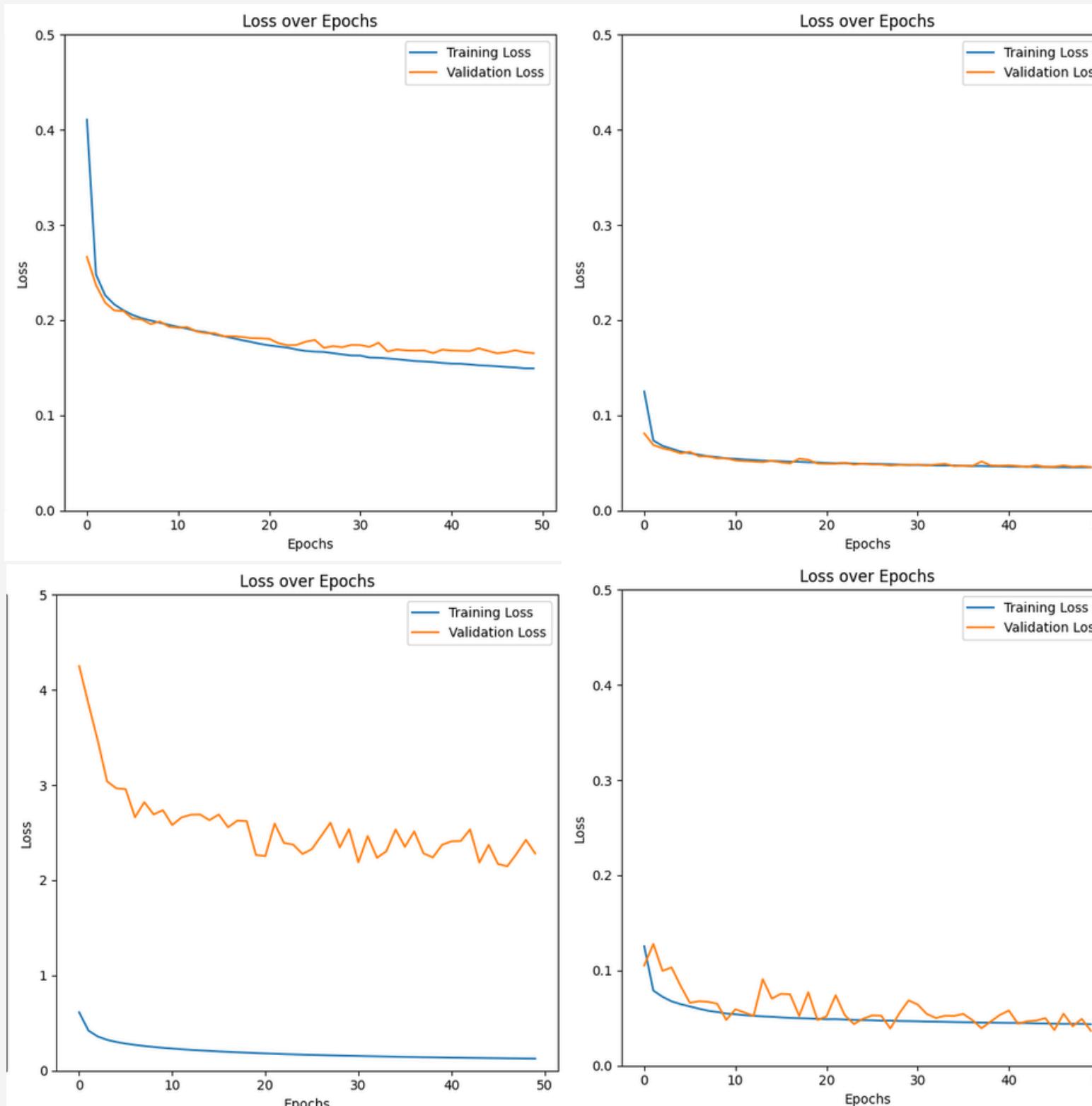
- 클래스 수만큼 출력 노드 생성
- softmax 활성화 함수 사용

컴파일 및 훈련

- 옵티마이저 adam(적응형 학습률)
- 에포크 50
- 배치 64

딥러닝 - 다중 분류

GRU



V2(특성 추출)

- 훈련 손실과 검증 손실이 모두 낮은 수준으로 안정적으로 감소하며, 최종적으로 약 0.05~0.1에서 수렴.
- 훈련 데이터와 검증 데이터 간 손실 차이가 적어 과적합이 나타나지 않음.

V3(SMOTE)

검증 손실의 변동 폭이 크고, 훈련 손실과의 차이가 커 과적합이 심각하게 발생.

V4(특성 추출 + SMOTE)

- 훈련 손실과 검증 손실이 모두 낮은 값에서 시작하며, 최종적으로 약 0.05~0.1에서 수렴.
- 검증 손실이 초기 학습 단계에서 변동이 크지만, 에포크가 진행됨에 따라 점차 안정화.

딥러닝 - 다중 분류

CNN

```
# 1. 데이터 준비  
# 2D 데이터 형태로 변환 (예: [batch_size, 1, height, width])  
height = 8  
width = X_train_multi.shape[1] // height  
  
X_train_cnn = X_train_multi.values.reshape(-1, 1, height, width).astype('float32')  
X_test_cnn = X_test_multi.values.reshape(-1, 1, height, width).astype('float32')  
  
# PyTorch 텐서로 변환  
X_train_tensor = torch.tensor(X_train_cnn, dtype=torch.float32)  
y_train_tensor = torch.tensor(y_train_multi.values, dtype=torch.long)  
X_test_tensor = torch.tensor(X_test_cnn, dtype=torch.float32)  
y_test_tensor = torch.tensor(y_test_multi.values, dtype=torch.long)  
  
# DataLoader 생성  
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)  
test_dataset = TensorDataset(X_test_tensor, y_test_tensor)  
  
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)  
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

```
# 2. CNN 모델 정의  
class CNNMultiClassifier(nn.Module):  
    def __init__(self, num_classes):  
        super(CNNMultiClassifier, self).__init__()  
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)  
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)  
        self.pool = nn.MaxPool2d(2, 2)  
        self.fc1 = None  
        self.fc2 = None  
  
    def forward(self, x):  
        x = self.pool(torch.relu(self.conv1(x)))  
        x = self.pool(torch.relu(self.conv2(x)))  
        if self.fc1 is None:  
            self.flatten_size = x.shape[1] * x.shape[2] * x.shape[3]  
            self.fc1 = nn.Linear(self.flatten_size, 128).to(x.device)  
            self.fc2 = nn.Linear(128, num_classes).to(x.device)  
        x = x.view(-1, self.flatten_size)  
        x = torch.relu(self.fc1(x))  
        x = self.fc2(x)  
        return x
```

- 데이터를 CNN이 처리할 수 있는 2D 이미지 형태로 변환
- PyTorch의 DataLoader를 사용해 학습용 데이터와 테스트 데이터를 배치 단위로 분리

- Conv2D: 데이터를 필터로 처리 후 특징 탐색
- MaxPooling: 데이터 크기를 줄이고 계산 효율 향상
- Fully Connected Layer: 최종적으로 클래스 분류.

딥러닝 - 다중 분류

CNN

```
# 4. 학습 루프
epochs = 20
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

start_train = time.time()
for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    correct_train = 0
    total_train = 0

    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        optimizer.zero_grad()
        outputs = model(X_batch)
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        correct_train += (predicted == y_batch).sum().item()
        total_train += y_batch.size(0)

    train_losses.append(running_loss / len(train_loader))
    val_losses.append(running_loss / len(val_loader))

    # 평가 및 출력
    if epoch % 5 == 0:
        print(f'Epoch {epoch+1}/{epochs} Train Loss: {running_loss / len(train_loader)}')

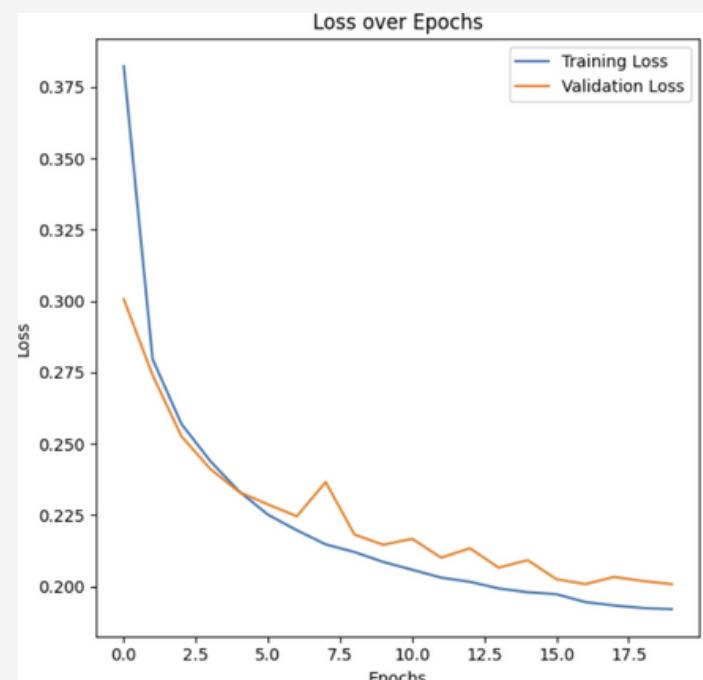
# 학습 끝나면 모델 저장
torch.save(model.state_dict(), 'model.pth')
```

1. 입력 데이터를 모델에 전달
2. 예측 값과 실제 값의 차이를 계산(손실 계산)
3. 손실을 기반으로 모델 가중치를 업데이트

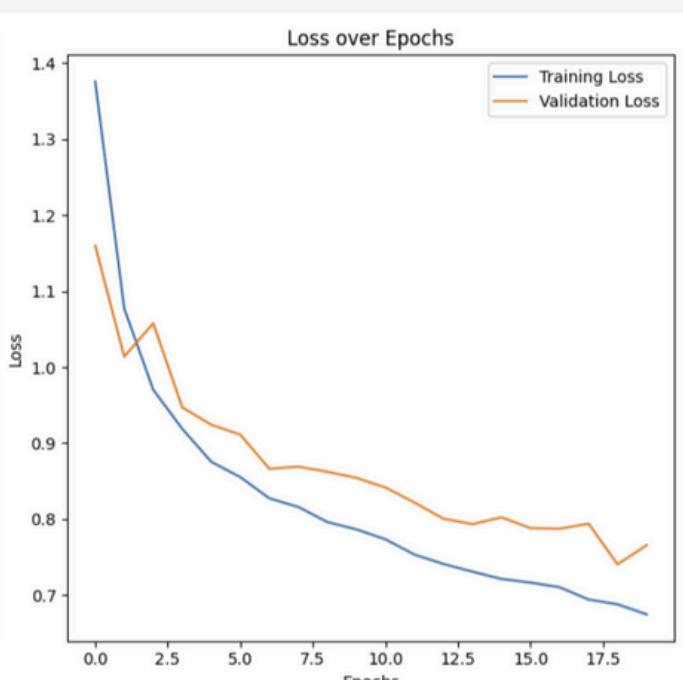
→ 이 과정을 20번 반복(Epoch)하면서 학습

딥러닝 - 다중 분류

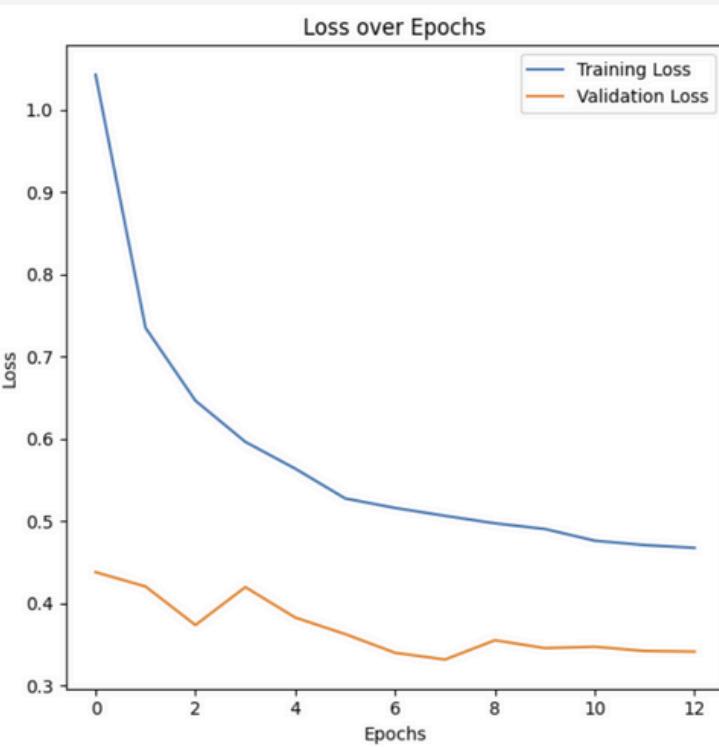
CNN



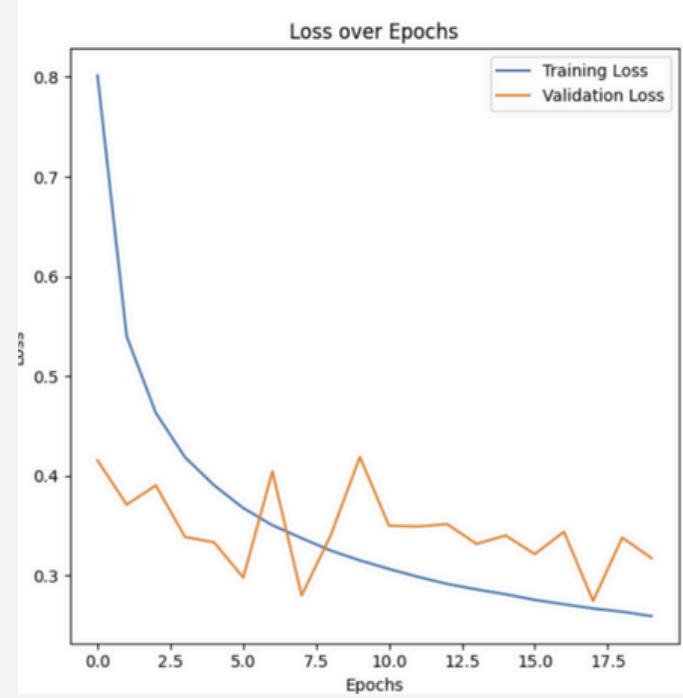
V1 전처리만



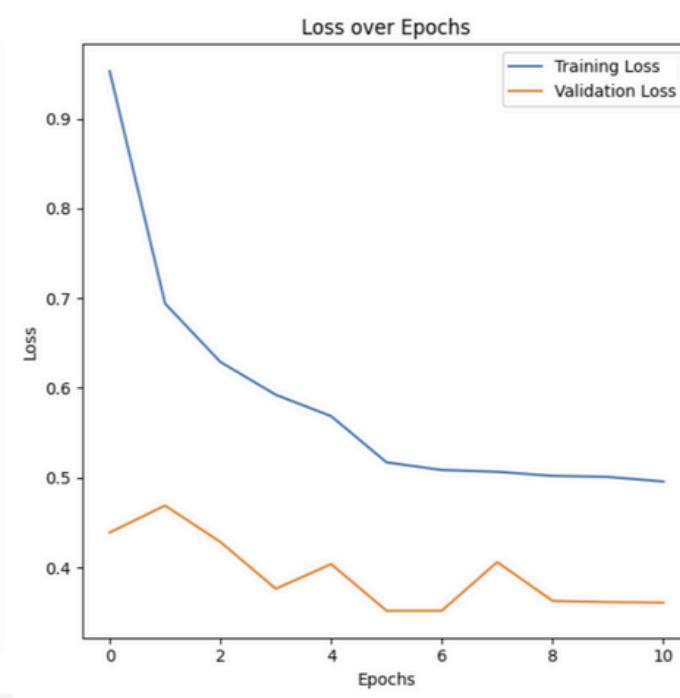
V2 클래스 가중치



V5 SMOTE+Random Search



V3 SMOTE



V4 Random Search

클래스 가중치

- Training loss와 Validation loss가 모두 불안정

SMOTE

- 소수 클래스들의 성능이 전반적으로 향상됨
- Training loss가 안정적으로 감소하며, Validation loss가 다소 불안정하지만, 학습이 진행됨에 따라 수렴하는 경향

하이퍼파라미터 튜닝

- Training loss와 Validation loss가 더 안정적인 감소 패턴
- 두 곡선 간의 간격이 적절히 유지되며 수렴

SMOTE + 하이퍼파라미터 튜닝

- 두 곡선 모두 매우 안정적으로 감소

성능 평가

Model Performance Summary									
	Model	Type	Accuracy	Recall	Precision	F1-Score	Time to Train	Time to Predict	Total Time
0	Random Forest	Binary	98.6120%	0.989053	0.992409	0.990728	30.89	0.331052	31.22
1	Random Forest	Multi-class	93.6210%	nan	nan	0.939169	302.54	0.334587	302.88
2	Extra Trees	Binary	98.5304%	0.988996	0.991382	0.990188	13.27	0.514757	13.78
3	Extra Trees	Multi-class	93.2772%	nan	nan	0.935549	76.66	0.453404	77.11
4	LSTM	Binary	97.8213%	nan	nan	nan	645.68	1.885014	647.57
5	LSTM	Multi-class	91.1509%	nan	nan	nan	1755.82	2.685532	1758.51
6	GRU	Binary	97.8729%	nan	nan	nan	755.38	1.539457	756.92
7	GRU	Multi-class	90.4917%	nan	nan	nan	1769.36	1.359433	1770.72

V2 전처리 + 특성추출

Model Performance Summary									
	Model	Type	Accuracy	Recall	Precision	F1-Score	Time to Train	Time to Predict	Total Time
0	Random Forest	Binary	98.5475%	0.988423	0.992176	0.990296	21.81	0.172932	21.98
1	Random Forest	Multi-class	93.1625%	nan	nan	0.935794	281.66	0.415768	282.07
2	Extra Trees	Binary	98.4702%	0.988308	0.991262	0.989783	11.10	0.265296	11.36
3	Extra Trees	Multi-class	92.8359%	nan	nan	0.932458	63.67	0.563854	64.23
4	LSTM	Binary	97.8987%	nan	nan	nan	636.73	2.294092	639.03
5	LSTM	Multi-class	89.5060%	nan	nan	nan	1833.72	1.749470	1835.47
6	GRU	Binary	97.8041%	nan	nan	nan	770.64	3.137188	773.77
7	GRU	Multi-class	89.7524%	nan	nan	nan	1789.59	1.942874	1791.54

V4 전처리 + 특성추출 + SMOTE

데이터에 따른 최적의 모델

간단한 데이터셋:

- Binary: Random Forest(v4)
- Multi-class: Random Forest(v2)

복잡한 데이터셋:

- Binary: LSTM/GRU(v2)
- Multi-class: LSTM/GRU(v2)

시간 효율성 중시:

- Binary: Extra Trees(v2)
- Multi-class: Extra Trees(v2)

최대 성능 추구:

- Binary: Random Forest(v4)
- Multi-class: Random Forest(v2)

성능 평가

Updated Model Performance Summary									
	Model	Type	Accuracy	Recall	Precision	F1-Score	Time to Train	Time to Predict	Total Time
0	Decision Tree	Binary	98.08%	0.987	0.987	0.987	2.21	0.008734	2.22
1	Decision Tree	Multi-class	91.61%	0.916	0.928	0.921	31.72	0.007798	31.72
2	XGBoost	Binary	98.50%	0.987	0.993	0.990	6.15	0.061483	6.21
3	XGBoost	Multi-class	93.09%	0.931	0.941	0.934	69.90	0.287441	70.18
4	CNN	Multi-class2	86.64%	nan	nan	nan	230.20	0.955592	231.15
5	CNN	Multi-class3	89.62%	nan	nan	nan	1032.94	0.925785	1033.86

V3 SMOTE

Updated Model Performance Summary									
	Model	Type	Accuracy	Recall	Precision	F1-Score	Time to Train	Time to Predict	Total Time
0	Decision Tree	Binary	98.11%	0.984	0.991	0.987	39.69	0.016435	39.71
1	Decision Tree	Multi-class	94.07%	0.941	0.938	0.938	32.40	0.008139	32.41
2	XGBoost	Binary	98.52%	0.990	0.990	0.990	156.59	0.169607	156.76
3	XGBoost	Multi-class	94.54%	0.945	0.943	0.941	955.39	0.313047	955.70
4	CNN	Multi-class4	86.82%	nan	nan	nan	703.40	1.000600	704.40
5	CNN	Multi-class5	87.40%	nan	nan	nan	720.37	1.581217	721.95

V5 하이퍼파라미터 튜닝

데이터에 따른 최적의 모델

간단한 데이터셋

- Binary: Decision Tree (v3)
- Multi-class: Decision Tree (v3)

복잡한 데이터셋:

- Binary: XGBoost (v5)
- Multi-class: XGBoost (v5)

시간 효율성 중시:

- Binary: Decision Tree (v3)
- Multi-class: Decision Tree (v3)

최대 성능 추구:

- Binary: XGBoost (v5)
- Multi-class: XGBoost (v5)

결론

1

다양한 모델 활용

Decision Tree, Random Forest, XGBoost, LSTM, GRU, CNN 등 다양한 모델을 적용하여 최적의 성능을 탐색

2

케이스 기반 테스트

각 모델마다 4~5개의 케이스로 나누어 데이터셋에 맞춘 최적화 접근

3

데이터셋 특성 반영

SMOTE를 활용한 클래스 불균형 해결로 데이터셋 특성을 최대한 반영

결론

침입탐지시스템의 핵심 요소를 중심으로 평가하여 상황별 최적의 모델 도출

요소	적합 모델	설명	향후 과제
실시간 탐지 능력	RF, DT	예측 속도가 매우 빠르고 실시간 탐지에 적합	딥러닝 모델(LSTM, GRU)의 학습 및 예측 시간 최적화 필요
정확성과 오탐률	XGBoost, RF	SMOTE 및 하이퍼파라미터 튜닝으로 높은 정확도와 낮은 오탐률	오탐률을 더욱 줄이기 위해 데이터 전처리 및 분류기 튜닝 방식 고도화 필요
적응성과 학습 능력	XGBoost	하이퍼파라미터 튜닝으로 학습 성능 강화	딥러닝 모델의 데이터 업데이트 자동화 및 학습 속도 최적화 필요
확장성	RF, ET	학습 및 예측 시간이 짧고 병렬 처리 지원으로 대규모 네트워크 환경에 적합	딥러닝 모델(CNN, LSTM, GRU)의 학습 시간 개선 및 대규모 데이터 처리 최적화 필요
다양한 공격 유형 탐지	XGBoost, CNN	XGBoost는 이진 및 다중 클래스 분류에서 뛰어난 성능을 보임 CNN은 공간적 패턴 학습에 강점	CNN의 학습 속도 및 효율성을 개선해 복잡한 공격 유형 탐지 성능 강화

감사합니다.

202211319 박정은

2022***** 이성민

| 참고 자료

데이터셋 URL

https://unsw-my.sharepoint.com/personal/z5025758_ad_unsw_edu_au/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fz5025758_ad_unsw_edu_au%2FDocuments%2FUNSW-NB15%20dataset&ga=1

참고 논문

- **The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set**
 - https://www.researchgate.net/publication/290061926_The_evaluation_of_Network_Anomaly_Detection_Systems_Statistical_analysis_of_the_UNSW-NB15_data_set_and_the_comparison_with_the_KDD99_data_set
- **특징선택 기법에 기반한 UNSW-NB15 데이터셋의 분류 성능 개선**
 - <https://koreascience.kr/article/JAKO201915561988445.page>

| 역할 분담

박정은 (Random Forest, Extra Trees, LSTM, GRU)

- 전처리
 - <https://colab.research.google.com/drive/1PmHMny4oHlqlvdOwJRJ8HOYhxiuKNxGq?usp=sharing>
- 전처리 + SMOTE
 - <https://colab.research.google.com/drive/1xV9P77XDELzkrBbiMR6jjOSlAiicGyLS?usp=sharing>
- 전처리 + 특성추출
 - https://colab.research.google.com/drive/1iMx2uI5YIziz8_zbtgRQ2Mbu7uWniekP?usp=sharing
- 전처리 + 특성 추출 + SMOTE
 - https://colab.research.google.com/drive/1Wk9jxG5ECOUmRe-e4YRI9VfCYkf_WIL-?usp=sharing

이성민 (Decision Tree, XGBoost, CNN)

- 머신러닝(Decision Tree, XGBoost)
 -
- CNN
 -
- 성능평가 결과만
 -