# Combustion Quality Classification


**Group 3**

**Lev Slepchuk, Qingyu Wang, Shweta Antony**

# Introduction

# Combustion Quality

How it can be determined?

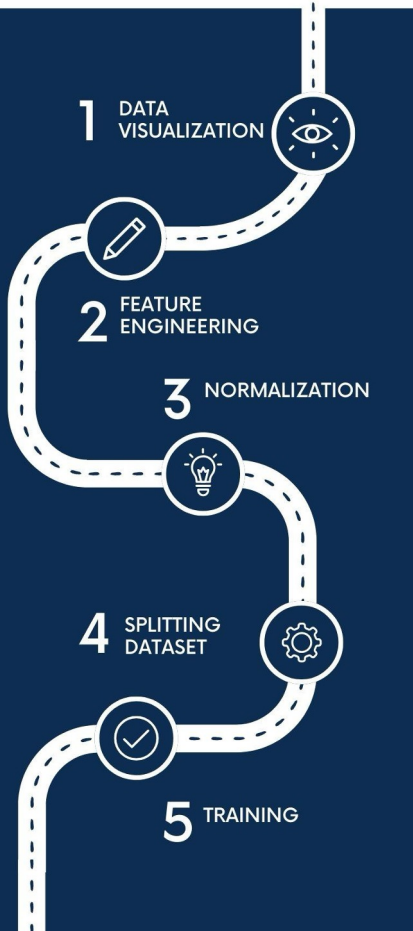
- Heat Signature
- Pressure Signature

**Objective: Determine if pressure data alone can classify combustion quality.**

# Theory

- Data: Two datasets with varying pressure signals.
- Focus: Binary classification in supervised learning.
- Importance: Enhancing efficiency and safety in combustion systems.


THEORY
PROJECT OUTLINE



# Data Preprocessing

# Dataset Analysis

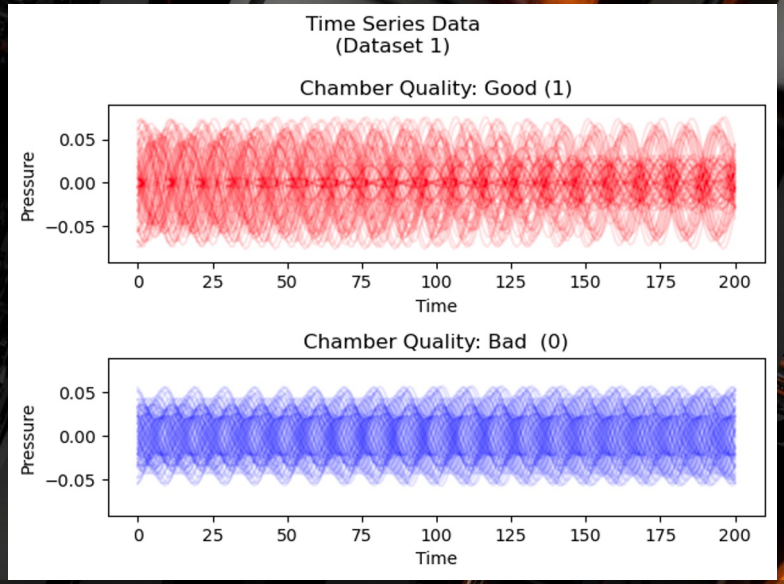
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | -0.006258 | -0.002433 | 0.0015647 | 0.0053475 | 0.0085559 | 0.0108912 | 0.0121447 | 0.0122164 | 0.0111225 |
| 0 | 0.0316902 | 0.03062 | 0.0267976 | 0.0205288 | 0.0123446 | 0.0029572 | -0.006801 | -0.016049 | -0.023941 |

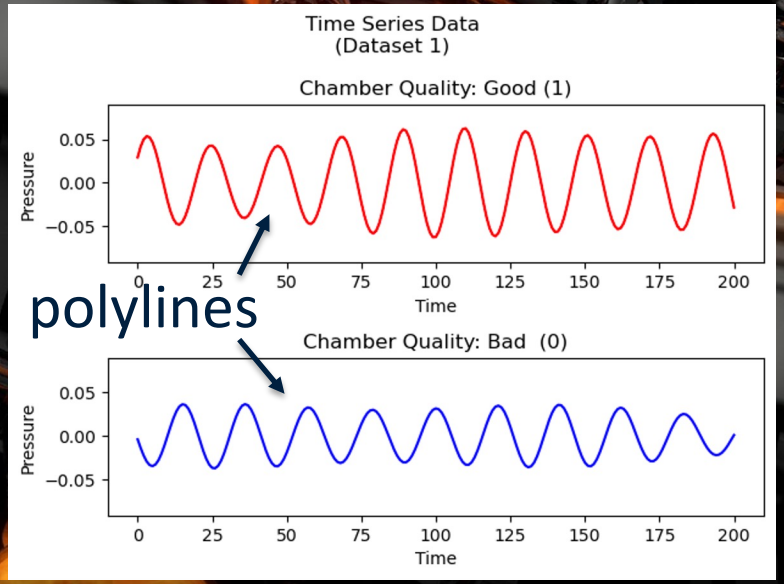
Class

Data Points

- Each dataset contains 154 data points, with 77 labeled positive and 77 negative.
- Total Features: 201 columns of pressure signals.
- Time Interval: Features represent pressure data recorded over specific time frames.
- Splitting datasets: feature set (chamber pressure data) and label set (chamber quality)

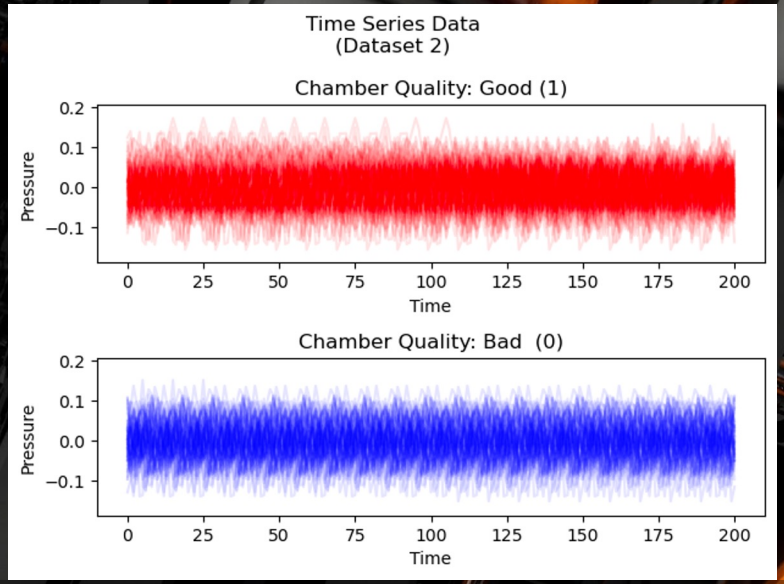
# Data Visualization - Dataset 1

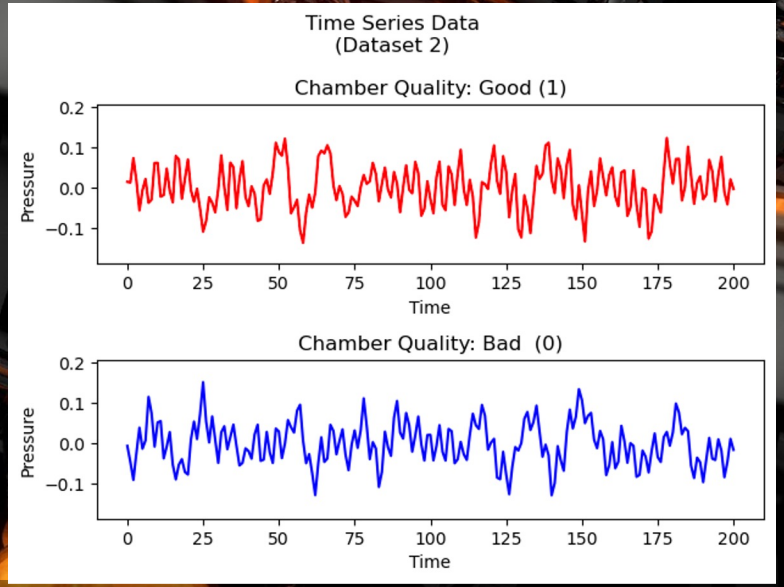



- Aim of visualization: observe how chamber pressure varies over time.
- The coordinate of the x-axis is the number of features, not the time.

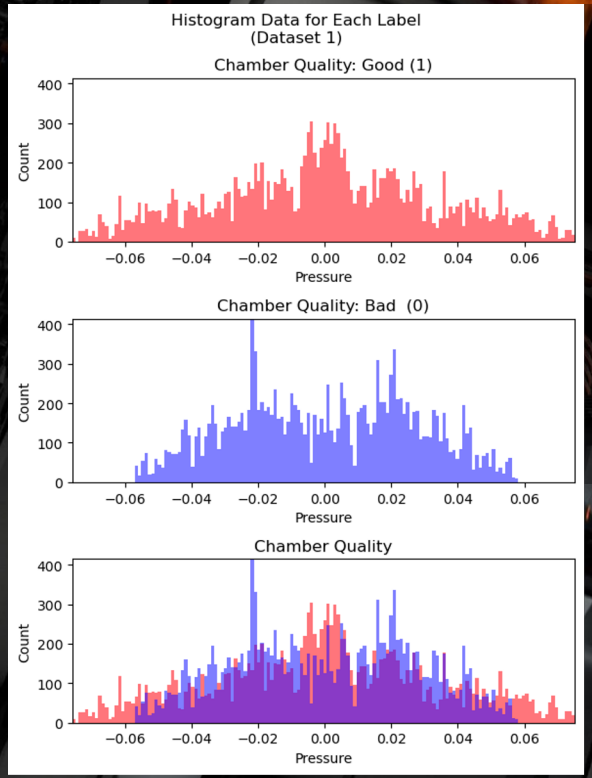
# Data Visualization - Dataset 2



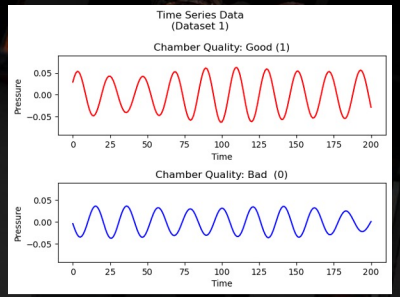



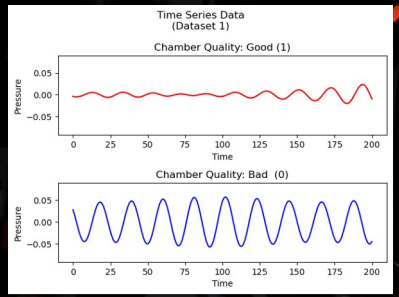

- **Challenge: Greater similarities between different quality data could cause the model to be unable to distinguish effectively.**
- **Feature Engineering: we attempt to extract significantly different features from the raw data for model learning.**

# Feature Engineering - Histogram

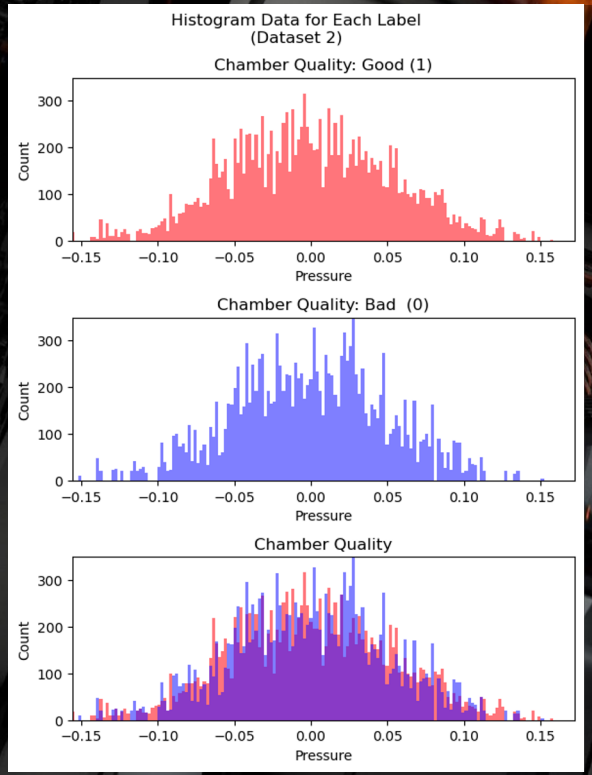



- **Method:** Plotting histograms based on raw data to compare feature distributions.
- **First Dataset Insights:** "Good-quality" data spans a broader pressure, with higher feature distribution around 0.
- **Second Dataset Insights:** "Bad quality" data more even feature distribution.

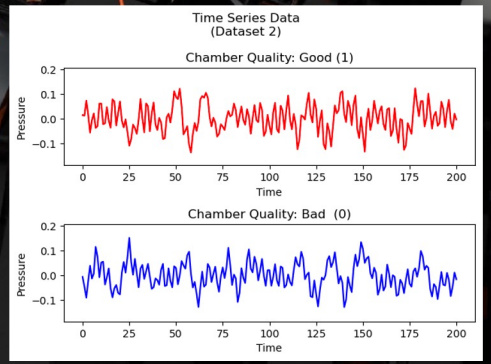


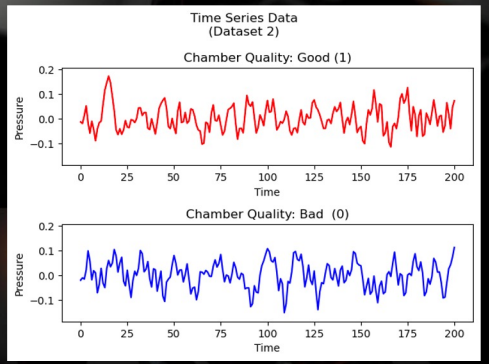



# Feature Engineering - Histogram

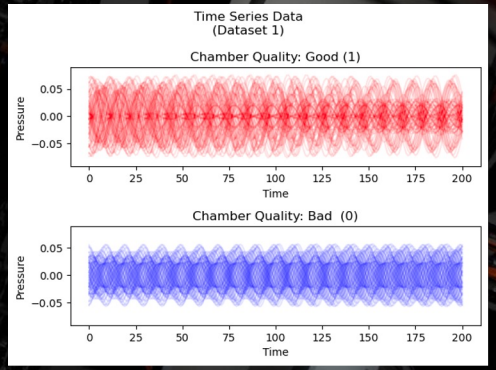


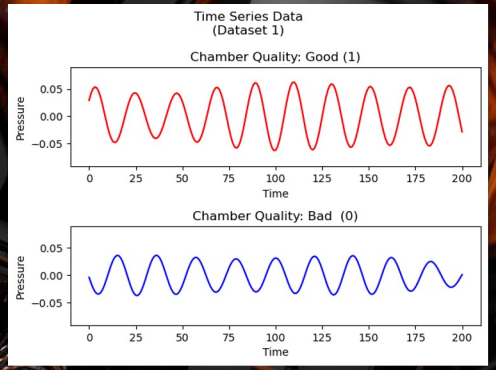

- Dataset 2 has less distinctive differences

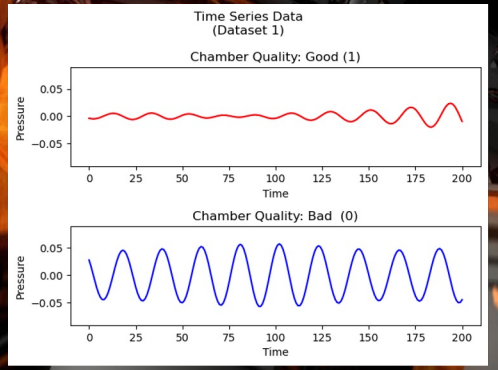


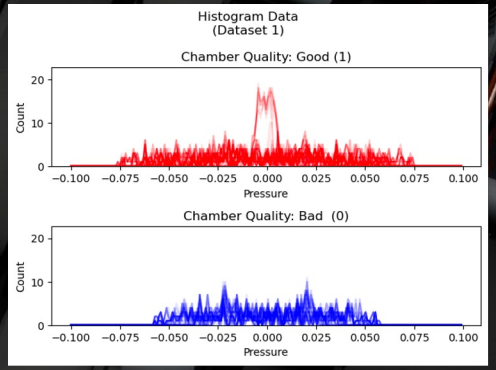



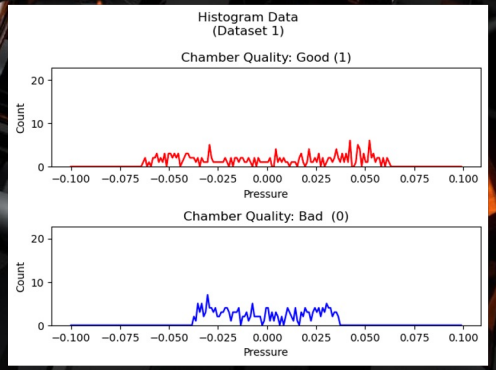
# Feature Comparison - Dataset 1

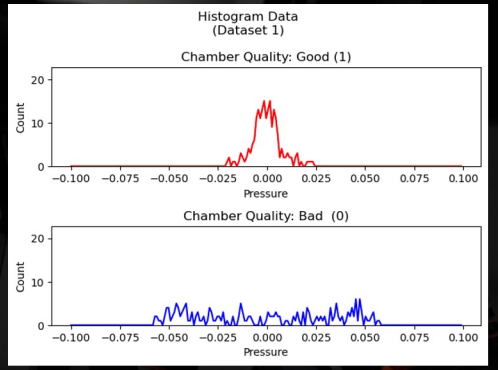


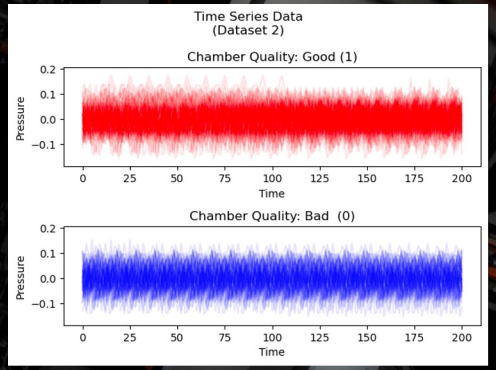



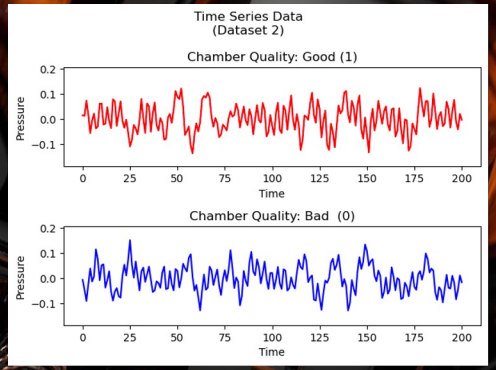



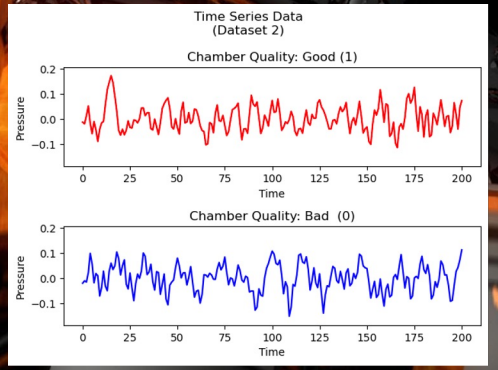



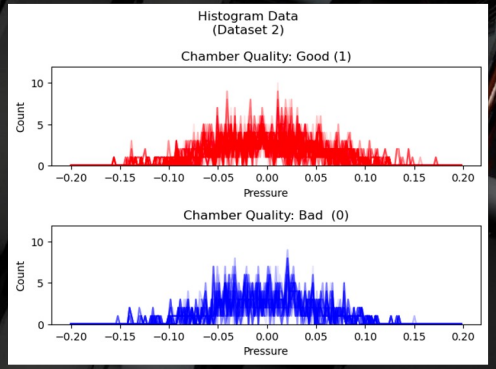



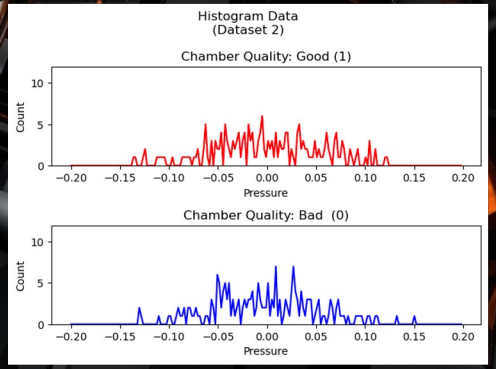
# Feature Comparison - Dataset 2

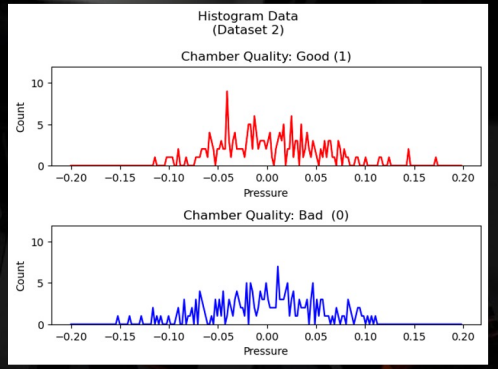


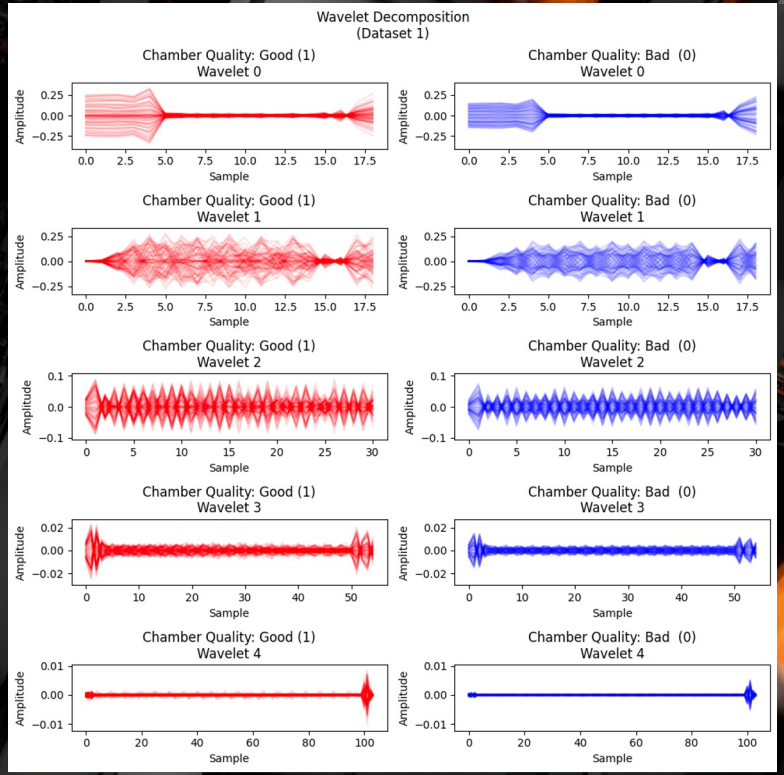



# Feature Engineering - Wavelet Decomposition




## Wavelet Decomposition

- Is not useful because it is also time series data

# Normalization

**Time Series Data Handling**

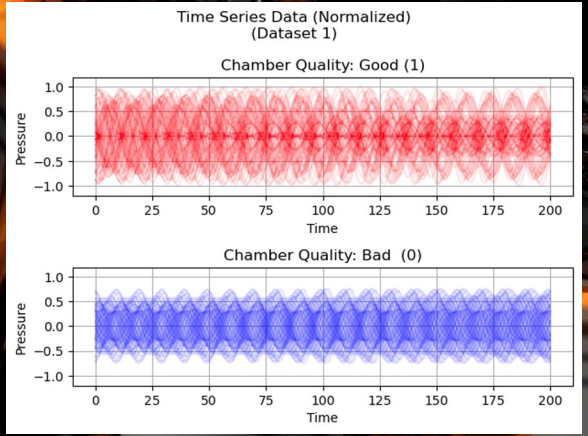
- Scale data from -1 to 1.
- Because it is Time Series data we are not normalizing each feature separately.

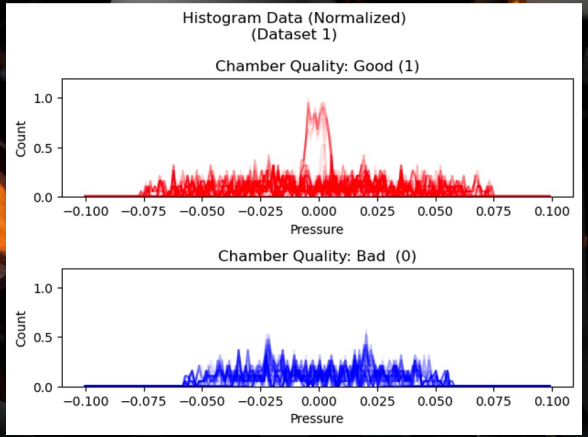
**Histogram Data Scaling**

- Scale data from 0 to 1.
- Normalizing all features at once.

**Impact on Model Training**

- Enhances training stability and performance.
- Ideal for initializing weights between 0 and 1.



# Splitting Dataset

**Dataset Preparation**

- Shuffle and split dataset into training (90%) and testing (10%).

**Label Balance**

- Maintain an 9:1 ratio for both positive and negative labels.

**Sampling Strategy**

- Balanced dataset does not need to be resampled.

# Algorithm Implementation

# Model - Logistic Regression

- Cost Function: Cross-entropy measures prediction accuracy, optimized via gradient descent.
- Parameter Adjustment: Gradient descent minimizes cost, reducing error iteratively.
- Regularization is employed to logistic regression cost function to prevent overfitting.
- Classification Threshold: Predict function classifies: values < 0.5 as class 0, > 0.5 as class 1.
- Implementation and Validation: Autograd for initial implementation, Scikit Learn for result validation.

$$g(\boldsymbol{w}) = -\frac{1}{P}\sum_{p=1}^{P} y_p \log\left(\sigma(\bar{\boldsymbol{x}}_p^T \boldsymbol{w})\right) + (1 - y_p)\log\left(1 - \sigma(\bar{\boldsymbol{x}}_p^T \boldsymbol{w})\right)$$

# Model - Logistic Regression

```python
# Model (y = wx)
def model(x, w):
    result = w[0] + np.dot(x, w[1:])
    return result


# Activation Function
def sigmoid(x):
    result = 1 / (1 + np.exp(-x))
    return result


# Cost Function - Cross Entropy (Lec4 - P12)
def cross_entropy(w, x, y):
    temp = sigmoid(model(x, w))

    cost = 0

    ind = np.argwhere(y==0)[:,0]
    cost += -np.sum(np.log(1-temp[ind,:]))

    ind = np.argwhere(y==1)[:,0]
    cost += -np.sum(np.log(temp[ind,:]))

    cost /= y.size
    return cost
```

```python
# Trainning Process (Gradient Descent)
def gradient_descent(g, step_size, max_iter, w):

    gradient = grad(g)

    history_weight = [w]
    history_cost   = [g(w)]

    for k in range(max_iter):
        grad_eval = gradient(w)
        grad_eval_norm = grad_eval / np.linalg.norm(grad_eval)

        w = w - step_size * grad_eval_norm

        history_weight.append(w)
        history_cost.append(g(w))

    return history_weight, history_cost

def predict(x, w):
    result = sigmoid(model(x, w))          # [0-1] contineous
    result = np.sign(result-0.5) / 2 + 0.5 # [0,1] descrete
    return result
```

```python
_time = time.time()
classifier = LogisticRegression(max_iter=5000)
classifier.fit(x_train, y_train.reshape(-1))
training_time = time.time()-_time
```

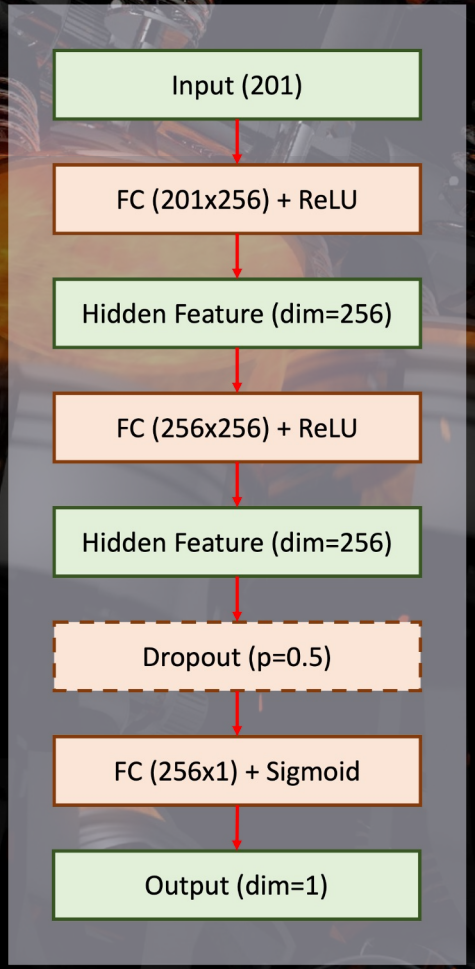
```python
y_pred = classifier.predict(x_test).reshape(-1, 1)

cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test, y_pred)
```

# Model - Neural Network

**PyTorch Model**

- **ReLU**
- **Dropout Layer**

```
model_pth = nn.Sequential(
    nn.Linear(201, 256),
    nn.ReLU(),
    nn.Linear(256, 256),
    nn.ReLU(),
    nn.Dropout(dropout_rate),
    nn.Linear(256, 1),
    nn.Sigmoid()
)
model_pth.apply(init_weights)
```



# Model - Neural Network

```python
def torch_train(x, y, num_epochs):
    x_tensor = torch.Tensor(x)
    y_tensor = torch.Tensor(y).reshape(-1, 1)

    history_cost = []
    history_lr = []

    model_pth.train()
    for n in range(num_epochs):

        y_pred_tensor = model_pth(x_tensor)

        loss = criterion(y_pred_tensor, y_tensor)

        history_cost.append(loss.detach())
        history_lr.append(scheduler.get_last_lr())

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        scheduler.step()

    return history_cost, history_lr
```

```python
def torch_predict(x):
    x_tensor = torch.Tensor(x)
    model_pth.eval()
    y_pred_tensor = model_pth.forward(x_tensor)
    y_pred = y_pred_tensor.detach()
    y_pred[y_pred>0.5] = 1
    y_pred[y_pred<0.5] = 0
    return y_pred


def init_weights(m):
    if isinstance(m, nn.Linear):
        torch.nn.init.kaiming_uniform_(m.weight)
        m.bias.data.fill_(0.01)
```

```python
criterion = nn.BCELoss()

optimizer = torch.optim.SGD(model_pth.parameters(), lr=0.1, weight_decay=0.01)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5000, gamma=0.1)
```

# Model - Neural Network

**Scikit Learn Model**

- **Use the basic setting as a baseline for reference.**

```
mlp = MLPClassifier(
    hidden_layer_sizes=(256, 256, 1),
    activation="relu",
    learning_rate_init=0.001,
    max_iter=5000
)
```

# Training Process

**Dataset**

- Dataset 1 (easy to distinguish)
- Dataset 2 (hard to distinguish with noise)

**Feature**

- Time Series Data
- Histogram Data

**Algorithms**

- Logistic Regression (Autograd & Scikit Learn)
- Neural Network (PyTorch & Scikit Learn)

# Training Process

**Strategy**

- Regularization

**Evaluation Metrics**

- Confusion Matrix
- Accuracy

# Training Process

**Problem**

- Lack of data

**Solution**

- Randomly split the dataset at a ratio of 9:1 each time (trade-off)
- Execute 100 times => mean and standard deviation of accuracy

**Result**

- Perform technique similar to “cross validation”
- Lower STD means higher stability and non-overfit

# Results

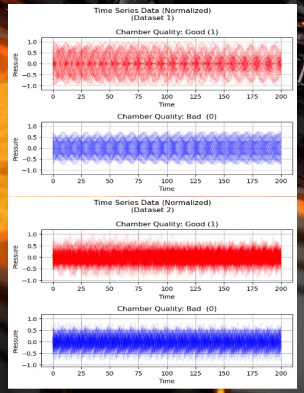
# Training Result

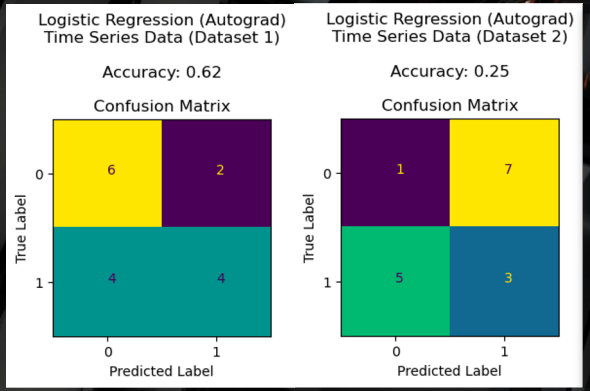
| Accuracy (average of 100 results) | Time Series Dataset 1 | Time Series Dataset 2 | Histogram Dataset 1 | Histogram Dataset 2 |
|---|---|---|---|---|
| Logistic Regression Autograd | 0.3856 STD 0.1072 | 0.3019 STD 0.1086 | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 |
| Logistic Regression Scikit Learn | 0.2906 STD 0.0872 | 0.2894 STD 0.1006 | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 |
| Neural Network PyTorch | 0.9425 STD 0.0557 | 0.4419 STD 0.1229 | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 |
| Neural Network PyTorch (dropout=0.5) | 0.9663 STD 0.0445 | 0.4088 STD 0.1194 | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 |
| Neural Network Scikit Learn | 0.7450 STD 0.2236 | 0.4550 STD 0.0977 | 0.7800 STD: 0.2482 | 0.7550 STD: 0.2499 |

| Training Time (s) (average of 100 results) | Time Series Dataset 1 | Time Series Dataset 2 | Histogram Dataset 1 | Histogram Dataset 2 |
|---|---|---|---|---|
| Logistic Regression Autograd | 0.1170 300 iters | 0.0424 100 iters | 0.0422s 100 iters | 0.0517 120 iters |
| Logistic Regression Scikit Learn | 0.0043 44 iters | 0.0038 32 iters | 0.0021 22 iters | 0.0024 24 iters |
| Neural Network PyTorch | 0.5572 1000 iters | 0.5892 1000 iters | 0.1141 200 iters | 0.1171 200 iters |
| Neural Network PyTorch (dropout=0.5) | 0.7302 1000 iters | 0.7466 1000 iters | 0.1463 200 iters | 0.1524 200 iters |
| Neural Network Scikit Learn | 2.9370 732 iters | 2.6057 705 iters | 2.3134 631 iters | 2.477 638 iters |

# Performance Analysis - Time Series Data

| Accuracy (average of 100 results) | Time Series Dataset 1 | Time Series Dataset 2 | Histogram Dataset 1 | Histogram Dataset 2 |
|---|---|---|---|---|
| Logistic Regression Autograd | 0.3856 STD 0.1072 | 0.3019 STD 0.1086 | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 |
| Logistic Regression Scikit Learn | 0.2906 STD 0.0872 | 0.2894 STD 0.1006 | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 |
| Neural Network PyTorch | 0.9425 STD 0.0557 | 0.4419 STD 0.1229 | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 |
| Neural Network PyTorch (dropout=0.5) | 0.9663 STD 0.0445 | 0.4088 STD 0.1194 | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 |
| Neural Network Scikit Learn | 0.7450 STD 0.2236 | 0.4550 STD 0.0977 | 0.7800 STD: 0.2482 | 0.7550 STD: 0.2499 |

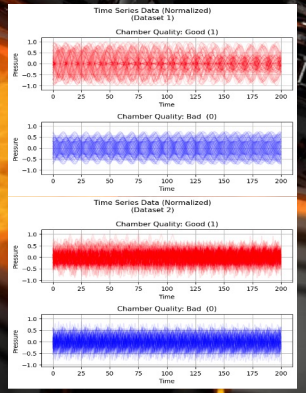


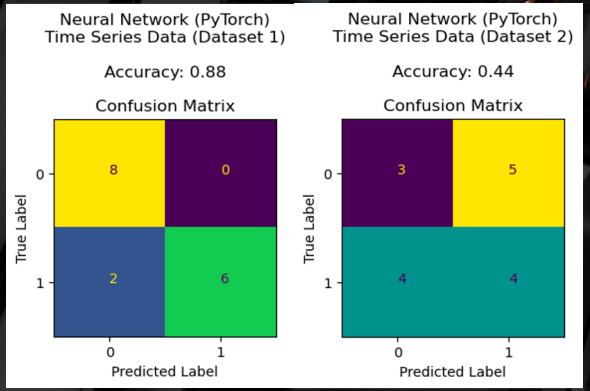




- LR model can not learn anything from the time series data.

# Performance Analysis - Time Series Data

| Accuracy (average of 100 results) | Time Series Dataset 1 | Time Series Dataset 2 | Histogram Dataset 1 | Histogram Dataset 2 |
|---|---|---|---|---|
| **Logistic Regression Autograd** | 0.3856 STD 0.1072 | 0.3019 STD 0.1086 | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 |
| **Logistic Regression Scikit Learn** | 0.2906 STD 0.0872 | 0.2894 STD 0.1006 | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 |
| **Neural Network PyTorch** | 0.9425 STD 0.0557 | 0.4419 STD 0.1229 | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 |
| **Neural Network PyTorch (dropout=0.5)** | 0.9663 STD 0.0445 | 0.4088 STD 0.1194 | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 |
| **Neural Network Scikit Learn** | 0.7450 STD 0.2236 | 0.4550 STD 0.0977 | 0.7800 STD: 0.2482 | 0.7550 STD: 0.2499 |

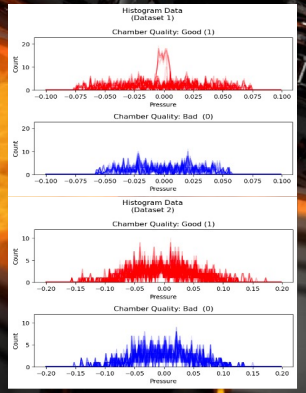


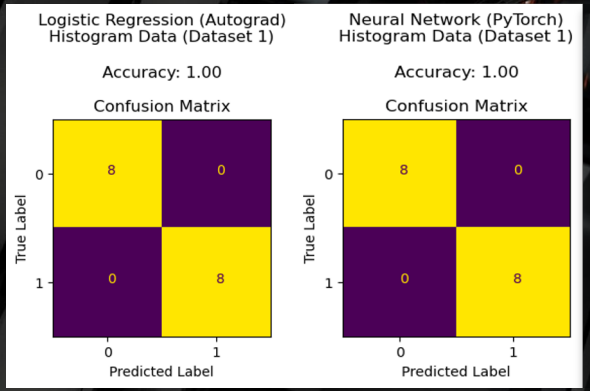




- NN model can learn the patterns in Dataset 1, but still fails in Dataset 2.
- Dropout layer can improve performance.
- Model is not stable enough.

# Performance Analysis - Histogram Data

| Accuracy (average of 100 results) | Time Series Dataset 1 | Time Series Dataset 2 | Histogram Dataset 1 | Histogram Dataset 2 |
|---|---|---|---|---|
| **Logistic Regression Autograd** | 0.3856 STD 0.1072 | 0.3019 STD 0.1086 | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 |
| **Logistic Regression Scikit Learn** | 0.2906 STD 0.0872 | 0.2894 STD 0.1006 | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 |
| **Neural Network PyTorch** | 0.9425 STD 0.0557 | 0.4419 STD 0.1229 | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 |
| **Neural Network PyTorch (dropout=0.5)** | 0.9663 STD 0.0445 | 0.4088 STD 0.1194 | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 |
| **Neural Network Scikit Learn** | 0.7450 STD 0.2236 | 0.4550 STD 0.0977 | 0.7800 STD: 0.2482 | 0.7550 STD: 0.2499 |




- **LR models can easily learn patterns in histogram data and achieve perfect results.**

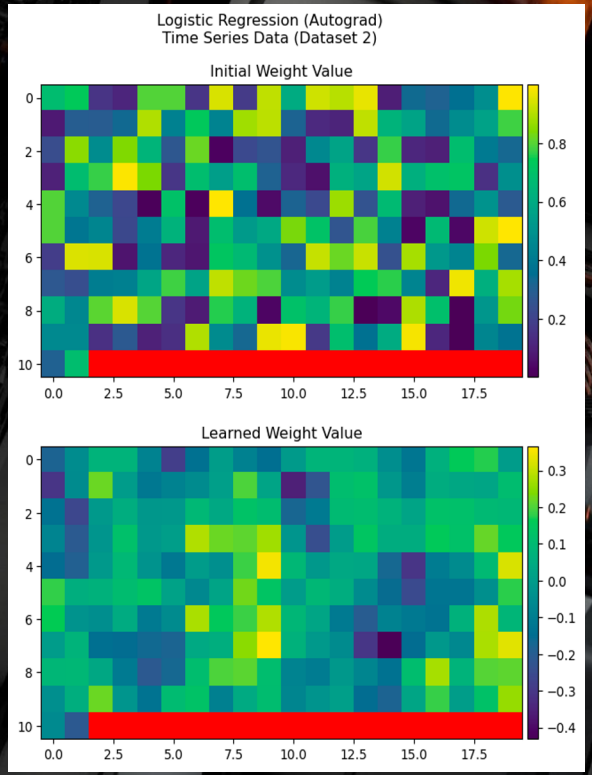
# Performance Analysis - Time Efficiency

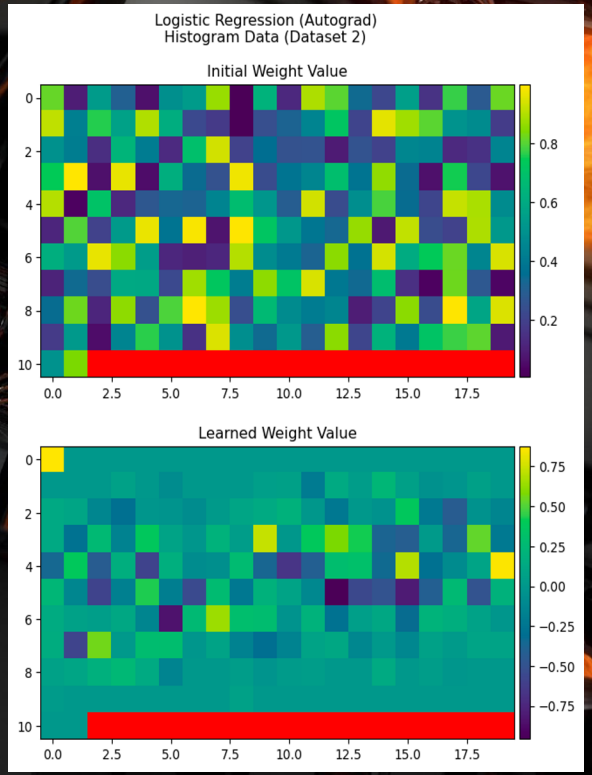
| Training Time (s) (average of 100 results) | Time Series Dataset 1 | Time Series Dataset 2 | Histogram Dataset 1 | Histogram Dataset 2 |
|---|---|---|---|---|
| Logistic Regression Autograd | 0.1170<br>300 iters | 0.0424<br>100 iters | 0.0422s<br>100 iters | 0.0517<br>120 iters |
| Logistic Regression Scikit Learn | 0.0043<br>44 iters | 0.0038<br>32 iters | 0.0021<br>22 iters | 0.0024<br>24 iters |
| Neural Network PyTorch | 0.5572<br>1000 iters | 0.5892<br>1000 iters | 0.1141<br>200 iters | 0.1171<br>200 iters |
| Neural Network PyTorch (dropout=0.5) | 0.7302<br>1000 iters | 0.7466<br>1000 iters | 0.1463<br>200 iters | 0.1524<br>200 iters |
| Neural Network Scikit Learn | 2.9370<br>732 iters | 2.6057<br>705 iters | 2.3134<br>631 iters | 2.477<br>638 iters |

Training LR model on histogram data has the lowest computational cost.

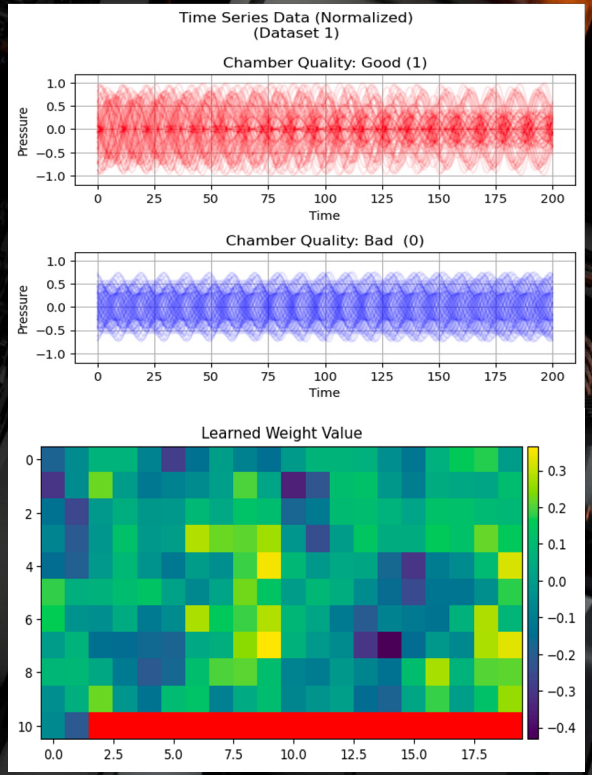
- x50 faster than training NN model on histogram data
- x250 faster than training NN model on time series data

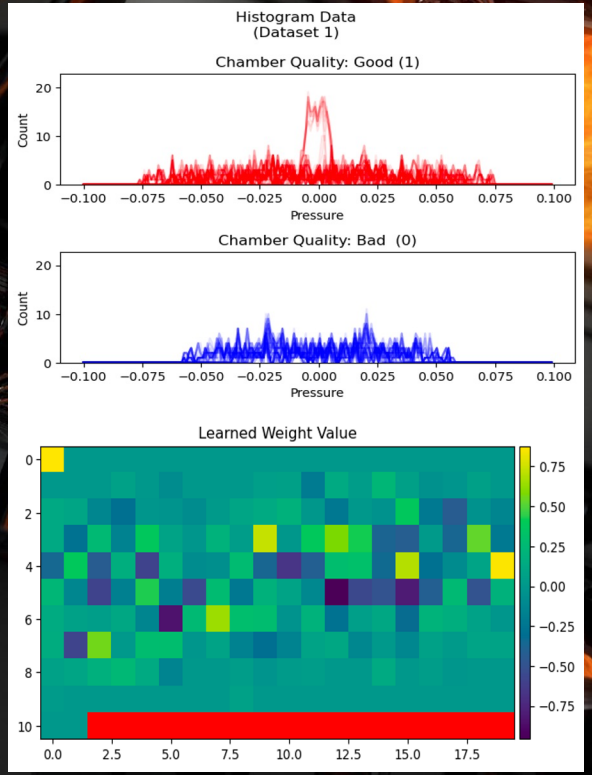
# Performance Analysis - Weight Heatmap



# Performance Analysis - Weight Heatmap



# Conclusion

**Project Conclusion**

- **Pressure data alone can effectively classify combustion quality.**

**Significance of Feature Engineering**

- **Simplify Problem**
- **Increase Computational Efficiency**

# Additional Experiments

Experiment

1. Training model on different split ratios
2. Training model on dataset A and testing on dataset B
3. Training model on combine dataset

Feature in Former Experiments

- normalized histogram data from time series data

Feature in Additional Experiment

- normalized histogram data from normalized time series data

# Extra Experiment

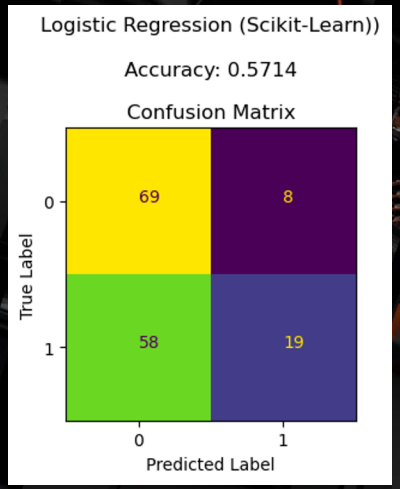
## Experiment 1: Training model on different split ratios

| Accuracy (average of 500 results) | Dataset 1 split_ratio=0.9 | Dataset 1 split_ratio=0.8 | Dataset 1 split_ratio=0.7 | Dataset 1 split_ratio=0.6 | Dataset 1 split_ratio=0.5 |
|---|---|---|---|---|---|
| Logistic Regression Scikit Learn | 1.0000 STD 0.0000 | 0.9996 STD 0.0047 | 0.9990 STD 0.0061 | 0.9976 STD 0.0087 | 0.9939 STD 0.0156 |

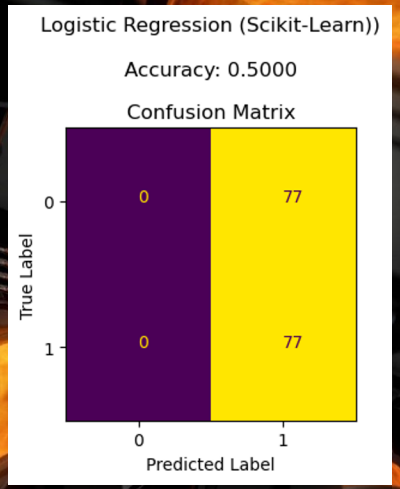
| Accuracy (average of 500 results) | Dataset 2 split_ratio=0.9 | Dataset 2 split_ratio=0.8 | Dataset 2 split_ratio=0.7 | Dataset 2 split_ratio=0.6 | Dataset 2 split_ratio=0.5 |
|---|---|---|---|---|---|
| Logistic Regression Scikit Learn | 1.0000 STD 0.0000 | 1.0000 STD 0.0000 | 0.9998 STD 0.0026 | 0.9995 STD 0.0037 | 0.9992 STD 0.0043 |

# Extra Experiment

**Experiment 2: Training model on dataset A and testing on dataset B**




Train 1 Test 2




Train 2 Test 1

# Extra Experiment

## Experiment 3: Training model on combined dataset

| Accuracy (average of 500 results) | Dataset 1+2 split_ratio=0.9 | Dataset 1+2 split_ratio=0.8 | Dataset 1+2 split_ratio=0.7 | Dataset 1+2 split_ratio=0.6 |
|---|---|---|---|---|
| Logistic Regression Scikit Learn | 0.9999 STD 0.0014 | 0.9997 STD 0.0027 | 0.9995 STD 0.0028 | 0.9990 STD 0.0039 |

| Accuracy (average of 500 results) | Dataset 1+2 split_ratio=0.5 | Dataset 1+2 split_ratio=0.4 | Dataset 1+2 split_ratio=0.3 | Dataset 1+2 split_ratio=0.2 |
|---|---|---|---|---|
| Logistic Regression Scikit Learn | 0.9977 STD 0.0064 | 0.9954 STD 0.0091 | 0.9888 STD 0.0151 | 0.9694 STD 0.0256 |

| Accuracy (average of 100 results) | Time Series Dataset 1 |
|---|---|
| Neural Network PyTorch (dropout=0.5) | 0.9663 STD 0.0445 |

# Future Experiment

**Experiment**

1. **Denoise the dataset 2**
2. **Reduce the histogram features**
3. **…**

# Reference

[1] M. Ihme, W. T. Chung, and A. A. Mishra, “Combustion machine learning: Principles, progress and prospects,” Progress in Energy and Combustion Science, vol. 91, p. 101010, 2022.

[2] Watt, J., Borhani, R., & Katsaggelos, A. K. (2016). Machine Learning Refined: Foundations, Algorithms, and Applications (2nd ed.). Cambridge University Press. (Chapter 9.2 "Feature Engineering and Selection: Histogram Features").


Q&A