기계학습론 IC-PBL 최종 보고서

2019024202 로봇공학과 김민표
2019045296 컴퓨터학과 설혁정
2019041694 로봇공학과 최호재
2018042624 산업경영공학과 김윤성

목차

1.서론

(1) 문제 상황

한 K 로보틱스 회사의 스테이지 로봇 연구 및 개발 팀 팀장으로서, 현재로서 회사들은 로봇 조인트의 주요 구성 요소인 다이나믹셀(Dynamixel)의 상태를 정확하게 진단할 수 있는 기술이 필요하다. 로봇이 무대에 올라가기 전에 로봇 상태를 사전 점검하여 사고를 예방하기 위해 계속해서 다이나믹셀 작동 데이터를 양품과 불량품에 대해 수집하고 있다. 이 데이터를 사용하여 머신러닝과 딥러닝을 통해 인공지능 모델을 개발하고 정확한 성능 분석을 통해 모델을 평가해보자.

Time	Vibration	Freq(Hz)	PowerSpectrum
0.000000	-2.964125e-01	0.000	9.606538e-04
0.002489	-3.731769e-01	0.392	8.126512e-02
0.004978	-6.611931e-02	0.785	4.611946e-02
0.007466	8.740945e-02	1.177	8.637087e-02
0.009955	1.641738e-01	1.570	8.173451e-02
0.012444	2.025560e-01	1.962	3.392458e-02
0.014933	1.641738e-01	2.354	1.413621e-01
0.017421	2.025560e-01	2.747	3.927828e-01
0.019910	-2.580303e-01	3.139	3.839231e-01
0.022399	-2.964125e-01	3.532	7.305730e-02
0.024888	1.064507e-02	3.924	8.947607e-02
0.027376	-1.428837e-01	4.316	7.617711e-02
0.029865	-2.580303e-01	4.709	7.492362e-02
0.032354	3.560848e-01	5.101	3.315541e-03
0.034843	3.560848e-01	5.493	8.622053e-02
0.037331	2.793204e-01	5.886	7.058191e-02
0.039820	8.740945e-02	6.278	4.298978e-02
0.042309	4.902726e-02	6.671	2.734938e-02
0.044798	-1.812659e-01	7.063	2.514008e-02
0.047286	-3.731769e-01	7.455	1.737695e-02
0.049775	-4.115590e-01	7.848	2.874996e-02
0.052264	-1.428837e-01	8.240	3.272058e-02
0.054753	-6.611931e-02	8.633	2.525425e-02
0.057241	1.641738e-01	9.025	2.592831e-02
0.059730	1.641738e-01	9.417	3.568760e-02
0.062219	2.025560e-01	9.810	6.853068e-02

→

	A	B
1	0	-0.02942
2	0.00249	0.1241
3	0.00498	-0.10619
4	0.00747	0.04734
5	0.00995	-0.25972
6	0.01244	-0.14457
7	0.01493	-0.25972
8	0.01742	-0.18295
9	0.01991	0.08572
10	0.0224	-0.33648
11	0.02489	0.20087
12	0.02737	-0.25972
13	0.02986	0.20087
14	0.03235	0.00896
15	0.03484	0.1241
16	0.03733	0.46954
17	0.03982	0.04734
18	0.04231	0.27763
19	0.04479	-0.22133
20	0.04728	0.31602
21	0.04977	-0.18295
22	0.05226	0.08572

제공된 데이터는 양품 모터 40개와 불량품 모터 40개의 시간에 따른 진폭 값을 (1024*2) 크기의 csv 파일로 편집하였다. 이때 양품과 불량품 모두1~30번 데이터를 Train set으로 31~40번 데이터를 Test set으로 설정하여 학습을 진행시킨다.

데이터를 추출하는 방법은 중복의 정도에 따라 총 3가지로 나뉜다.

1024행을 256개의 행으로 분할할 때 중복을 0%,25%,50%로 설정하고 진행한다.

(1) 중복 0% : 1024개의 행이 256 *4 로 총 4개의 데이터 셋으로 분할

(2) 중복 25% : 1024개의 행이 256개씩 나눌 때 25% 즉 64개씩 겹치게 추출하여 총 5개의 데이터 셋으로 분할

(3)) 중복 25% : 1024개의 행이 256개씩 나눌 때 50% 즉 128개씩 겹치게 추출하여 총 7개의 데이터 셋으로 분할

(2) 데이터 전처리 – jupyter notebook환경에서 진행

```
import glob
import pandas as pd
import numpy as np

file_paths = [r"C:\Users\호재\Desktop\대학\3학년 2학기\기계학습론\data\IC_PBL\abnormal\train",\
              r"C:\Users\호재\Desktop\대학\3학년 2학기\기계학습론\data\IC_PBL\abnormal\test",\
              r"C:\Users\호재\Desktop\대학\3학년 2학기\기계학습론\data\IC_PBL\normal\train",\
              r"C:\Users\호재\Desktop\대학\3학년 2학기\기계학습론\data\IC_PBL\normal\test"]

dataframes_train_abn = []
dataframes_test_abn  = []
dataframes_train_n   = []
dataframes_test_n    = []

df_list = [dataframes_train_abn, dataframes_test_abn, dataframes_train_n, dataframes_test_n]

for i in range(len(file_paths)) :
    files = glob.glob(file_paths[i] + "\\**\\*.csv", recursive=True)

    for file_path in files:
        df = pd.read_csv(file_path, encoding='cp949', header=None)
        df_list[i].append(df)
```

glob 라이브러리를 이용해 폴더에 있는 모든 csv 파일 업로드

```
dataframes_train_abn_0 = []
dataframes_test_abn_0 = []
dataframes_train_n_0  = []
dataframes_test_n_0 = []

df_list_0 = [dataframes_train_abn_0, dataframes_test_abn_0, dataframes_train_n_0, dataframes_test_n_0]

for j in range(len(df_list)):
    df_list_0[j].extend([df[i:i+256] for df in df_list[j] for i in range(0, df.shape[0], 256)])

dataframes_train_abn_25 = []
dataframes_test_abn_25 = []
dataframes_train_n_25  = []
dataframes_test_n_25 = []

df_list_25 = [dataframes_train_abn_25, dataframes_test_abn_25, dataframes_train_n_25, dataframes_test_n_25]

for j in range(len(df_list)):
    df_list_25[j].extend([df[i:i+256] for df in df_list[j] for i in range(0, df.shape[0]-64, 256-64)])

dataframes_train_abn_50 = []
dataframes_test_abn_50 = []
dataframes_train_n_50  = []
dataframes_test_n_50 = []

df_list_50 = [dataframes_train_abn_50, dataframes_test_abn_50, dataframes_train_n_50, dataframes_test_n_50]

for j in range(len(df_list)):
    df_list_50[j].extend([df[i:i+256] for df in df_list[j] for i in range(0, df.shape[0]-128, 256-128)])
```

하나의 csv 파일을 각각 중복 정도에 맞춰 분할

a. 머신러닝

```python
from scipy.stats import skew
from scipy.stats import kurtosis

def abs_mean(x):
    return np.mean(np.abs(x))
def abs_std(x) :
    return  np.std(np.abs(x))

def calculate_outlier_count(df):
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1
    outliers = ((df < (Q1 - 1.0 * IQR)) | (df > (Q3 + 1.0 * IQR))).sum()
    return outliers


def m_average(df) :
    window_size = 4
    moving_average = df.rolling(window=window_size).mean()
    moving_average = moving_average.dropna(axis = 0)

    return(abs_mean(moving_average))

def abs_median(x) :
    return np.median(np.abs(x))
```

```python
# 함수 정의
def calculate_summary_statistics(df_list):

    summary_data = []
    for df in df_list:

        summary_data.append({
            'min': df.min().tolist(),
            'max': df.max().tolist(),
            'mean': df.mean().tolist(),
            'abs_mean' : abs_mean(df).tolist(),
            'std': df.std().tolist(),
            'abs_std' : abs_std(df).tolist(),
            'median': df.median().tolist(),
            'abs_median': abs_median(df).tolist(),
            'skew': df.skew().tolist(),
            'kurt': df.kurtosis().tolist(),
            'outlier': calculate_outlier_count(df).tolist(),
            'M_average': m_average(df).tolist(),

    })

    return summary_data
```

256개로 분할한 데이터의 여러 기술통계량을 추출하여 데이터프레임화

```python
def process_df_list(df_list):
    result_list = []

    for i, df in enumerate(df_list):
        result_df = calculate_summary_statistics(df)
        result_df = pd.DataFrame(result_df)

        if i < 2 :
            result_df['target'] = 1  # 1: 불량
        else:
            result_df['target'] = 0

        result_list.append(result_df)


    train_df = pd.concat([result_list[i] for i in range(len(result_list)) if i % 2 == 0], ignore_index=True)
    test_df = pd.concat([result_list[i] for i in range(len(result_list)) if i % 2 == 1], ignore_index=True)

    return train_df, test_df


train_0, test_0 = process_df_list(df_list_0)

train_25, test_25 = process_df_list(df_list_25)

train_50, test_50 = process_df_list(df_list_50)
```

마지막 양품과 불량품에 대한 변수 'target'를 생성하여 최종 데이터프레임화

(불량품 : target = 1, 양품 : target = 0)

b. 딥러닝

```
def Fourier(df_list) :
    amp_df_list = []
    for df in df_list:
        time = df[0].values
        amplitude = df[1].values

        freq = np.fft.rfftfreq(len(time), np.diff(time)[0])  # 주파수 축 계산
        fft_values = np.fft.rfft(amplitude)  # 푸리에 변환

        # 푸리에 변환 결과에서 진폭값만 추출하고 데이터프레임으로 변환
        amp_df = pd.DataFrame({

            'Amplitude': np.abs(fft_values)
        })


        amp_df_list.append(amp_df)

    return amp_df_list
```

```
for i in range(4) :
    df_list_0[i] = Fourier(df_list_0[i])

for i in range(4) :
    df_list_25[i] = Fourier(df_list_25[i])

for i in range(4) :
    df_list_50[i] = Fourier(df_list_50[i])
```

```
for i in range(4) :
    reform_list = []
    for j in range(len(df_list_50[i])) :
        df = df_list_50[i][j].transpose()
        reform_list.append(df)
    df_list_50[i] = reform_list

train_abn = pd.DataFrame
test_abn = pd.DataFrame
train_n = pd.DataFrame
test_n = pd.DataFrame
a_list = [train_abn, test_abn, train_n, test_n]

for i in range(4) :
    a_list[i]  = pd.concat(df_list_50[i], ignore_index=True)

train_abn = a_list[0]
train_abn['target'] = 1
test_abn = a_list[1]
test_abn['target'] = 1
train_n = a_list[2]
train_n['target'] = 0
test_n = a_list[3]
test_n['target'] = 0

train_50 = pd.concat([train_abn,train_n], ignore_index=True)
test_50 = pd.concat([test_abn,test_n], ignore_index=True)
```

푸리에 변환한 데이터프레임을 Transpose 시켜 129개의 열로 변환

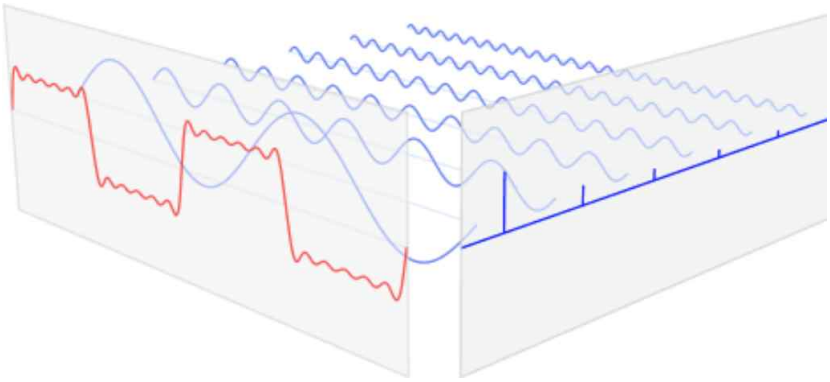
마지막 양품과 불량품에 대한 변수 'target'를 생성하여 최종 데이터프레임화 (불량품 : target = 1, 양품 : target = 0)

최종 결과

DT	DU	DV	DW	DX	DY	DZ
123	124	125	126	127	128	target
1.415163031	4.815775091	1.924190541	1.570017983	1.030901726	1.72716	1
1.226318802	5.266011887	0.658262552	1.02587627	0.884461934	0.49898	1
1.413754169	6.408808415	1.626649999	0.939278501	1.273210841	1.0747	1
1.296622659	5.37354755	1.092680067	1.35063521	0.310274371	0.99792	1
1.413656437	6.408825854	1.626702834	0.939242361	1.273168067	1.07475	1
1.296634207	5.373606872	1.09273905	1.350590035	0.31028305	0.99789	1
1.385211569	4.850059264	0.776114431	1.210818092	0.572094958	0.11515	1
1.925877093	6.096860129	0.68976687	0.579121407	1.183978611	1.45859	1
1.38524076	4.850110563	0.776046272	1.210822118	0.572096606	0.11519	1

(3) 새롭게 사용한 이론

a. FFT(Fast Fourier Transform)



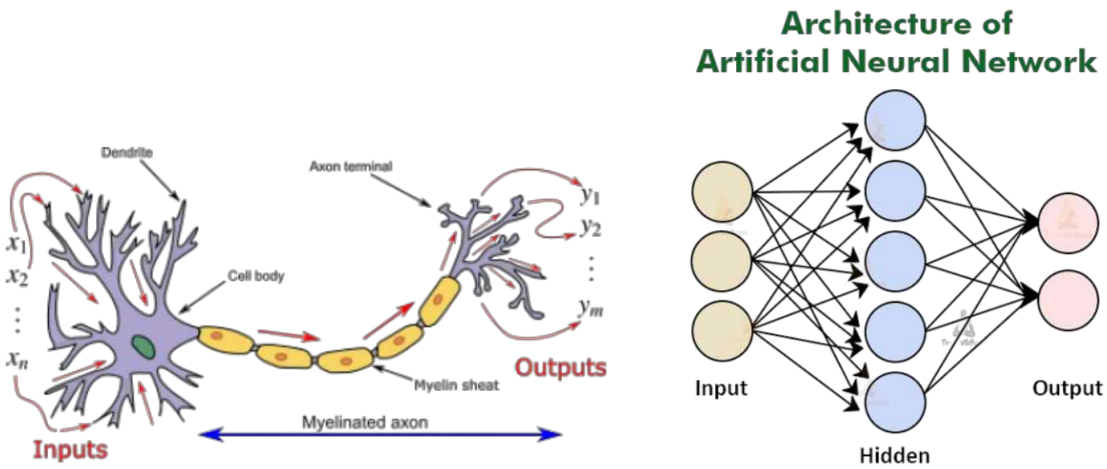
왼쪽부터 실제 시그널, 분해된 sin과 cos함수들, FFT결과

$$
\begin{aligned}
f(t) &= \lim_{T\to\infty} \frac{1}{T} \sum_{m=-\infty}^{\infty} F_m T e^{im\omega_0 t} \\
&= \lim_{T\to\infty} \frac{1}{2\pi} \sum_{m=-\infty}^{\infty} F_m T e^{im\omega_0 t} [(m+1)\omega_0 - m\omega_0] \\
&= \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega
\end{aligned}
$$

FFT의 기본식

FFT(고속 푸리에 변환)는 주어진 시간 도메인의 신호를 주파수 도메인으로 변환하는 효율적인 알고리즘이다. FFT는 시간 도메인에서 샘플링된 신호를 주파수 도메인으로 변환하여 해당 신호의 주요 주파수 성분을 식별할 수 있게 해준다. 일반적인 푸리에 변환보다 계산 속도가 훨씬 빠르다. 모터 상태 판단에서 FFT를 사용하여 모터의 진동 데이터를 수집 후 적용하여 주파수 도메인으로 변환해, 각 주파수 스펙트럼을 효과적으로 분석하여 정상 상태 및 비정상 상태를 구분할 수 있을 것으로 기대된다.

b. ANN(Artificial Neural Network)



ANN의 모델

인공 신경망(ANN)은 생물학적 뉴런의 작동 방식에서 영감을 받은 기계 학습 모델 중 하나이다. 여러 뉴런들이 연결된 계층으로 구성되어 있으며, 각 연결은 가중치와 결합되어 있다. 이 가중치는 입력과의 상대적인 중요성을 나타내며, 학습 과정에서 조절되어 네트워크가 데이터에서 의미 있는 패턴을 학습할 수 있도록 한다. ANN은 입력을 받아 가중치를 곱한 후 활성화 함수를 통과시켜 출력을 생성한다. 이러한 과정은 생물학적 뇌의 뉴런이 전기적 신호를 통해 정보를 전달하는 방식을 모방한 것인데 활성화 함수는 비선형성을 추가하여 네트워크가 복잡한 함수를 근사할 수 있게 한다.

주로 분류, 회귀, 패턴 인식과 같은 다양한 작업에서 ANN이 활용되고 있다. 분류 작업에서는 입력 데이터를 특정 클래스로 분류하고, 회귀 작업에서는 입력에 대한 예측값을 출력한다. 또한, ANN은 다양한 응용 분야에서 효과적으로 활용될 수 있는데, 모터와 같이 비선형적 특성을 가진 데이터 패턴을 학습할 수 있는 능력을 갖추고 있다.

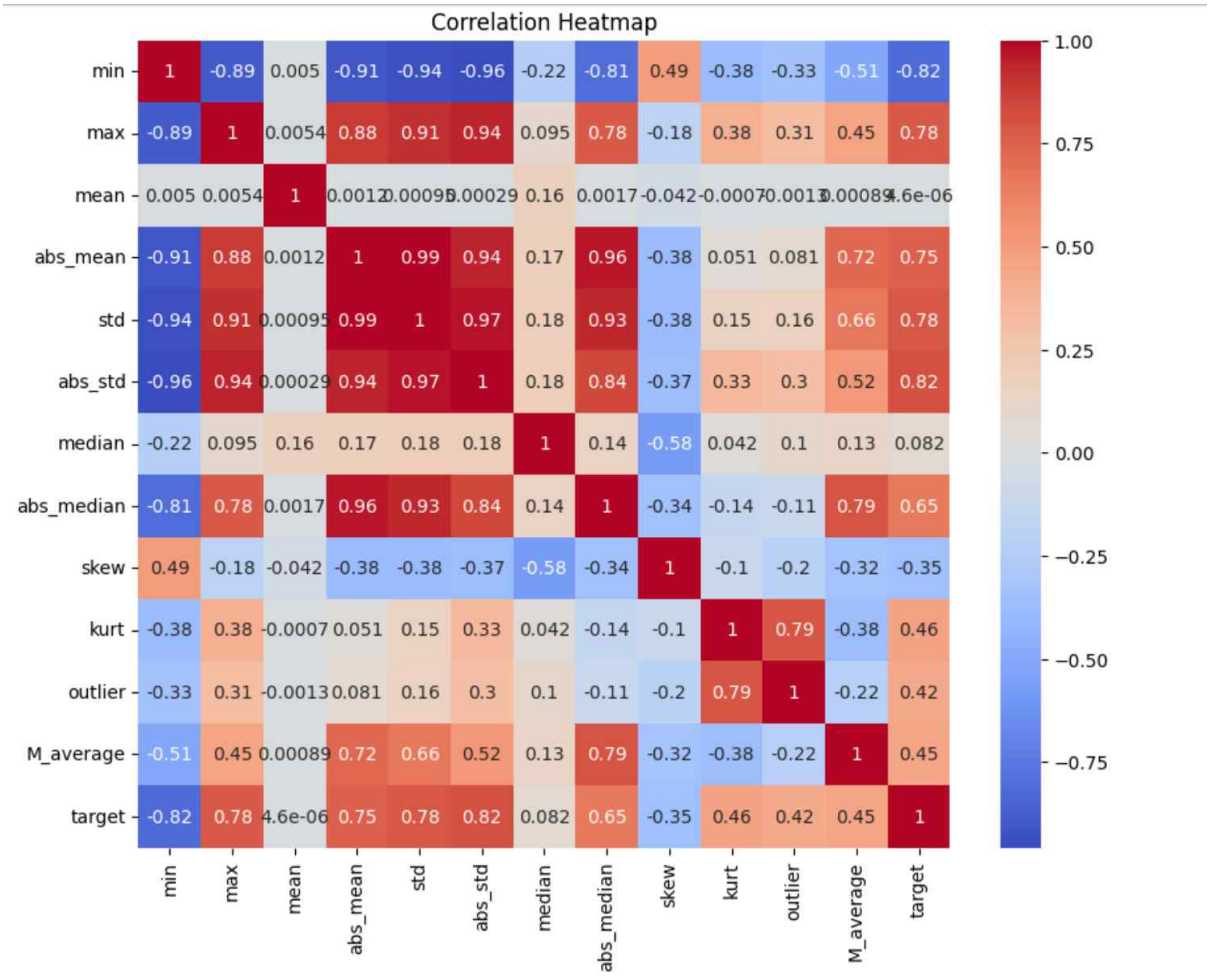
모터의 경우, ANN을 활용하여 진동 데이터를 학습하고 분석함으로써 모터의 상태를 식별할 수 있다. 모터의 정상 및 비정상 상태에 따라 발생하는 진동 데이터의 주파수 도메인 변화를 학습함으로써, 이를 기반으로 모터의 현재 상태를 판단할 수 있다. 이는 기존의 방법보다 효과적으로 모델링하고 모니터링할 수 있는 방법 중 하나라고 생각한다.

2. 제안하는 방법

(1) 통계 변수 데이터 vs fft 변환 변수 데이터를 실험을 통해 비교 분석

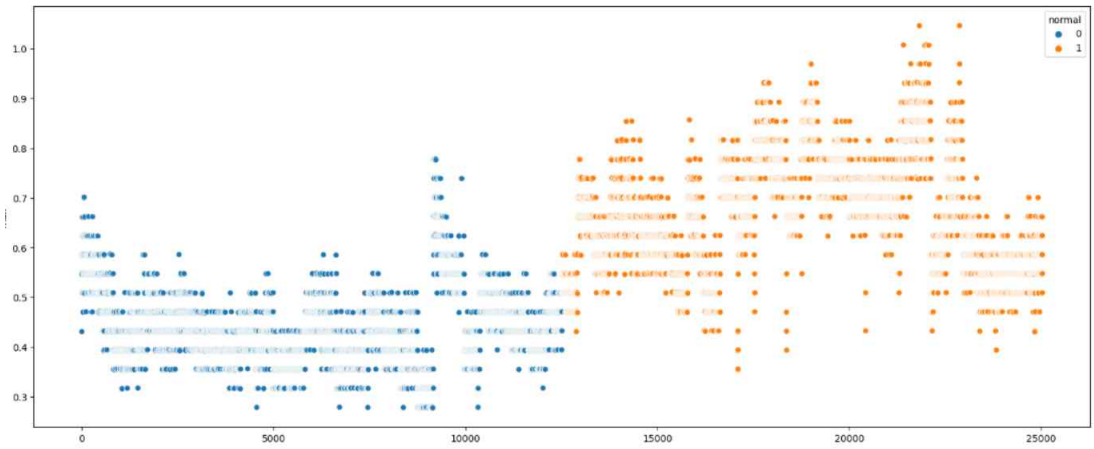
a. 다양한 기술 통계량 선택

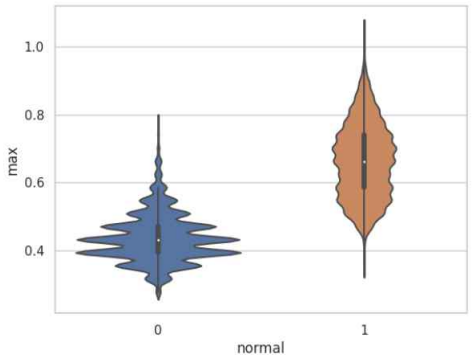
feature이름	feature설명	feature이름	feature설명
min	진폭의 최소 값	std	진폭의 표준편차
max	진폭의 최대 값	abs_std	절대값(진폭)의 표준편차
mean	진폭의 평균	median	진폭의 중앙값
abs_mean	절대값(진폭)의 평균	abs_median	절대값(진폭)의 중앙값
skew	진폭의 왜도	kurt	진폭의 첨도
outlier	진폭 중 이상치 개수	M_average	진폭의 이동평균

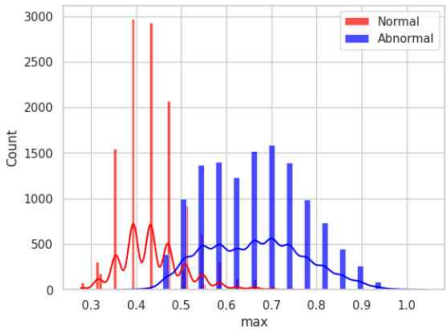


여러 기술 통계량들의 상관 관계

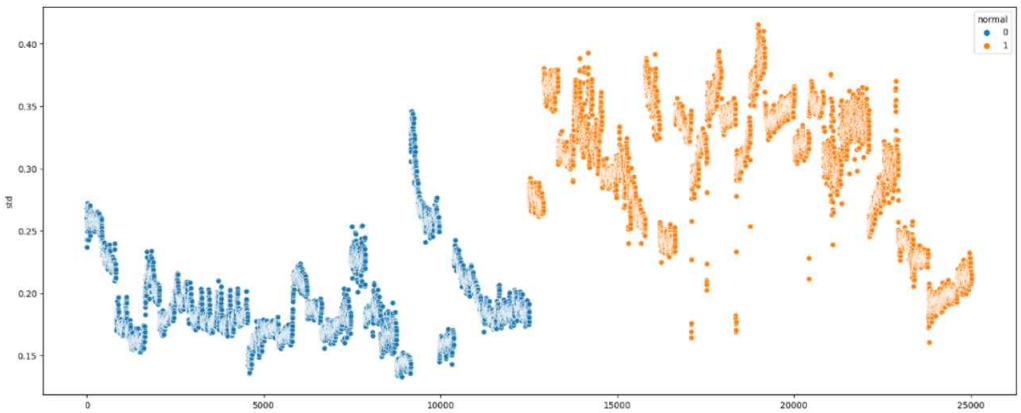
max 변수에 대한 시각화

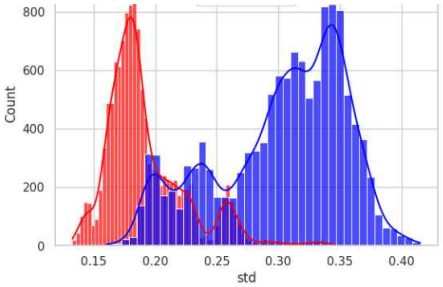


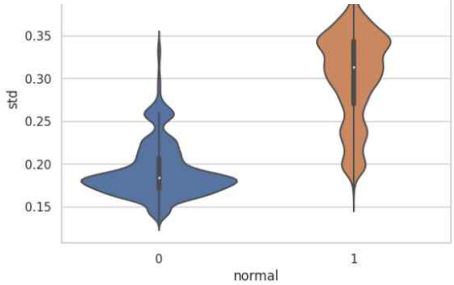




min 변수에 대한 시각화







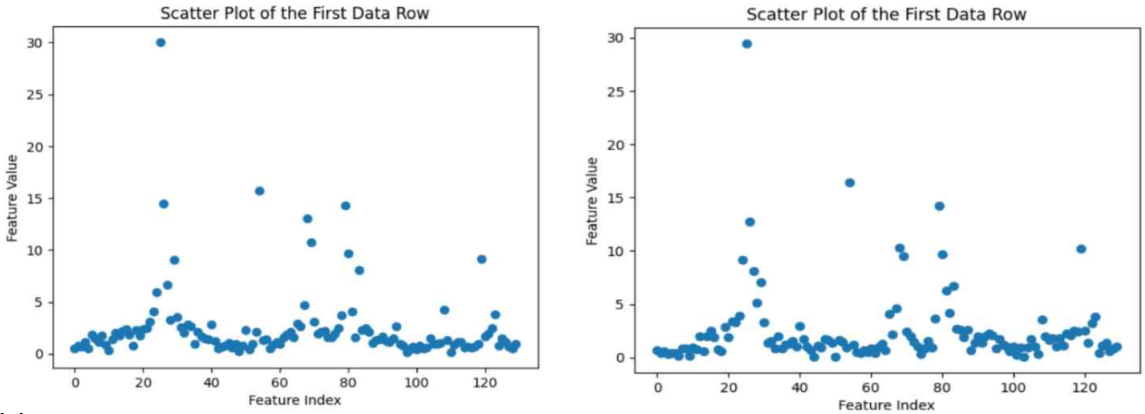
다양한 기술 통계량을 선정하고 이를 feature로

Logistic regression, Random Forest, Decision tree, 딥러닝 모델에 학습

AUC 스코어 : 0.75~ 0.83

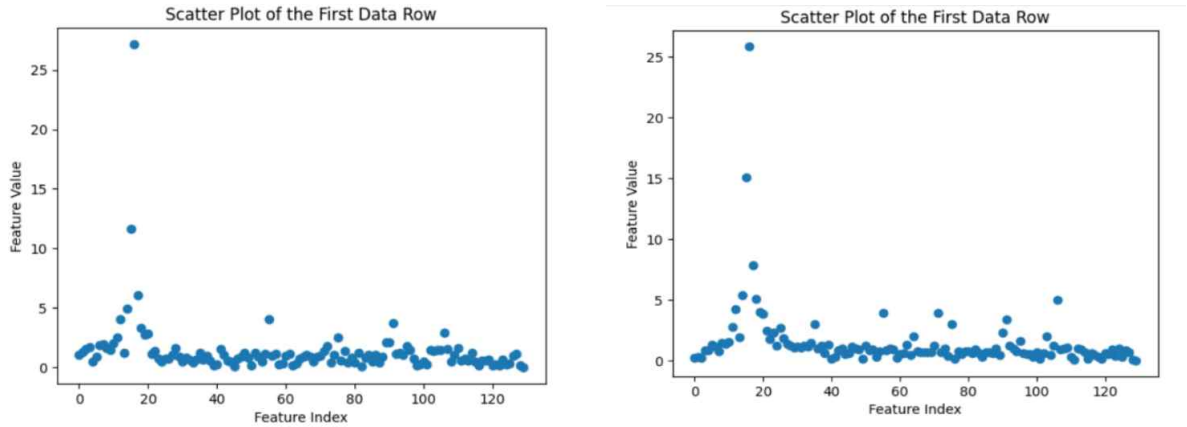
b. FFT 변환

<불량품 모터 FFT 변환 후 시각화>



나.

Logistic regression, Random Forest, Decision tree 모델에 학습



Logistic 모델을 제외한 나머지 모델의

성능은 비슷했지만 standard scaler를 적용한 Logistic regression의 AUC 스코어가 무려 0.98이란 값이 출력

```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)

y_pred = logistic_model.predict(X_test)
y_pred_proba = logistic_model.predict_proba(X_test)


auc = roc_auc_score(y_test, y_pred)
print('AUC Score: ', auc)
```

```
AUC Score:  0.9814345018450185
```

결론:

FFT 변수 데이터로 한 결과가 통계 변수 데이터에 대한 모델 성능 보다 좋았고 시각적으로 특징이 잘 보여 학습시킬 시 긍정적인 결과를 얻을 수 있을 것이라고 예상하여 FFT 변수 데이터를 선택하기로 하였다.

(2) FFT 변수 데이터를 여러 모델에 적용하여 AUC score 비교

로지스틱 모델을 제외하고 나머지 모델들의 AUC 값은 0.75~0.83 범위 내에서 움직이지만 이 중 딥러닝 모델이 AUC score = 0.86~0.92사이 형성

일단 로지스틱 모델의 제외한 나머지 모델 중 가장 성능이 좋은 딥러닝 모델을 결정하였다.

-결론: 사용 모델을 딥러닝으로 결정

(3) 데이터 중복도 선택

a. 로지스틱 회귀

```
Best AUC: 0.9863
Confusion Matrix:
 [[4217  119]
 [   0 4228]]
```

```
Best AUC: 0.9750
Confusion Matrix:
 [[5156  264]
 [   7 5278]]
```

```
Best AUC: 0.9790
Confusion Matrix:
 [[7269  319]
 [   0 7399]]
```

중복도 0% 중복도 25% 중복도 50%

학습 결과 중복도 0%의 데이터가 가장 AUC 스코어가 좋음.

b. 딥러닝

딥러닝도 마찬가지로 중복도가 올라간다고 AUC 스코어가 올라가지 않았다.
중복도 0%의 AUC 스코어가 가장 높았다

결론 -이를 토대로 중복도 0%에 대하여 학습을 진행하여 모델을 분석하였다.

(4) 선택한 딥러닝 모델의 하이퍼 파라미터를 직접 조정하여 최고의 성능을 내는 모델을 도출함.

최적 모델 구조 설명

딥러닝 모델 중 ANN 기반으로 모델을 설계하였다. input layer, hidden layer 3층, output layer 로 구성하였으며 노드 수는 64개부터 디코딩 구조로 추가하였다. 또한 각 hidden layer에 Batch Normalization 레이어와 Dropout 레이어를 추가하여 과적합을 방지하였다. 활성화 함수는 leaky relu를 사용하였으며 마지막 output layer에선 이진 분류 문제이므로 sigmoid 함수를 활용하였다. 최적화 함수는 adam을 사용하였고 이진 분류 문제이므로 binary_crossentropy를 손실 함수로 사용하여 모델을 학습 시켜주었다. 모델의 과적합 방지를 위해 epochs은 30으로 설정해주었고 과적합 여부를 확인하기 위해 validation set을 따로 설정하여 validation loss값도 같이 확인하며 학습을 하였다.

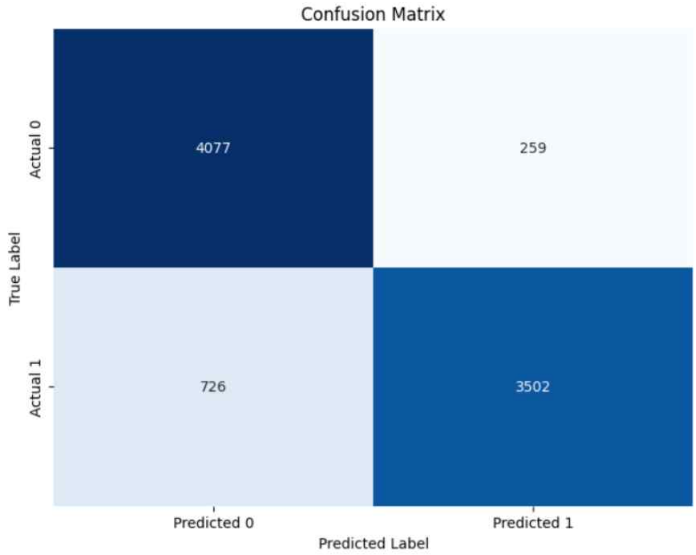
a. 모델1

```
# 신경망 모델 생성
model = Sequential()

# 입력층과 은닉층 추가
model.add(Dense(32, input_dim=129, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Dense(16, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.25))

# 출력층 추가 (이진 분류의 경우 1개의 뉴런과 sigmoid 활성화 함수 사용)
model.add(Dense(1, activation='linear'))

model.summary()
```

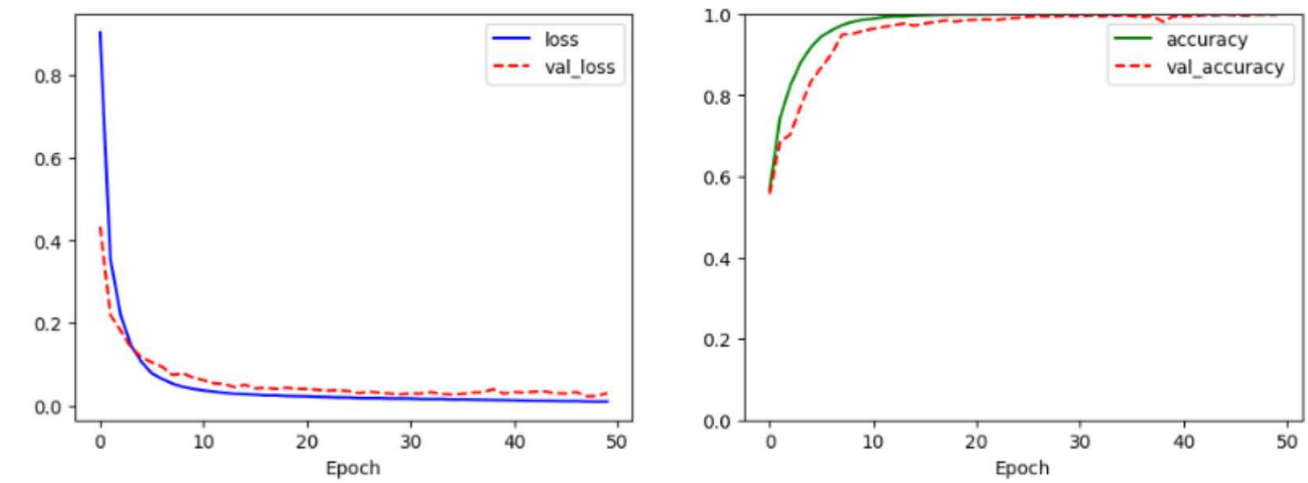
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 32)                4160

 batch_normalization_6 (Bat  (None, 32)                128
 chNormalization)

 dropout_6 (Dropout)         (None, 32)                0

 dense_10 (Dense)            (None, 16)                528

 batch_normalization_7 (Bat  (None, 16)                64
 chNormalization)

 dropout_7 (Dropout)         (None, 16)                0

 dense_11 (Dense)            (None, 1)                 17

=================================================================
Total params: 4897 (19.13 KB)
Trainable params: 4801 (18.75 KB)
Non-trainable params: 96 (384.00 Byte)
_________________________________________________________________
```

모델1의 모델 생성

AUC: 0.8843
정확도: 0.8850
F1 스코어: 0.8767



모델1의 Confusion matrix



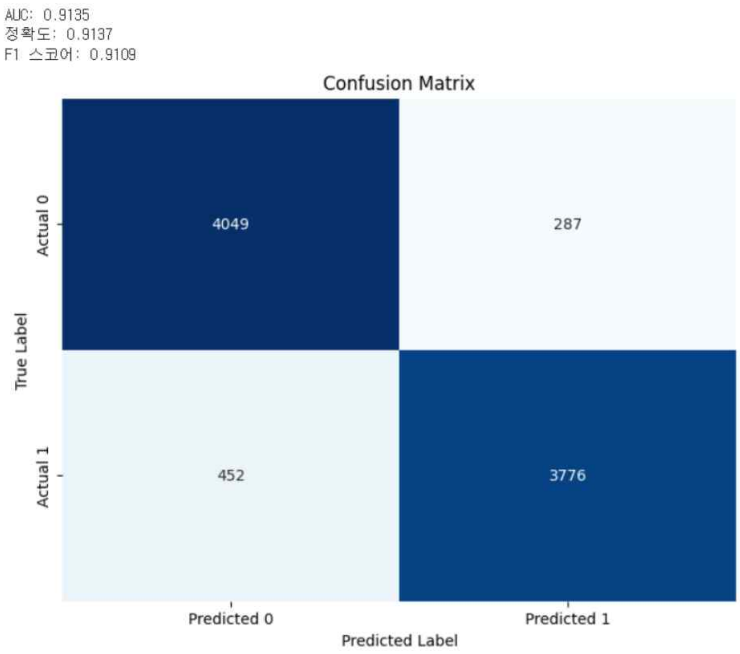
모델1의 loss그래프 및 Accuracy그래프

b. 모델2

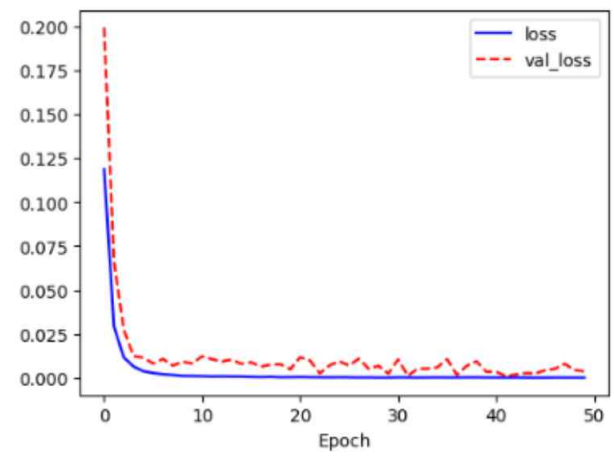
```
# 신경망 모델 생성
model = Sequential()

# 입력층과 은닉층 추가
model.add(Dense(32, input_dim=129, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Dense(16, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.25))

# 출력층 추가 (이진 분류의 경우 1개의 뉴런과 sigmoid 활성화 함수 사용)
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

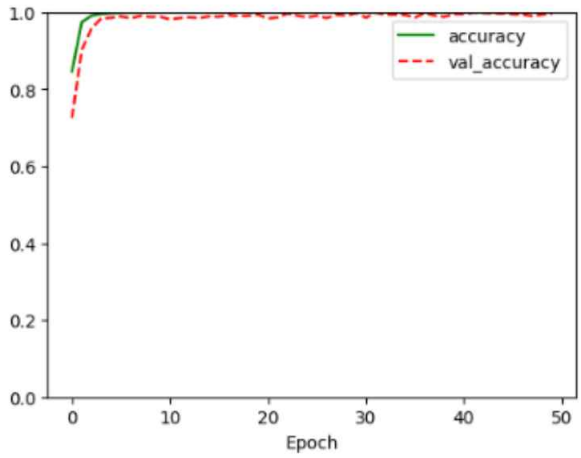
```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 32)                4160

 batch_normalization_8 (Bat  (None, 32)                128
 chNormalization)

 dropout_8 (Dropout)         (None, 32)                0

 dense_13 (Dense)            (None, 16)                528

 batch_normalization_9 (Bat  (None, 16)                64
 chNormalization)

 dropout_9 (Dropout)         (None, 16)                0

 dense_14 (Dense)            (None, 1)                 17

=================================================================
Total params: 4897 (19.13 KB)
Trainable params: 4801 (18.75 KB)
Non-trainable params: 96 (384.00 Byte)
_________________________________________________________________
```

모델2의 모델 생성



모델2의 Confusion matrix





모델2의 loss그래프 및 Accuracy그래프

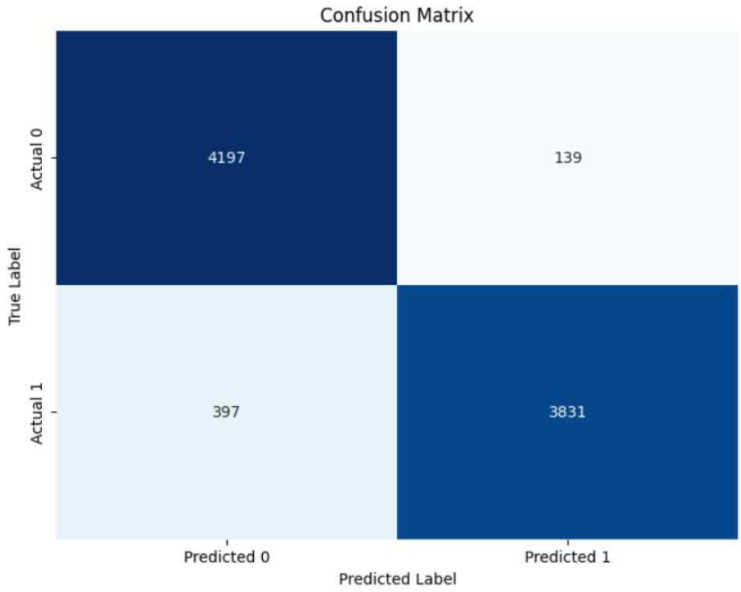
c. 모델3

```
# 신경망 모델 생성
model = Sequential()

# 입력층과 은닉층 추가
model.add(Dense(32, input_dim=129, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Dense(16, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.25))

# 출력층 추가 (이진 분류의 경우 1개의 뉴런과 sigmoid 활성화 함수 사용)
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

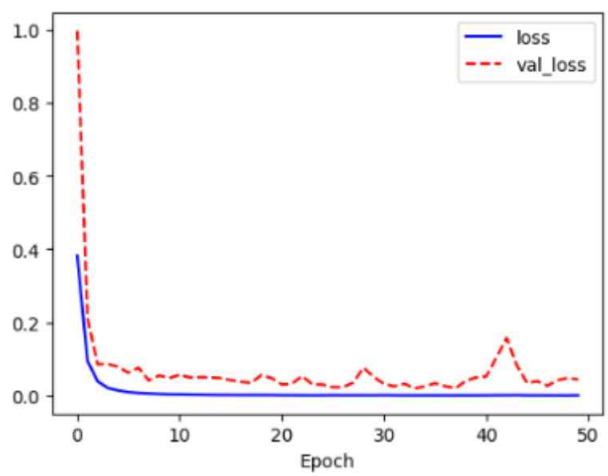
```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 32)                4160

 batch_normalization_10 (Ba  (None, 32)                128
 tchNormalization)

 dropout_10 (Dropout)        (None, 32)                0

 dense_16 (Dense)            (None, 16)                528

 batch_normalization_11 (Ba  (None, 16)                64
 tchNormalization)

 dropout_11 (Dropout)        (None, 16)                0

 dense_17 (Dense)            (None, 1)                 17

=================================================================
Total params: 4897 (19.13 KB)
Trainable params: 4801 (18.75 KB)
Non-trainable params: 96 (384.00 Byte)
_________________________________________________________________
```

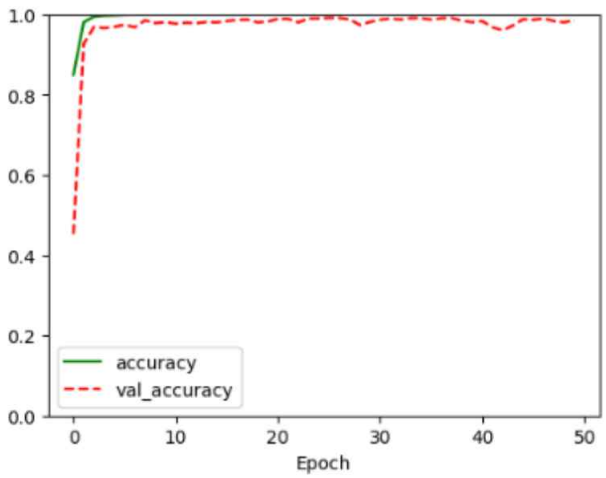
모델3의 모델 생성

```
AUC: 0.9370
정확도: 0.9374
F1 스코어: 0.9346
```



모델3의 Confusion matrix





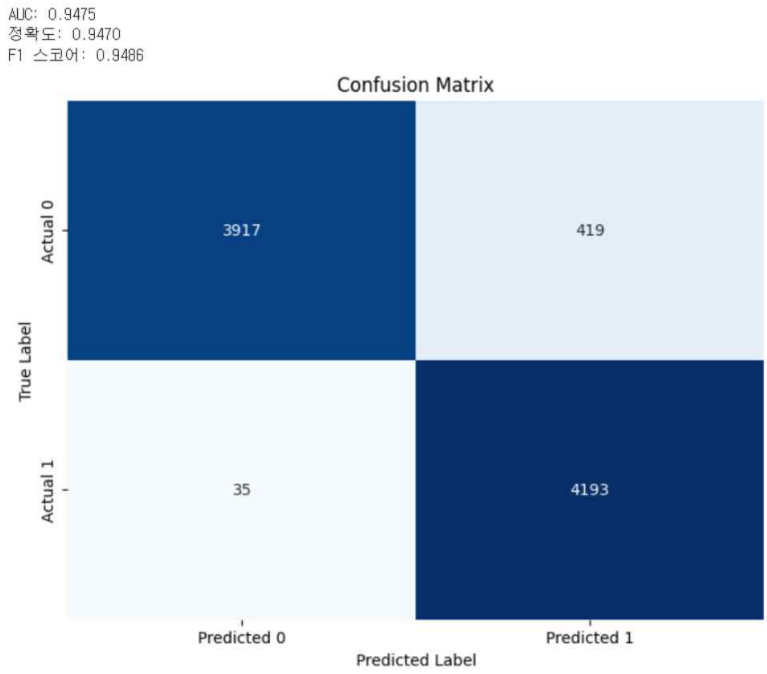
모델3의 loss그래프 및 Accuracy그래프

d. 모델4

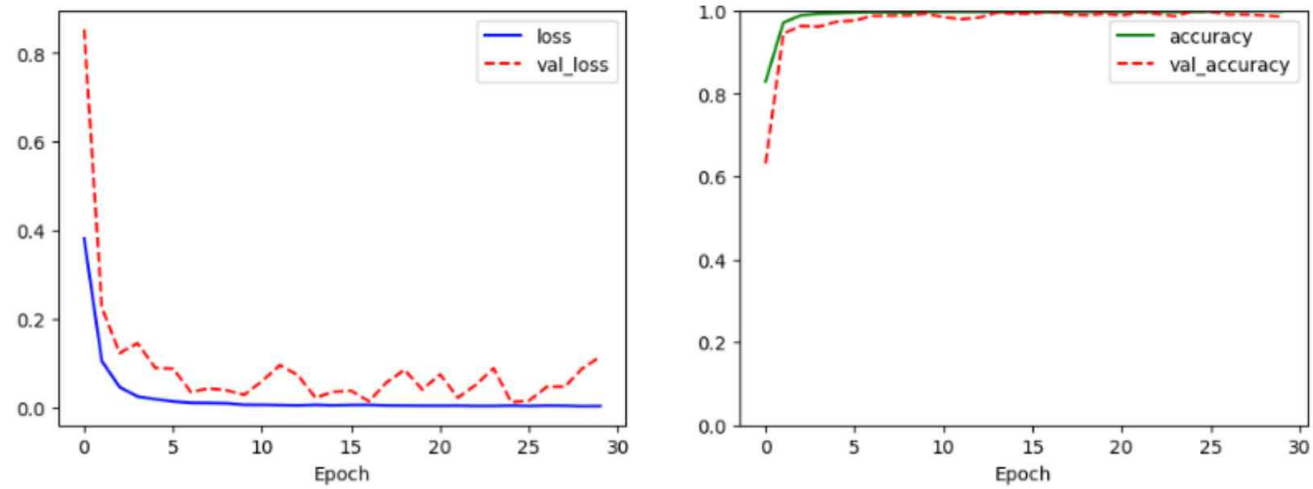
```
# 신경망 모델 생성
model = Sequential()

# 입력층과 은닉층 추가
model.add(Dense(64, input_dim=129, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.5))

# 출력층 추가 (이진 분류의 경우 1개의 뉴런과 sigmoid 활성화 함수 사용)
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_22 (Dense)            (None, 64)                8320

 batch_normalization_14 (Ba  (None, 64)                256
 tchNormalization)

 dropout_15 (Dropout)        (None, 64)                0

 dense_23 (Dense)            (None, 32)                2080

 batch_normalization_15 (Ba  (None, 32)                128
 tchNormalization)

 dropout_16 (Dropout)        (None, 32)                0

 dense_24 (Dense)            (None, 16)                528

 dropout_17 (Dropout)        (None, 16)                0

 dense_25 (Dense)            (None, 1)                 17

=================================================================
Total params: 11329 (44.25 KB)
Trainable params: 11137 (43.50 KB)
Non-trainable params: 192 (768.00 Byte)
_________________________________________________________________
```

모델4의 모델 생성



모델4의 Confusion matrix



모델4의 loss그래프 및 Accuracy그래프

e. 전이학습(Transfer Learning)

전이 학습은 기계 학습에서 미리 학습된 모델을 활용하여 다른 관련 작업에 전이하여 활용하는 기술임. 이는 적은 데이터에서도 효과적이며 성능을 향상시킬 수 있다.

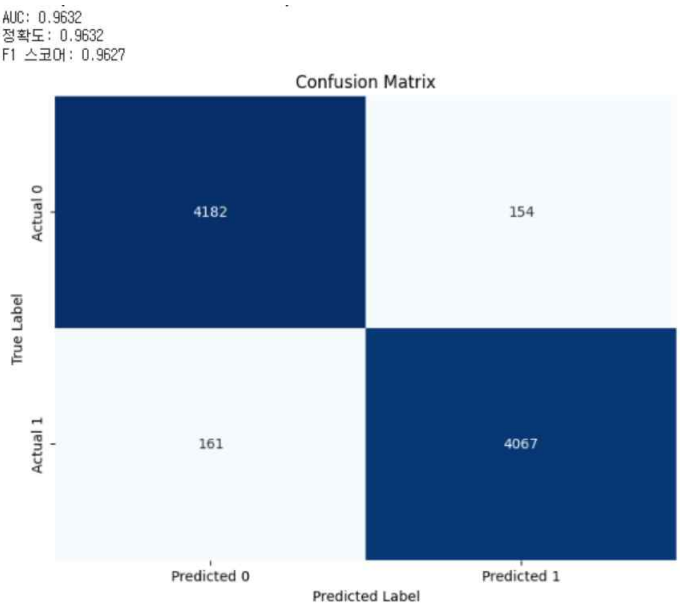
전이학습 이용 후 모델5의 성능 검증

e.1. 모델5(최적)

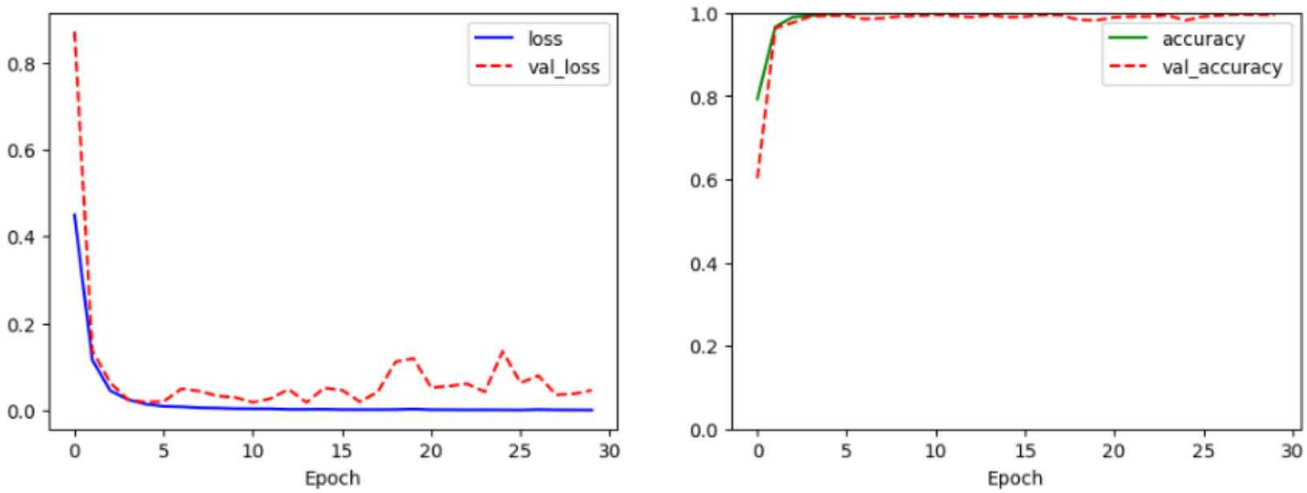
```
# 신경망 모델 생성
model = Sequential()

# 입력층과 은닉층 추가
model.add(Dense(64, input_dim=129, activation='leaky_relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(32, activation='leaky_relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(16, activation='leaky_relu'))
model.add(Dropout(0.5))

# 출력층 추가 (이진 분류의 경우 1개의 뉴런과 sigmoid 활성화 함수 사용)
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_42 (Dense)            (None, 64)                8320

 batch_normalization_25 (Ba  (None, 64)                256
 tchNormalization)

 dropout_28 (Dropout)        (None, 64)                0

 dense_43 (Dense)            (None, 32)                2080

 batch_normalization_26 (Ba  (None, 32)                128
 tchNormalization)

 dropout_29 (Dropout)        (None, 32)                0

 dense_44 (Dense)            (None, 16)                528

 dropout_30 (Dropout)        (None, 16)                0

 dense_45 (Dense)            (None, 1)                 17

=================================================================
Total params: 11329 (44.25 KB)
Trainable params: 11137 (43.50 KB)
Non-trainable params: 192 (768.00 Byte)
_________________________________________________________________
```

모델5의 모델 생성



모델5의 Confusion matrix

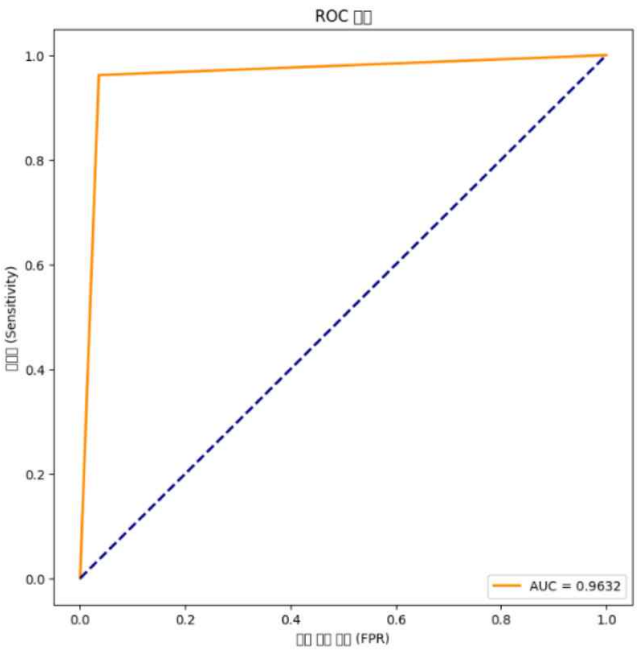


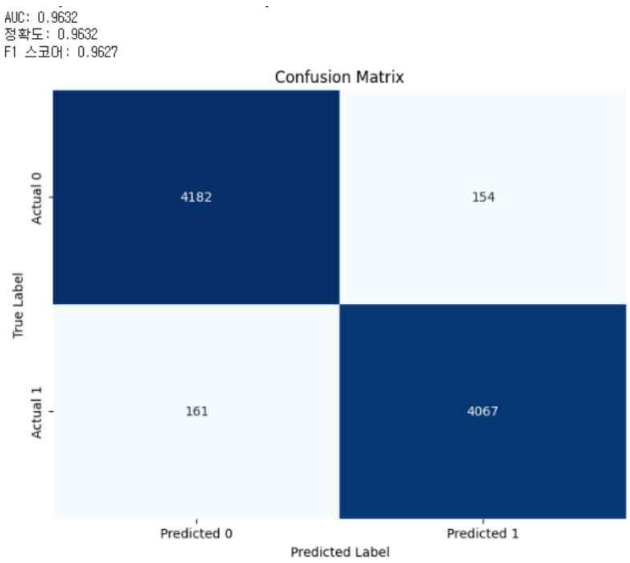
모델5의 loss그래프 및 Accuracy그래프

최적의 모델을 설계하기 위해 모델 노드 수, dropout 값, 은닉층 활성화 함수, 출력층 활성화 함수, 손실 함수, 최적화 함수, epochs, batch size의 하이퍼 파라미터 값을 조정하며 실험하였으며 이때 AUC score: 0.9632 가 나오는 모델을 최적 모델로 선정하였다.

3. 성능평가

(1) 최적 모델(5)의 AUC_ROC커브 그리기





AUC 값이 평가기준의 0.96보다 높은 0.9632가 나왔음으로 일단 가장 기본적인 목표는 달성하였다. AUC 값이 1에 가까울수록 분류를 할 수 있는 모델이라 할 수 있기에 로지스틱 모델을 제외한 모델 중 가장 성능이 좋은 모델이라고 할 수 있다.

(2) Logistic Regression

```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score

logreg = LogisticRegression(random_state=1)

# Set hyperparameters to explore
param_grid = {'penalty': ['l1', 'l2'],
              'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'max_iter': [100, 200, 300, 400]}

grid_search = GridSearchCV(logreg, param_grid, cv=5, scoring='roc_auc')

grid_search.fit(X_train, y_train)

y_pred = grid_search.predict(X_test)

# Calculate AUC score
auc_score = roc_auc_score(y_test, y_pred)

print("Best hyperparameters: ", grid_search.best_params_)
print("Highest AUC score: ", grid_search.best_score_)
print("Best performing model: ", grid_search.best_estimator_)
print("Test data AUC score: ", auc_score)
```

```
Best hyperparameters:  {'C': 100, 'max_iter': 100, 'penalty': 'l2'}
Highest AUC score:  0.9971927128355853
Best performing model:  LogisticRegression(C=100, random_state=1)
Test data AUC score:  0.9562128858043547
```

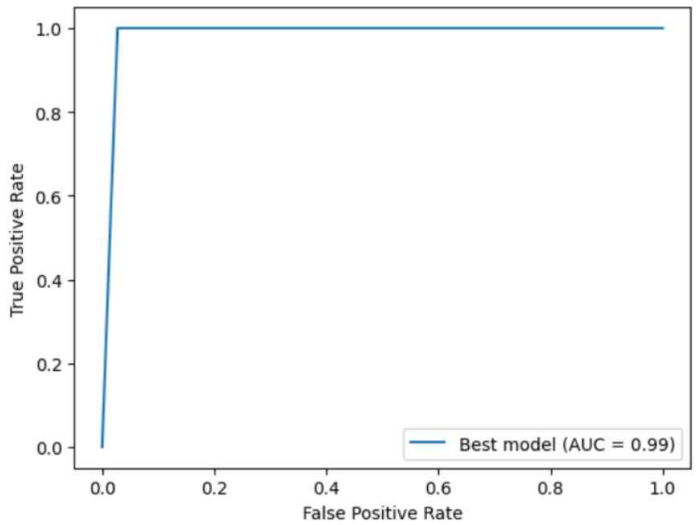
```python
from sklearn.model_selection import ParameterGrid
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

# 파라미터 그리드 생성
param_grid = {'penalty': ['l1', 'l2'],
              'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'max_iter': [100, 200, 300, 400]}

best_auc = 0
best_params = None

# 파라미터 그리드에 대해 모든 조합에 대해 로지스틱 회귀 모델 학습
for params in ParameterGrid(param_grid):
    model = LogisticRegression(penalty=params['penalty'], C=params['C'],\
        max_iter=params['max_iter'], solver='liblinear', random_state= 1)

    model.fit(X_train, y_train)

    # 테스트 데이터에 대한 예측 확률 계산
    y_pred = model.predict(X_test)

    # AUC 계산
    auc_score = roc_auc_score(y_test, y_pred)

    # 파라미터와 AUC 출력
    print(f"Parameters: {params} - AUC: {auc_score:.4f}")

    if auc_score > best_auc:
        best_auc = auc_score
        best_params = params

# 최고 AUC 값을 가진 파라미터 출력
print(f"Best parameters: {best_params} - Best AUC: {best_auc:.4f}")
```

Grid Search CV를 사용하여 로지스틱 회귀의 parameter을 찾고자 하였지만 과적합이 되어 실제 best parameter가 test_set에 적용시키면 AUC값이 낮았다.

그래서 우리는 모든 Parameter에 대해 학습시켜보고 가장 높은 AUC값이 나오는 파라미터를 선정하였다.

```
Best parameters: {'C': 10, 'max_iter': 100, 'penalty': 'l1'} - Best AUC: 0.9863
```

```
Best AUC: 0.9863
Confusion Matrix:
 [[4217  119]
 [   0 4228]]
```



이 경우도 마찬가지로 AUC값이 1에 가까워 좋은 성능의 모델임을 알 수 있다.

딥러닝 모델보다 더 높은 AUC 값을 가지고 있기에 확실히 더 좋은 모델이라고 할 수 있다.

4. 결론

AUC score가 0.96을 넘는 두 가지 모델을 결정했다. 하나는 로지스틱 회귀 모델, 하나는 딥러닝 모델으로 둘 다 좋은 모델임은 분명하지만 로지스틱 모델은 데이터의 분포에 따라 엄청 다른 결과를 출력할 수 있다. 그러나 이번에 푸리에 데이터를 변환한 magnitude 값을 standard scaler를 적용시키게 되면서 로지스틱 모델에 적합한 데이터로 변하여 좋은 성능의 모델을 찾을 수 있었다.

반면에 딥러닝의 경우, 너무 레이어를 많이 쌓게 되면 과적합이 발생하여 test_set에서는 좋은 성능을 내지 못했다. 레이어의 수를 줄이면서 과적합을 완화하면서 결과에 따라 딥러닝 모델의 수정하고, 전이학습을 통해 좀 더 성능을 향상시키고자 하였다. 그 결과 학습 시킬 때 마다 결과가 변하기는 하지만 꾸준히 0.96을 넘는 성능을 보여주면서 좀 더 보편적인 데이터에서도 이용할 수 있는 학습 방법을 경험해보았다.

그리고 중복 데이터의 경우도 학습을 시켜봤지만 중복도가 없는 경우보다 성능이 좋지 못하였다. 그 이유는 푸리에 변환을 진행하고, 각각의 시점을 feature화 했기에 반복되는 주기에 대한 정보만 있는 경우인 중복도가 없는 경우가 좀 더 학습에 유리했다고 추측하였다.

5. 참고 문헌

기계학습론 수업자료

A Study of Vibration and Current Data Characteristic Analysis for Motor Mechanical Fault Level Determination by Deep Learning(호서대학교)

Condition Monitoring and Fault Diagnosis of Electrical Motors(S. Nandi)

6. 소감

모터 상태 판단을 위한 기계학습 팀프로젝트를 진행하면서 데이터 전처리, 특성 추출, 모델 선택, 평가 및 결과 해석 등 다양한 단계에서 우리 스스로 적용해 볼 수 있는 기술을 익힐 수 있었다. 특히, 데이터 전처리 과정에서 시계열 데이터를 다루는 방법을 잘 몰라 이에 대한 이해가 필요했고, 데이터를 모델 학습에 적합한 형태로 만들기 위해 노이즈 제거, 결측치 제거, 스케일링 등에 시간을 많이 투자했던 것 같다. 하지만, 모델의 결과가 잘 나오지 않아 시간 도메인에서 주파수 도메인으로의 변환인 푸리에 변환을 이용해 주파수 특성을 추출하여, 이를 기반으로 모터를 정확도 높게 분류할 수 있었다. 또한, 최적의 모델을 찾기 위해 하이퍼파라미터 튜닝 등을 통해 최적의 모델을 찾을 수 있었다. 이 프로젝트를 통해 모터 상태 판단에 기계학습을 적용하는 전반적인 과정을 경험하면서, 현업에서의 응용 가능성과 도전적인 과제에 대한 이해를 높일 수 있었으며 더 다양하고 복잡한 데이터셋을 맞닥뜨려도 충분히 해결할 수 있는 자신감을 얻었다.