HD현대 AI Challenge <본선>

HD HYUNDAI AI Challenge <Final>

Agenda

1. 탐색적 데이터 분석 (Exploratory Data Analysis, EDA)
2. 데이터 전처리 (Data Preprocessing)
3. 학습 모델 (Learning model)
4. 학습 전략 (Learning strategies)
5. 결론 (Conclusion)

2023년 11월 10일

경기대학교 일반대학원 컴퓨터과학과 석사과정

팀명: 이상민.

발표자 및 팀원: 이상민

시간 경과에 따른 신호 데이터

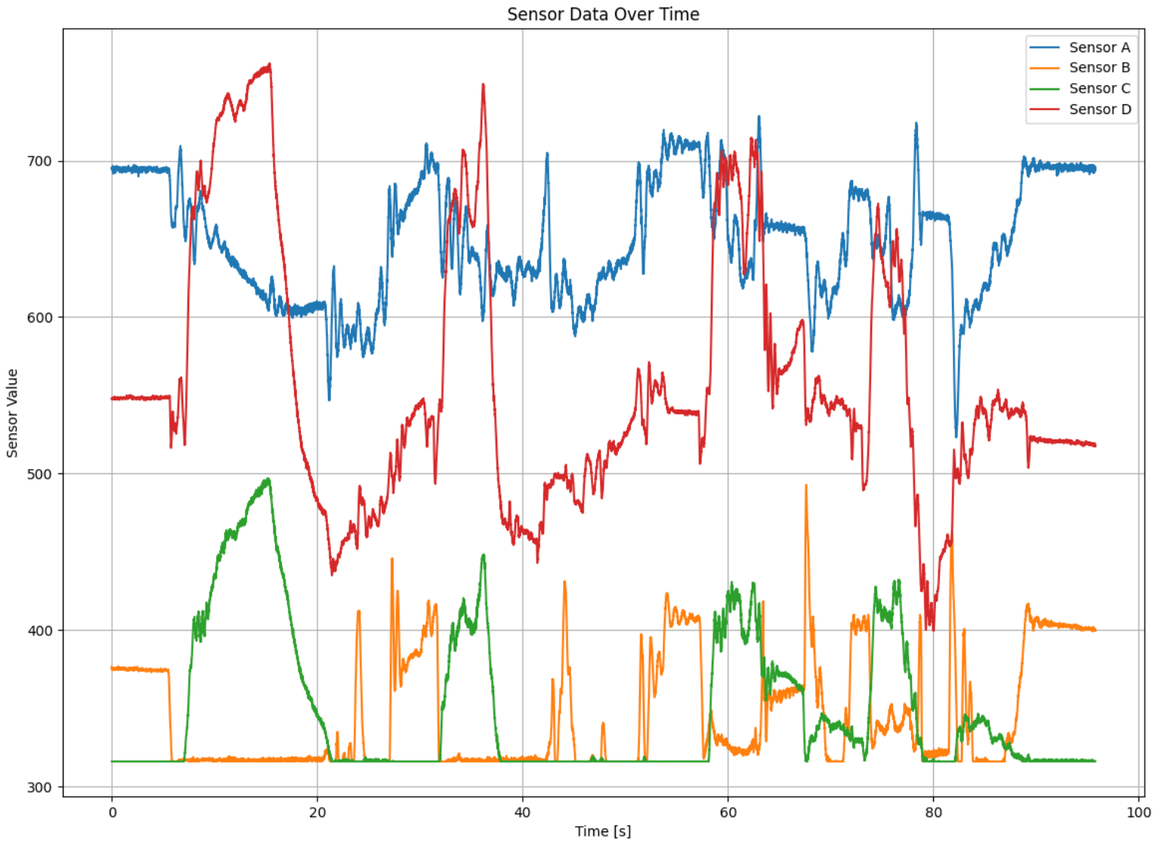


그림 1. 200kg_B운전자데이터.csv의 Sensor 특성 시각화

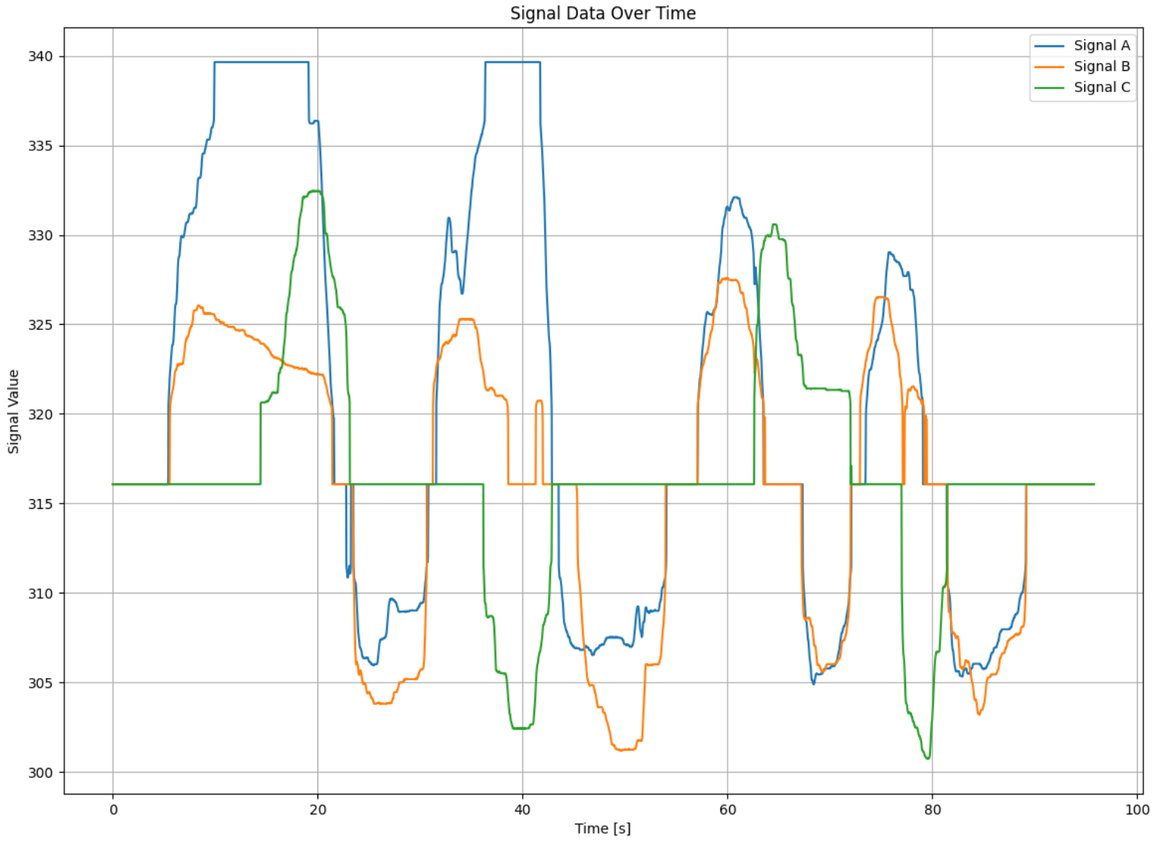


그림 2. 200kg_B운전자데이터.csv의 Signal 특성 시각화

200kg_B운전자데이터.csv 에서 시간에 따른 특성의 시각화를 수행

일정 구간에서 동일한 경향을 보이는 특성 값의 포착하였으며, 이를 근거로 특성간 관계가 있음을 판단

특성간 관계를 분석하기 위해 특 성별 기술분석 수행

시계열 데이터의 특성 별 기술통계 명세

시계열 데이터의 효과적인 반영(스케일링)을 위해 특성 별 통계치 분석 및 상관관계 분석 수행

- **평균(Mean):** 데이터 포인트의 총합을 개수로 나눈 값으로, 데이터의 중심 경향 의미
- **중앙값(Median):** 데이터의 중앙에 위치하는 값
- **최소값(Minimum):** 데이터 중 가장 작은 값
- **최대값(Maximum):** 데이터 중 가장 큰 값
- **표준편차(Standard Deviation):** 평균으로부터 얼마나 떨어져 있는지를 나타내는 값, 데이터의 분산 정도를 의미
- **분산(Variance):** 표준편차의 제곱, 데이터가 퍼져있는 정도를 의미
- **사분위수(Quartiles):**
  - **제1사분위수(Q1):** 하위 25%의 데이터
  - **제2사분위수(Q2) or 중앙값(Median):** 중앙 값을 의미
  - **제3사분위수(Q3):** 하위 75%의 데이터
- **IQR(Interquartile Range):** 제1사분위수와 제3사분위수의 차이
- **왜도(Skewness):** 데이터 대칭성을 나타내는 척도로, 분포의 비대칭 정도를 의미
- **첨도(Kurtosis):** 데이터의 뾰족한 정도와 꼬리의 길이를 나타내는 척도
- **범위(Range):** 최대값과 최소값의 차이, 데이터의 전체적인 퍼짐 정도를 의미
- **합계(Sum):** 모든 데이터 포인트의 합계
- **개수(Count):** 데이터 포인트의 총 개수

Variable	Time[s]	Signal A	Signal B	Signal C	Sensor A	Sensor B	Sensor C	Sensor D
Mean	65.36	318.37	315.55	316.07	648.71	358.88	364.24	542.52
Std	46.24	9.31	7.25	5.28	63.73	47.03	56.07	96.34
Min	0	299.25	292.5	296.7	417.27	316.07	316.07	316.07
25%	30.06	311.23	311.48	316.07	607.23	317.03	316.07	476.43
50%	60.13	316.07	316.07	316.07	647.59	340.62	339.73	526.15
75%	90.2	325.63	321.44	316.07	694.91	394.92	405.49	608.18
Max	244.75	339.64	332.25	339.64	878.78	611.89	598.91	980.87
Variance	2138.57	86.66	52.63	27.84	4061.75	2211.52	3144.14	9281.71
Range	244.75	40.39	39.75	42.94	461.51	295.81	282.84	664.79
IQR	60.14	14.39	9.96	0	87.68	77.89	89.42	131.75
Skew	1.16	0.66	-0.47	0.14	-0.25	1.02	1.01	0.54
Kurtosis	1.89	-0.33	-0.38	4.16	1.07	0.49	0.1	0.05

표 1. 16개 학습 데이터의 특성 별 기술 통계 값

특성 별 기술통계 분석(통계치 분석)

1. **Time[s]**

   Time[s]는 1분에서 2분 사이의 중앙값을 가지며, 큰 범위(0초에서 약 244.75초)와 **높은 표준편차**를 가짐으로써 데이터 포인트가 **평균값에서 멀리 떨어져 있음**, 왜도가 1.16으로 데이터가 오른쪽으로 긴 꼬리를 가지고 있으며, 일부 데이터 포인트가 매우 긴 지속 시간을 가지고 있음, 첨도 1.89로, 정규 분포보다 약간 뾰족한 분포를 가지고 있음을 확인

2. **Signal A와 Signal B**

   두 신호 모두 300에서 340 사이의 값들을 가지며, **비교적 작은 표준편차**를 가지고 있어 값들이 **평균 주위에 밀집**해 있음

   Signal A의 왜도는 0.66로, 이는 분포가 오른쪽으로 약간 긴 꼬리를 가지고 있음을 확인

   Signal B의 왜도는 -0.47로, 이는 분포가 왼쪽으로 약간 긴 꼬리를 가지고 있음을 확인

3. **Signal C**

   Signal C는 5.28로 **낮은 표준편차**를 가지고 있으며, 모든 값들이 **중앙값에 매우 밀집**해 있음 (IQR 0).

   추가적으로 **첨도가 4.16로 매우 높으며**, 이는 **분포가 매우 뾰족**하고 **꼬리 부분에 이상치가 존재**할 가능성이 높다는 것을 의미

4. **Sensor A, Sensor B, Sensor C, Sensor D**

   모든 센서들은 300 이상의 값을 가지며,

   Sensor A와 Sensor D의 **표준편차가 상대적으로 높아**, Sensor A와 Sensor D는 **더 넓은 범위의 값**을 가지고 있음을 확인

   Sensor B와 Sensor C는 **왜도가 1 이상**으로, 분포가 오른쪽으로 긴 꼬리를 가지고 있음을 확인

   이는 **꼬리 부분에 높은 값**의 데이터 포인트가 존재함을 의미

특성 별 기술통계 분석(상관관계 분석)

1. **Time[s]**는 Sensor B 와 중간의 양의 상관관계(약 0.299),
   Signal A, Sensor C 와는 중간의 음의 상관관계(각각 약 -0.227, -0.202)
   이는 시간이 지남에 따라 **Sensor B의 값이 증가**하고,
   반면에 **Signal A와 Sensor C의 값은 감소**하는 경향을 보임
2. **Signal A**와 Signal B는 **매우 강한 양의 상관관계**(약 0.725)
   이는 두 신호가 같은 원인이나 상호 영향을 받을 가능성 높으며,
   Sensor C (약 0.515)와 Sensor D (약 0.447)와도 **강한 양의 상관관계**
   → **Signal A의 변화가 Signel B, Sensor C, Sensor D와 관련**
3. **Signal B** 또한 Sensor C (약 0.529)와 Sensor D (약 0.400)와
   중간 정도의 양의 상관관계를 보여, **Signal A와 유사한 관계**를 보임
4. **Sensor A 와 Sensor B 는 약한 양의 상관관계**(약 0.217)를 보여,
   해당 센서들 간에 약한 상관 관계 가짐
5. **Sensor C 와 Sensor D 는 중간 정도의 양의 상관관계**(약 0.441)
   해당 센서들이 서로 관련된 현상을 측정하거나 유사한 요인의 영향을
   받을 수 있음을 확인

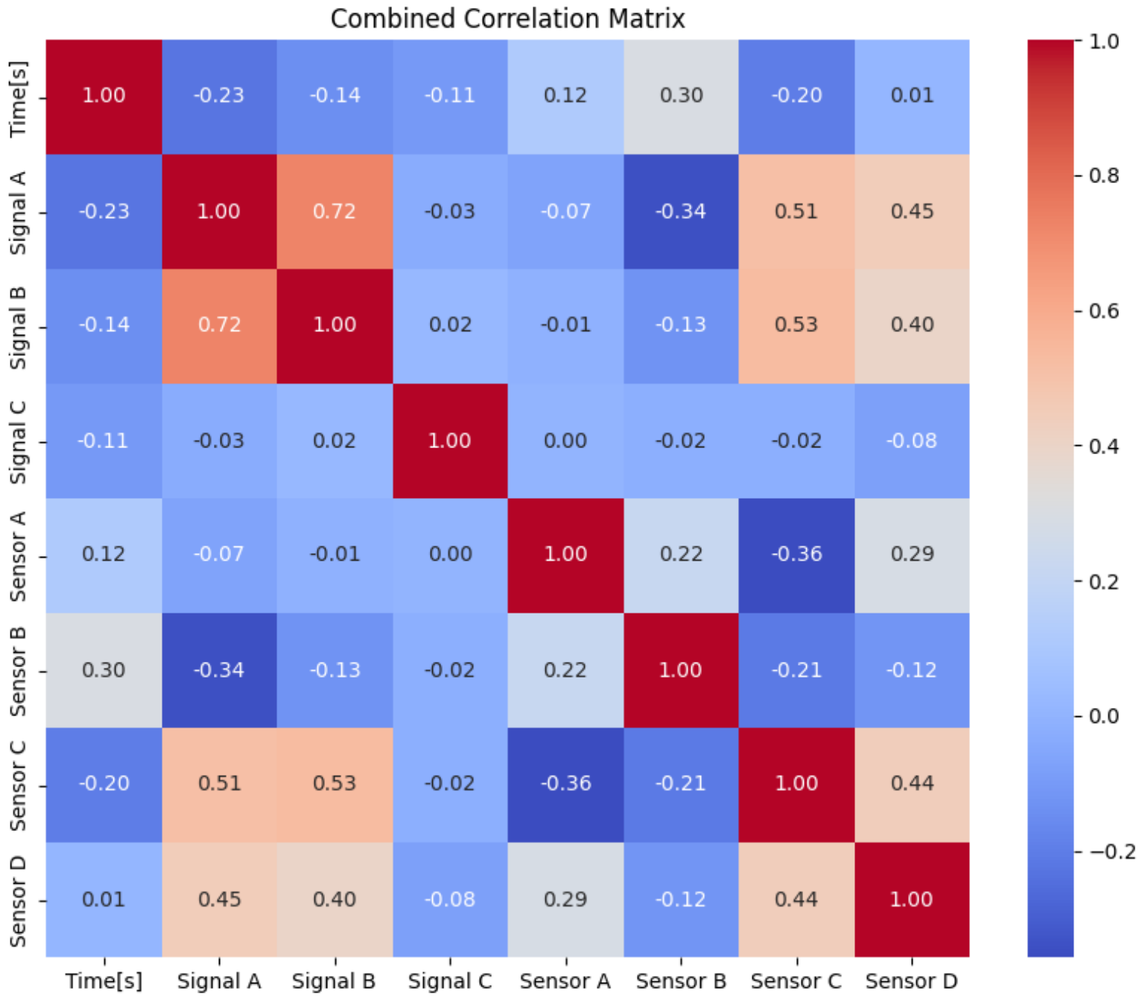


그림 3. 16개 학습 데이터가 합산된 상관 관계 행렬

특성 별 통계 값 계산

```python
extremes = {
    'Time[s]': {'min': float('inf'), 'max': float('-inf'), 'mean': float('inf'), 'std': float('inf'),
                'median': float('inf'), 'sum': float('inf'), 'q1': float('inf'), 'q3': float('inf')},
    'Signal A': {'min': float('inf'), 'max': float('-inf'), 'mean': float('inf'), 'std': float('inf'),
                 'median': float('inf'), 'sum': float('inf'), 'q1': float('inf'), 'q3': float('inf')},
    'Signal B': {'min': float('inf'), 'max': float('-inf'), 'mean': float('inf'), 'std': float('inf'),
                 'median': float('inf'), 'sum': float('inf'), 'q1': float('inf'), 'q3': float('inf')},
    'Signal C': {'min': float('inf'), 'max': float('-inf'), 'mean': float('inf'), 'std': float('inf'),
                 'median': float('inf'), 'sum': float('inf'), 'q1': float('inf'), 'q3': float('inf')},
    'Sensor A': {'min': float('inf'), 'max': float('-inf'), 'mean': float('inf'), 'std': float('inf'),
                 'median': float('inf'), 'sum': float('inf'), 'q1': float('inf'), 'q3': float('inf')},
    'Sensor B': {'min': float('inf'), 'max': float('-inf'), 'mean': float('inf'), 'std': float('inf'),
                 'median': float('inf'), 'sum': float('inf'), 'q1': float('inf'), 'q3': float('inf')},
    'Sensor C': {'min': float('inf'), 'max': float('-inf'), 'mean': float('inf'), 'std': float('inf'),
                 'median': float('inf'), 'sum': float('inf'), 'q1': float('inf'), 'q3': float('inf')},
    'Sensor D': {'min': float('inf'), 'max': float('-inf'), 'mean': float('inf'), 'std': float('inf'),
                 'median': float('inf'), 'sum': float('inf'), 'q1': float('inf'), 'q3': float('inf')}
}

for path in train_paths:
    data = pd.read_csv(path)
    for column in extremes.keys():
        extremes[column]['min'] = min(extremes[column]['min'], data[column].min())
        extremes[column]['max'] = max(extremes[column]['max'], data[column].max())
        extremes[column]['median'] = data[column].median()
        extremes[column]['mean'] = data[column].mean()
        extremes[column]['sum'] = data[column].sum()
        extremes[column]['std'] = data[column].std()
        extremes[column]['q1'] = data[column].quantile(0.25)
        extremes[column]['q3'] = data[column].quantile(0.75)
```

그림 4. 특성 별 통계 값 계산을 위한 과정

1. 각 특성의 최대값, 최소값, 평균, 표준편차, 중앙값, 사분위 수를 담은 사전(dictionary)를 정의
2. Dictionary에 특성 별 통계 값 계산

특성 별 스케일링 반영

1. **최소-최대 정규화 (Min-Max Scaling):**
   Signal A, Signal B, Signal C는 비교적 **작은 범위와 왜도**를 가지고 있기에, 해당 방법을 사용하여 0과 1 사이로 값을 제한하였다. 이를 통해 **모든 특성이 동일한 스케일**을 갖도록 하였으며, 모델이 **모든 입력을 동일하게 취급**하도록 하였다.

2. **표준 정규화 (Standardization):**
   Time[s]는 **큰 범위와 높은 표준편차**를 가지고 있기에, 표준 정규화를 통해 특성의 평균을 0으로, 표준편차를 1로 만들어주었으며, 이를 통해 **다른 특성들과의 균형**을 맞추었다.

3. **로버스트 스케일링 (Robust Scaling):**
   Sensor A, Sensor B, Sensor C, Sensor D는 **이상치의 영향**을 받을 수 있기에, 중앙값과 사분위 범위를 사용하여 로버스트 스케일링을 진행하였으며, **로버스트 스케일링을 통해 이상치에 견고**하게 하였다.

```python
def make_train_data(train_paths, window_size=CFG['WINDOW_SIZE'], stride=10):
    sequences = []
    sequence_labels = []
    for path in tqdm(train_paths):
        driver = str(path.split('/')[-1].split('.')[0].split('_')[1][0])
        data = pd.read_csv(path)
        data['driver'] = 0 if driver == 'A' else 1
        label = float(path.split('/')[-1].split('.')[0].split('_')[0][:-2])
        label = label / 902.
        for i in range(0, len(data) - window_size + 1, stride):
            window_data = data.iloc[i:i + window_size].copy()
            for i, (column, stats) in enumerate(extremes.items()):
                if column in ['Signal A', 'Signal B', 'Signal C']:
                    window_data[column] = (window_data[column] - stats['min']) / (stats['max'] - stats['min'])
                elif column == 'Time[s]':
                    window_data[column] = (window_data[column] - stats['mean']) / stats['std']
                elif column in ['Sensor A', 'Sensor B', 'Sensor C', 'Sensor D']:
                    window_data[column] = (window_data[column] - stats['median']) / (stats['q3'] - stats['q1'])
            sequences.append(window_data.to_numpy())
            sequence_labels.append(label)
    return np.array(sequences), np.array(sequence_labels)

def make_predict_data(test_paths, window_size=CFG['WINDOW_SIZE']):
    sequences = []
    for path in tqdm(test_paths):
        driver = str(path.split('/')[-1].split('.')[0].split('_')[1][0])
        data = pd.read_csv(path)
        data['driver'] = 0 if driver == 'A' else 1
        window_data = np.zeros((window_size, data.shape[1]))
        window_data[:len(data), :] = data.iloc[:len(data)].to_numpy()
        for i, (column, stats) in enumerate(extremes.items()):
            if column in ['Signal A', 'Signal B', 'Signal C']:
                window_data[:, i] = (window_data[:, i] - stats['min']) / (stats['max'] - stats['min'])
            elif column == 'Time[s]':
                window_data[:, i] = (window_data[:, i] - stats['mean']) / stats['std']
            elif column in ['Sensor A', 'Sensor B', 'Sensor C', 'Sensor D']:
                window_data[:, i] = (window_data[:, i] - stats['median']) / (stats['q3'] - stats['q1'])
        sequences.append(window_data)
    return np.array(sequences)
```

그림 5. 특성 별 통계 값을 학습 및 테스트 데이터에 반영 과정

Time Series Transformer (TST)

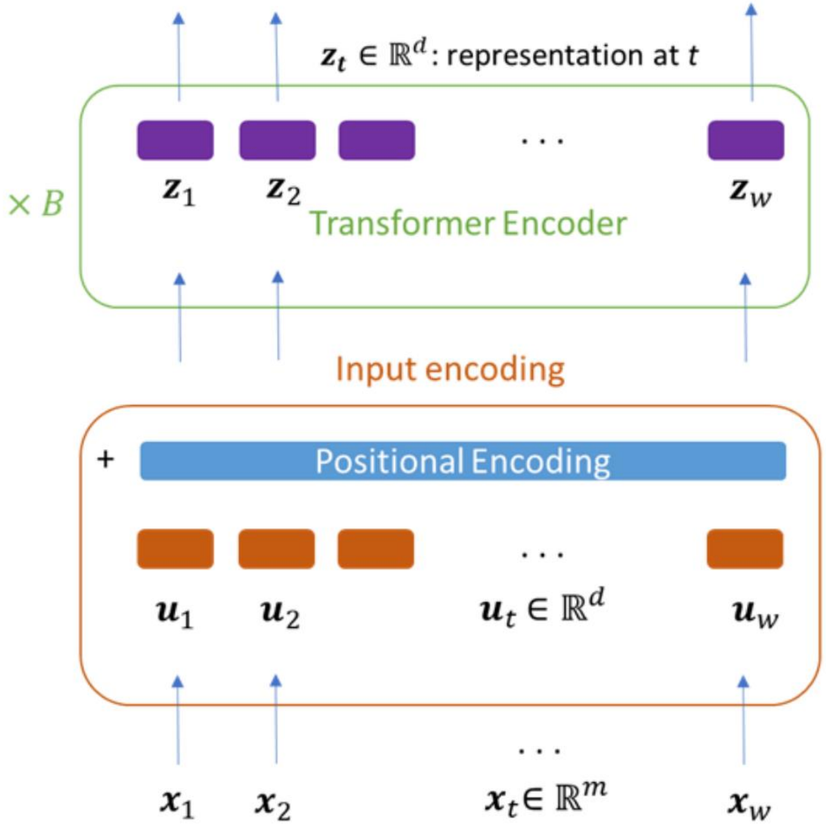
```python
class TimeSeriesTransformer(nn.Module):
    def __init__(self, input_size=9, num_layers=4, nhead=9, max_seq_length=500):
        super(TimeSeriesTransformer, self).__init__()
        self.input_size = input_size
        self.hidden_size = input_size
        self.num_layers = num_layers
        self.nhead = nhead
        encoder_layers = nn.TransformerEncoderLayer(
            d_model=self.hidden_size,
            nhead=nhead,
            dim_feedforward=self.hidden_size * 4,
            dropout=0.1,
            activation='relu'
        )
        self.transformer_encoder = nn.TransformerEncoder(encoder_layers, num_layers)
        self.fc_out = nn.Linear(self.hidden_size, 1)
        self.positional_encoding = nn.Parameter(torch.zeros(1, max_seq_length, self.input_size))

    def forward(self, x):
        batch_size, seq_length, features = x.size()
        pos_encoding = self.positional_encoding[:, :seq_length, :]
        x = x + pos_encoding
        x = x.permute(1, 0, 2)
        transformer_out = self.transformer_encoder(x)
        last_output = transformer_out[-1, :, :]
        output = self.fc_out(last_output)
        return output.squeeze(1)
```

그림 6. TST 모델 정의

1. Transformer의 Encoder 구조만 사용
2. 위치 인코딩을 사용하여, 시계열 데이터의 시간적 특성을 캡처

→ 다양한 Task에 적용 가능

※ LSTM, Dlinear 모델을 통한 실험을 진행하였으나, TST 모델보다 낮은 성능



이미지 출처: George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. 2021. "A Transformer-based Framework for Multivariate Time Series Representation Learning." *In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery Data Mining (KDD '21)*. Association for Computing Machinery, New York, NY, USA, 2114–2124. https://doi.org/10.1145/3447548.3467401

그림 7. TST 모델 구조도

Overpred MAE Loss Function

1. **Overpred MAE:**
   손실 함수에 조건부 로직을 추가하여 예측값이 실제값보다 클 경우 추가 패널티를 부여
   → 과대 예측에 대한 패널티를 부여

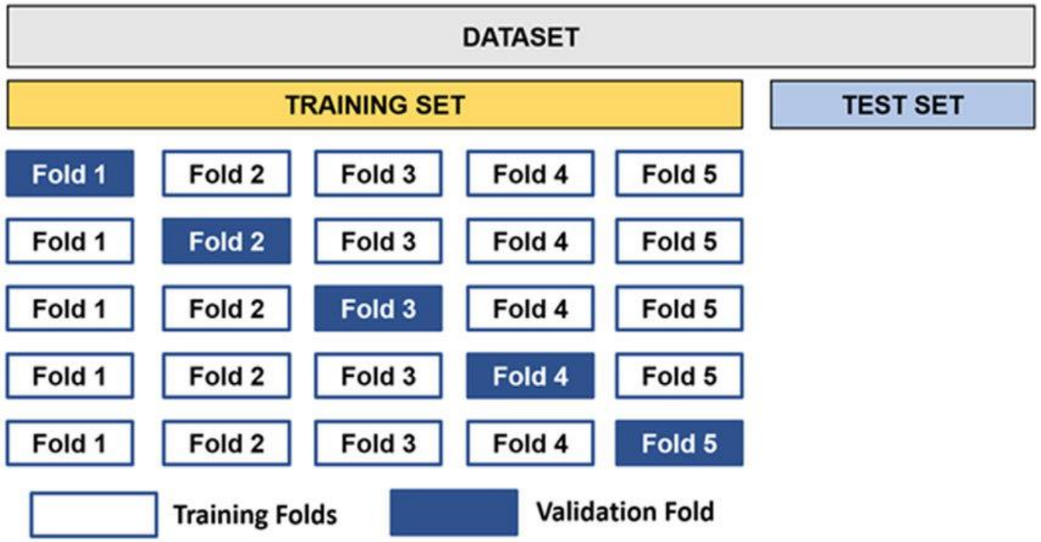
2. predictions가 targets보다 클 때만 20%의 패널티를 추가로 부과 torch.mean을 사용하여 배치 내의 평균 손실을 계산

3. Penalty의 경우 10%, 20%, 25%, 30%의 실험을 진행하였으며, 20%의 경우 최고의 성능을 보임을 확인

```python
class OverpredMAELoss(nn.Module):
    def __init__(self, penalty=0.1):
        super().__init__()
        self.penalty = penalty

    def forward(self, predictions, targets):
        loss = torch.abs(predictions - targets)
        overpred_penalty = torch.where(
          predictions > targets,
          loss * self.penalty,
          torch.zeros_like(loss)
        )
        return torch.mean(loss + overpred_penalty)

```

그림 8. OverpredMAELoss 손실 함수 정의

K-fold Cross Validation



이미지 출처: Srivastava, Amiy & Wang, Ruibin & Dinda, Soumitra & Chattopadhyay, Kinnor. (2021). "Ensemble prediction of mean bubble size in a continuous casting mold using data driven modeling techniques." *Expert Systems with Applications*. vol. 6. pp. 100180. 10.1016/j.mlwa.2021.100180.

그림 9. K-fold 학습 과정 구조도

1. 학습 데이터를 5개의 Fold로 나누어 학습 진행
2. 각각의 Training Set을 학습한 모델을 통해 Test Set의 예측 수행
3. 총 5개 모델의 예측을 합산하여 평균

```python
kf = KFold(n_splits=CFG['FOLD'], shuffle=True, random_state=CFG['SEED'])
fold_performance = []
fold_models = []

for fold, (train_idx, val_idx) in enumerate(kf.split(train_window_data)):
    print(f'Fold {fold} Start!')
    train_subset = Subset(dataset, train_idx)
    val_subset = Subset(dataset, val_idx)
    train_loader = DataLoader(train_subset, batch_size=CFG['BATCH_SIZE'], shuffle=True)
    val_loader = DataLoader(val_subset, batch_size=CFG['BATCH_SIZE'], shuffle=False)
    model = TimeSeriesTransformer(input_size=9, num_layers=4, nhead=9, max_seq_length=500).to(device)
    optimizer = torch.optim.Adam(params=model.parameters(), lr=CFG["LEARNING_RATE"])
    best_model, fold_loss = train(model, optimizer, train_loader, val_loader, device)
    fold_performance.append(fold_loss)
    fold_models.append(best_model)
```

그림 10. 5-fold 학습 과정

- **K-fold Cross Validation을 통해 일반화 성능 강화**
- **데이터 양이 제한적이였기에,**
  데이터 활용 효율성 강화
- **검증 데이터에 대한 편향 감소**
- **예측치의 평균 값을 통해 결과 신뢰성 향상**
- **과적합 방지**

Prediction mapping

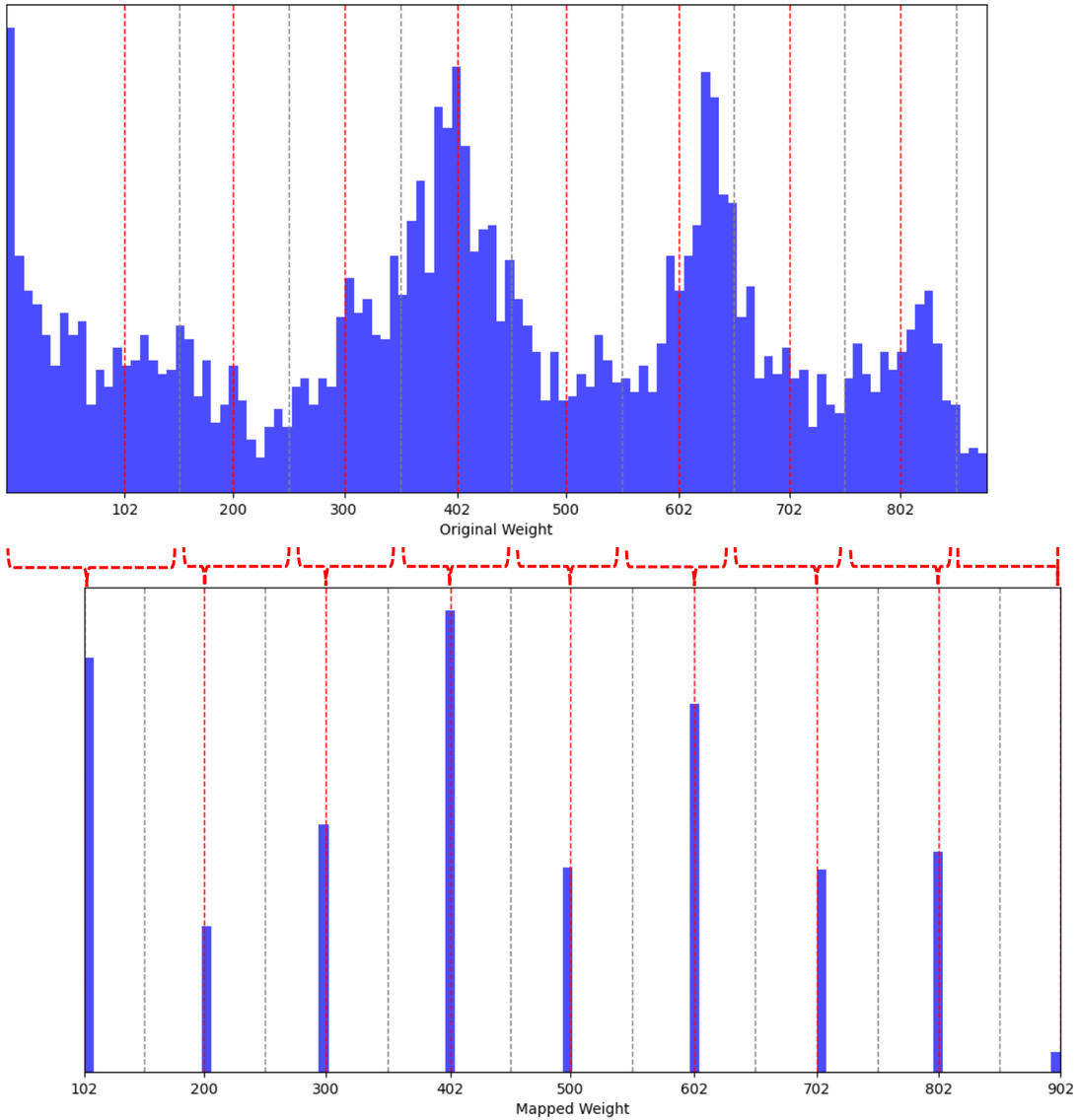


그림 11. 예측 값 매핑 과정

```python
target_values = np.array([102, 200, 300, 402, 500, 602, 702, 802, 902])
submit = pd.read_csv('./sample_submission.csv')

def inference(model, test_loader, device):
    predictions = []
    with torch.no_grad():
        for X in tqdm(iter(test_loader)):
            X = X.to(device)
            output = model(X)
            output = output * 902.
            output = output.cpu().numpy()
            predictions.extend(output)
    return np.array(predictions)

def map_predictions(predictions, target_values):
    mapped_predictions = []
    for pred in predictions:
        differences = np.abs(target_values - pred)
        index_of_min_diff = np.argmin(differences)
        mapped_predictions.append(target_values[index_of_min_diff])
    return np.array(mapped_predictions)
```

그림 12. 평가 함수와 예측 값 매핑 정의

- 학습 데이터에 존재하는 weight를 타겟으로 하여 모델이 예측한 값을 가장 가까운 값으로 매핑
- 후처리 기법을 통해 전체적인 예측 값의 보정을 통해 예측 성능 증가

전체 요약

1. **탐색적 데이터 분석**
    - 학습 데이터의 기술적 분석 수행(통계적, 상관관계)
    - 특성별 데이터 스케일링(최소-최대 정규화, 표준 정규화, 로버스트 스케일링)
2. **학습 모델**
    - Time Series Transformer (TST)
3. **학습 전략**
    - Overpred MAE Loss Function
    - K-fold
    - Prediction mapping
4. **Public Score:** 98.21432
5. **추가적인 성능 개선 가능성:**
    1. 다변량 시계열 모델을 통한 학습 (LSTNet, N-HiTS, MICN)
    2. 하이퍼 파라미터 최적화

Github(Code, Weight): https://github.com/d9249/HDAIChallenge

Docker(Experiment environment): https://hub.docker.com/repository/docker/dodo9249/hdaichallenge/general