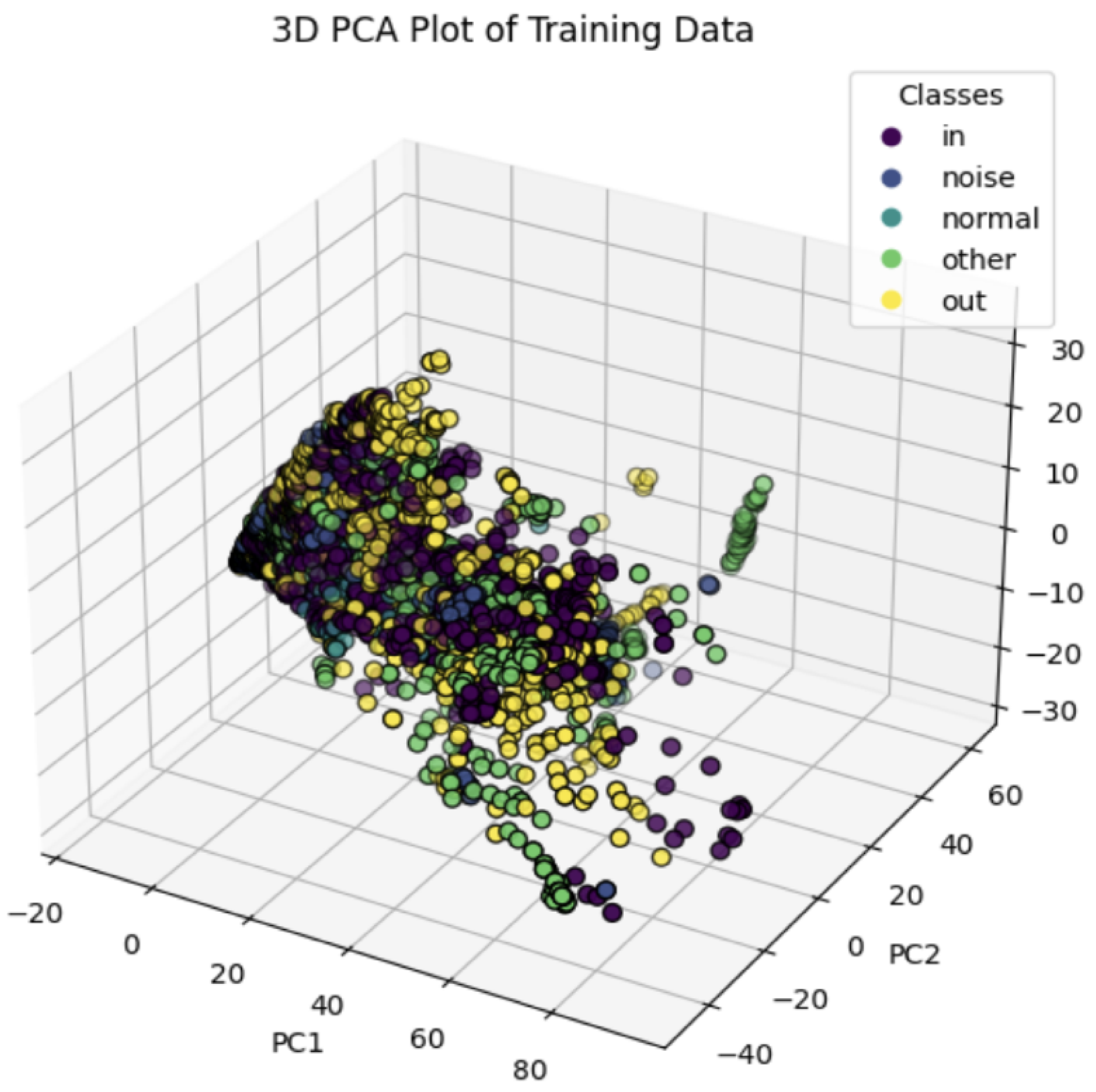
Classification of Water Supply Pipeline Leakage

이권림, 송지운, 편예빈

머신러닝 기초 2024-1

[목차]

[1. Attempting SVM]



1 겹치는 다른 클래스의 데이터 포인트들

→ 효과적으로 클래스를 분리할 선형 hyperplane을 찾기가 어려워진다

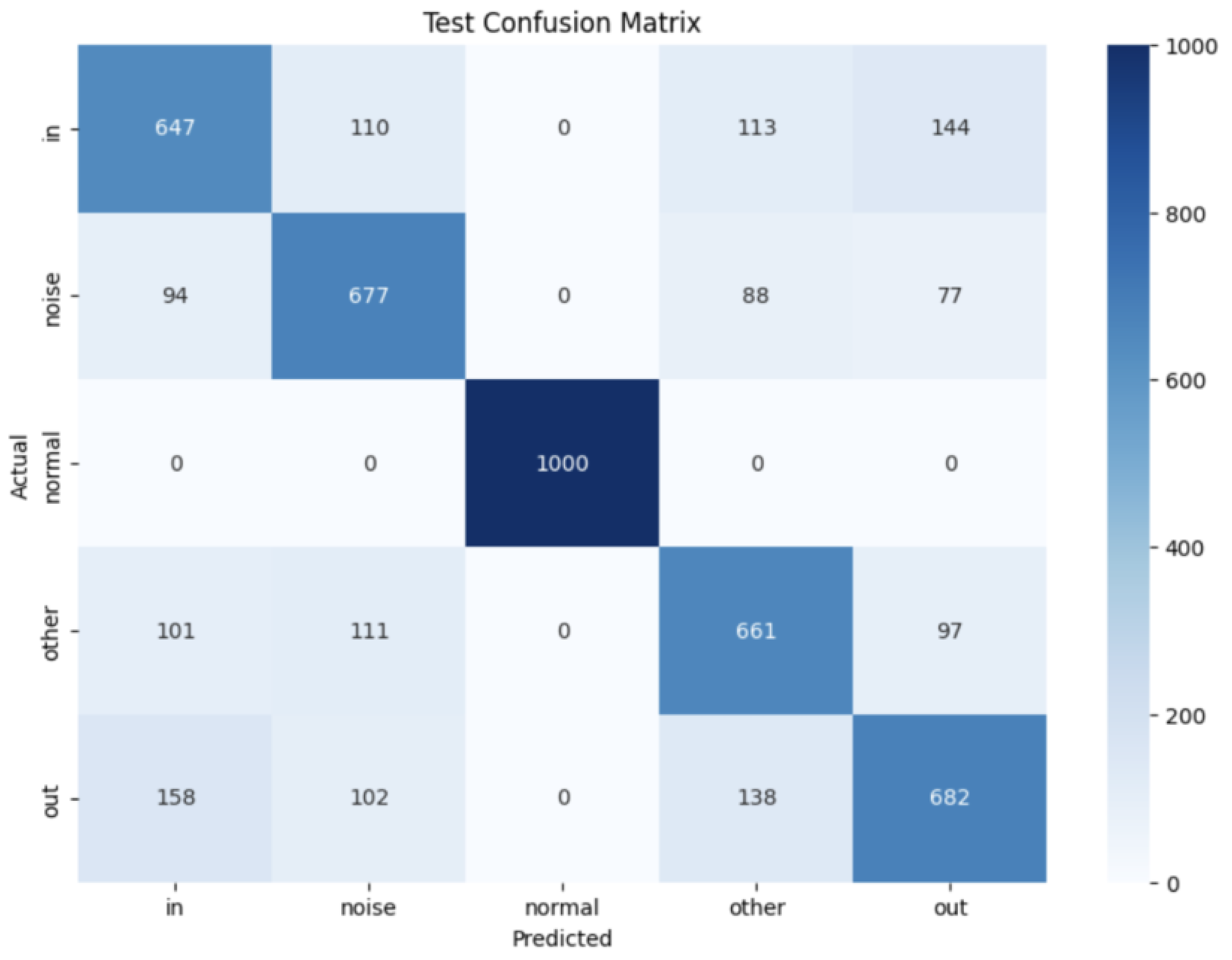
2 Kernel Function 사용

→ 차원의 복잡성으로 계산 비용 증가
→ 충분한 너비의 margin을 확보하지 못해 classification의 성능 감소

3 과적합과 계산 복잡성

데이터셋은 500개 이상의 특성을 가진 고차원 공간이다.
이런 상황에서는 과적합과 계산 복잡성 문제가 더 심화될 수 있다.

[2. Attempting Decision Tree]




```
Test Accuracy: 0.7334
Test Precision: {'in': 0.647, 'noise': 0.677, 'normal': 1.0, 'other': 0.661, 'out': 0.682}
Test Recall: {'in': 0.6380670611439843, 'noise': 0.7232905982905983, 'normal': 1.0,
'other': 0.6814432989690722, 'out': 0.6314814814814815}
```

1 Decision Tree의 장점

→ 데이터의 공간적 겹침에 덜 민감하다
→ 데이터 전처리 과정이 적고 다양한 데이터 유형을 처리할 수 있다

2 비교적 낮은 Accuracy, Recall, Precision 값

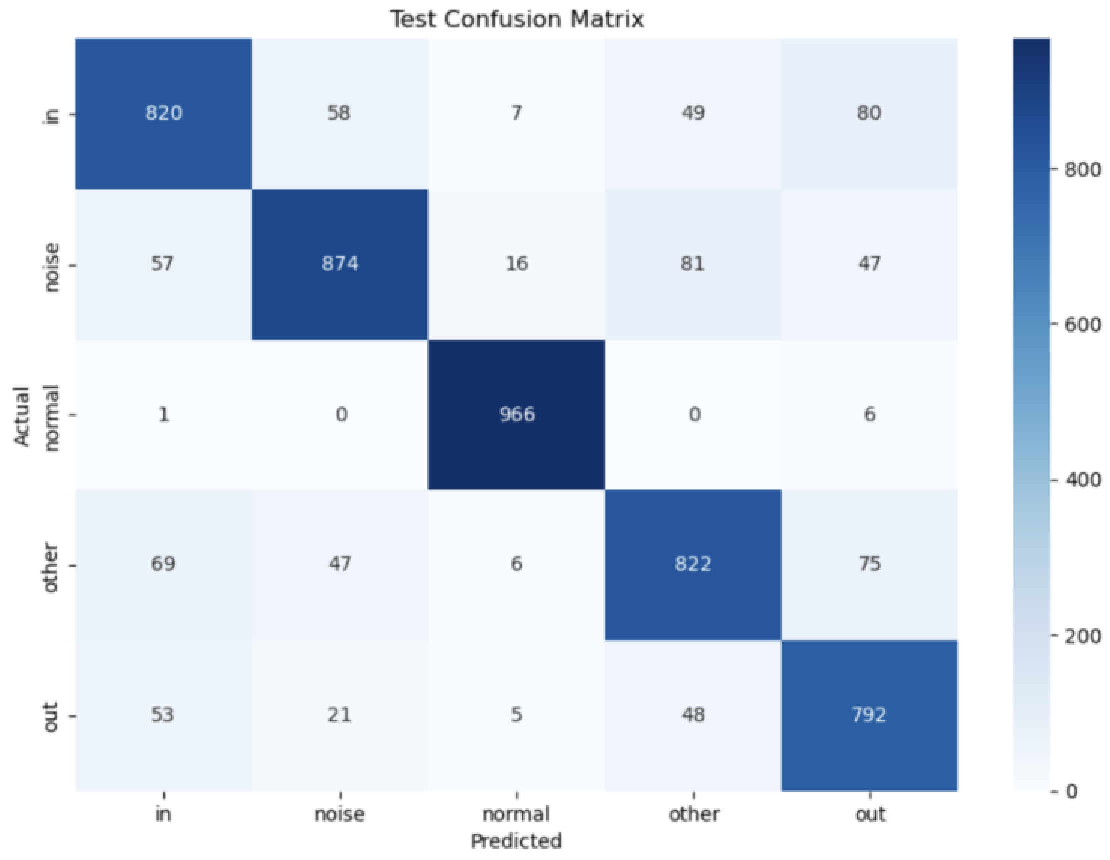
→ 복잡한 데이터셋과, 명확하지 않은 경계로 수많은 노드가 생성되어 과적합 발생
→ 만약 데이터셋에 클래스 불균형이 존재했다면, 빈도가 높은 클래스로 학습이 치우치게 되어 성능이 저하되었을 것

[3. Removing Max Values]

Accuracy and Recall of "in" and "out" Classes

- Max Values 를 사용할 때 더 감소한 Accuracy와 in, out의 recall 값
- False Negative 를 줄이는 것의 중요성 → in, out 클래스의 Recall 값에 집중
- Max 값이 모델 과적합, 또는 차원의 저주에 영향을 미칠 가능성

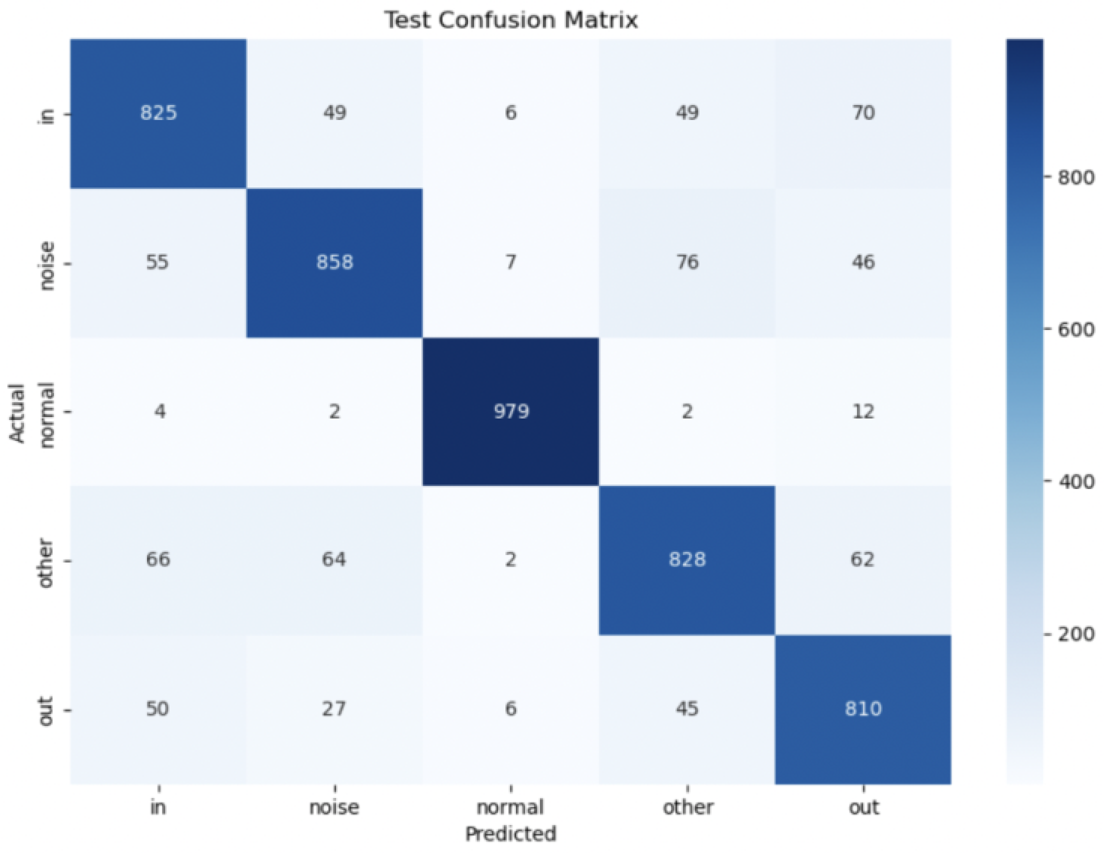
WITH MAX VALUES



Accuracy: 0.8548

Recall:
{'in': 0.809,
'noise': 0.813,
'normal': 0.993,
'other': 0.807,
'out': 0.862}

WITHOUT MAX VALUES



Accuracy: 0.86

Recall:
{'in': 0.826,
'noise': 0.823,
'normal': 0.980,
'other': 0.810,
'out': 0.864}

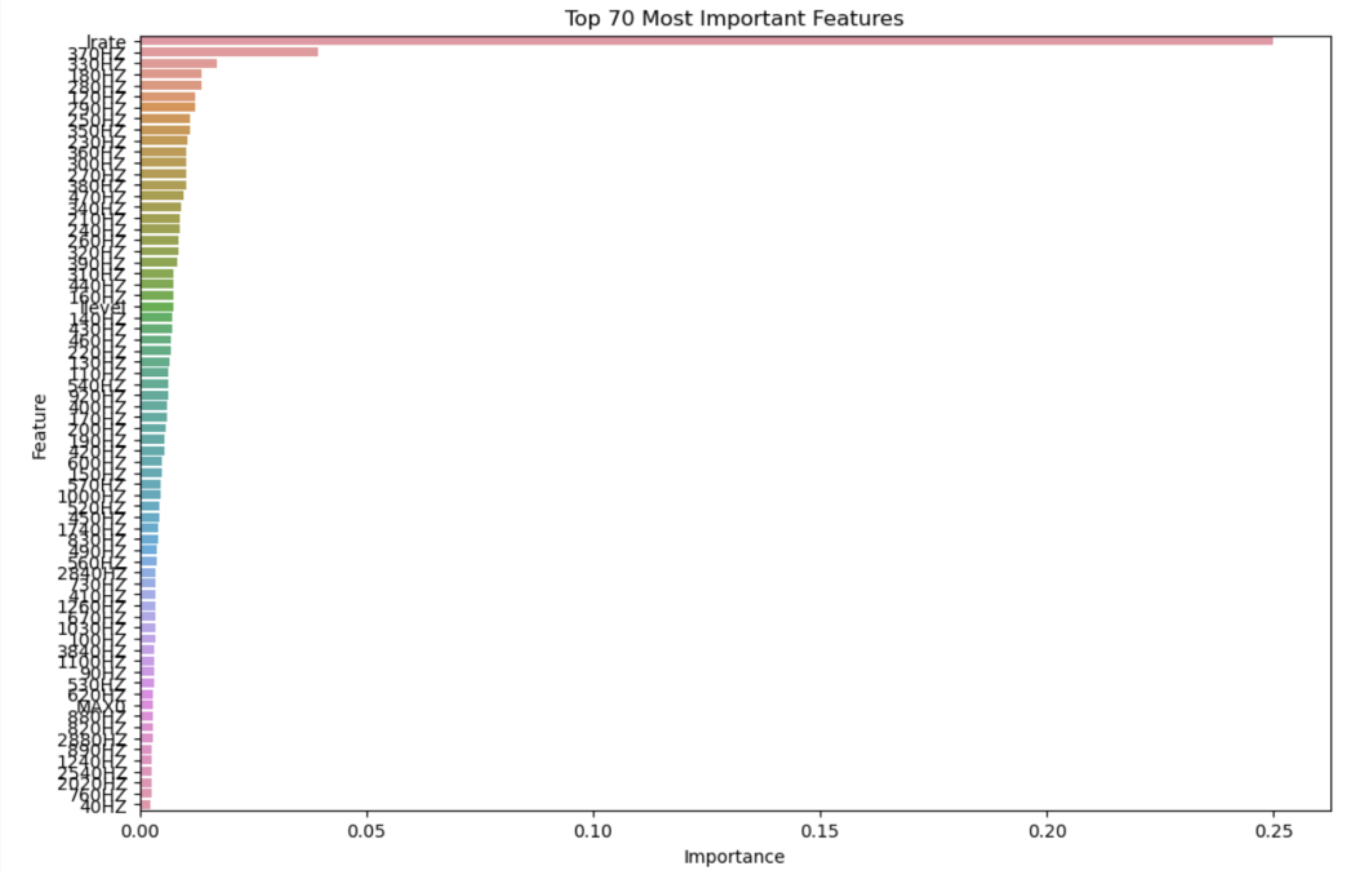
[4. Important Features - Decision Tree]

01 Using Decision Tree

- KNN의 차원의 저주
- 훈련 데이터로 Decision Tree 알고리즘으로 모델을 훈련
- 그 중 70개를 골라 KNN 알고리즘에 사용
    - 70개가 가장 좋은 성능을 내었다

02 Results

- 중요한 70개의 Features 만 사용했을 때 향상된 모델의 정확도와 in, out의 Recall 값



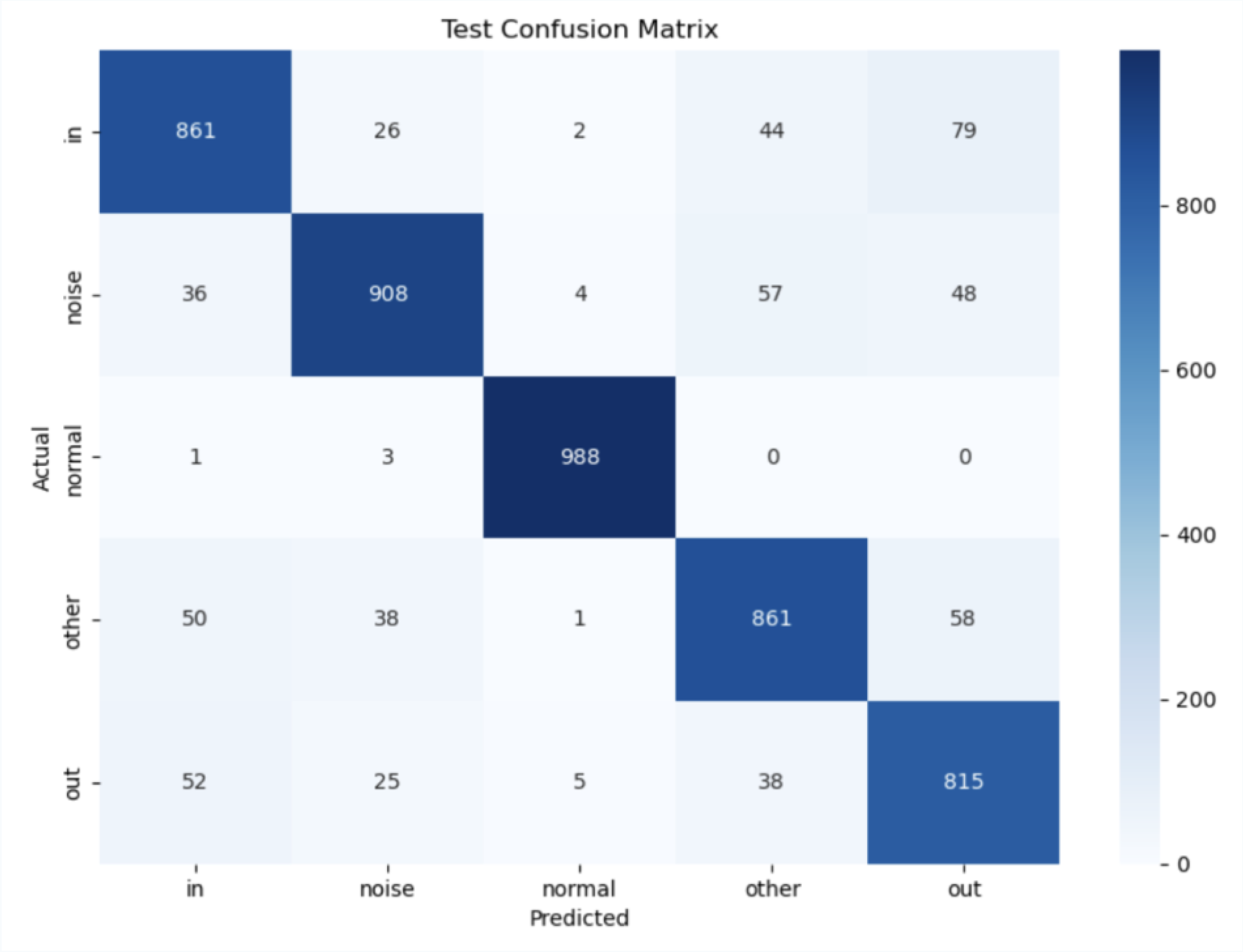
[4. Important Features - Decision Tree]

01 Using Decision Tree

- KNN의 차원의 저주
- 훈련 데이터로 Decision Tree 알고리즘으로 모델을 훈련
- 그 중 70개를 골라 KNN 알고리즘에 사용
    - 70개가 가장 좋은 성능을 내었다

02 Results

- 중요한 70개의 Features 만 사용했을 때 향상된 모델의 정확도와 in, out의 Recall 값




```
Test Accuracy: 0.8866

Precision:'in': 0.861, 'noise': 0.908, 'normal': 0.988, 'other': 0.861, 'out': 0.815

Recall:'in': 0.851, 'noise': 0.862, 'normal': 0.996, 'other': 0.854, 'out': 0.872
```

5. KNN Model & Confusion Matrix

<table>
<tr><th></th><th>Code</th></tr>
<tr><td>Splitting Data</td><td><pre># Splitting features and target in the training dataset
X_train = train_data.loc[:, 'lrate':'MAX0'].drop(columns=['leaktype'])
y_train = train_data['leaktype']</pre></td></tr>
<tr><td>Scaling</td><td><pre># Standardizing the training data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)</pre></td></tr>
<tr><td>Important Features</td><td><pre># Implementing Decision Tree classification to identify top 20 features
decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train_scaled, y_train)

# Identifying the top 70 most important features
feature_importances = decision_tree.feature_importances_
features = X_train.columns

# Create a DataFrame for feature importances
feature_importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': feature_importances
})

# Sort the DataFrame by importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Select the top 20 features
top_70_features = feature_importance_df.head(70)['Feature']
# Create new datasets with the top 20 features
X_train_top_70 = X_train[top_70_features]</pre></td></tr>
</table>

5. KNN Model & Confusion Matrix

Confusion Matrix

```python
def confusion_matrix_metrics(y_true, y_pred):
    unique_classes = np.unique(y_true)
    cm = {cls: {cls_: 0 for cls_ in unique_classes} for cls in unique_classes}

    for true, pred in zip(y_true, y_pred):
        cm[true][pred] += 1

    accuracy = np.sum([cm[cls][cls] for cls in unique_classes]) / len(y_true)

    precision = {cls: cm[cls][cls] / sum(cm[cls].values()) if sum(cm[cls].values()) > 0 else 0 for cls in unique_classes}
    recall = {cls: cm[cls][cls] / sum([cm[cls_][cls] for cls_ in unique_classes]) if sum([cm[cls_][cls] for cls_ in
unique_classes]) > 0 else 0 for cls in unique_classes}

    return cm, accuracy, precision, recall
```

Plotting Confusion Matrix

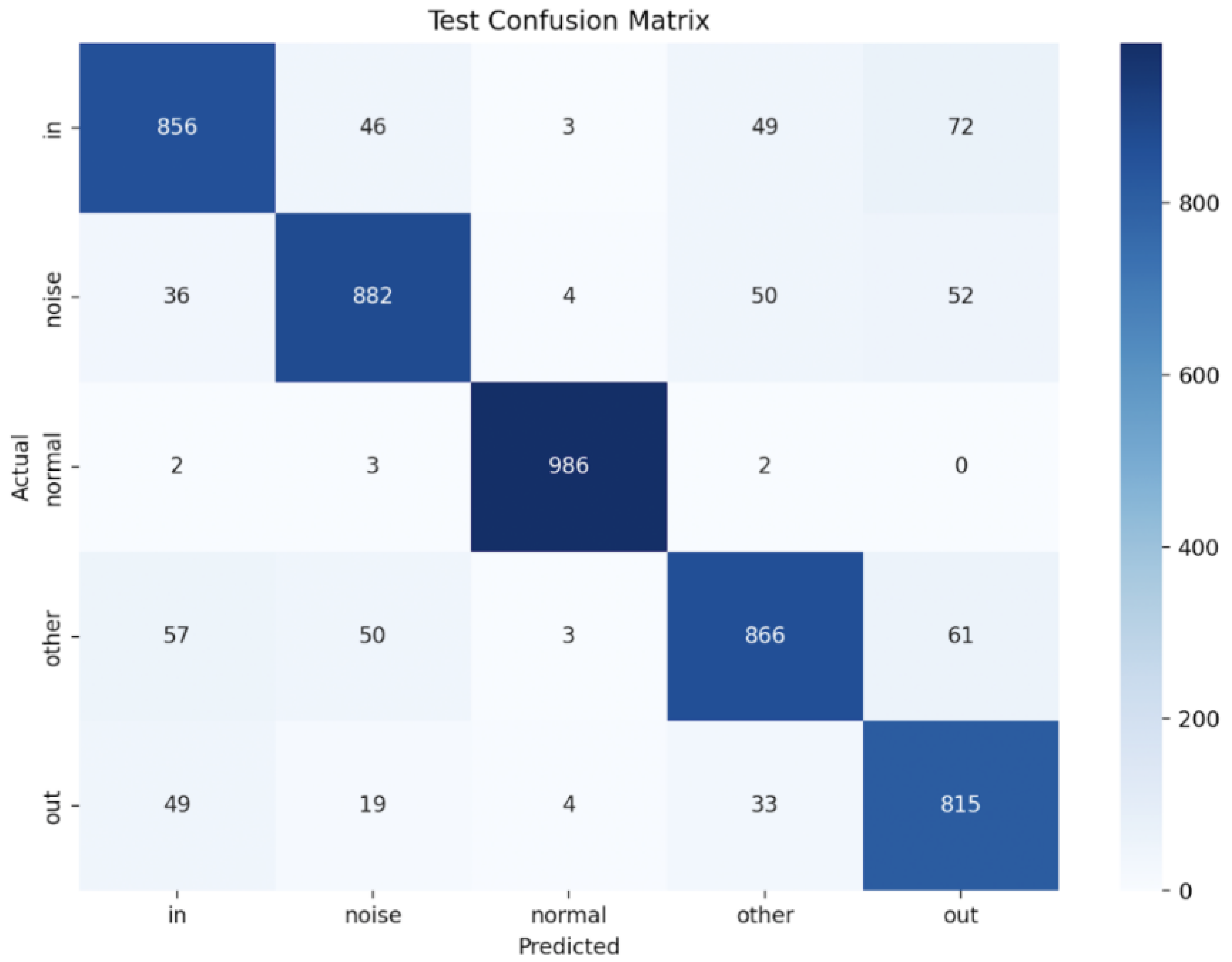
```python
def plot_confusion_matrix(cm, title):
    cm_df = pd.DataFrame(cm)
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm_df, annot=True, fmt='d', cmap='Blues')
    plt.title(f'{title} Confusion Matrix')
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()
```

Making Predictions

```python
    test_file_path = 'Classification_testing_data.csv'  # Update this path
    test_data = pd.read_csv(test_file_path)
    X_test_new = test_data.loc[:,'lrate':'MAX0'].drop(columns=['leaktype'])
    X_test_new=X_test_new[top_70_features]
    y_test_new = test_data['leaktype']
    X_test_new_scaled = scaler.transform(X_test_new)

    # Making predictions on the new test dataset
    y_pred_new = knn.predict(X_test_new_scaled)
```

[6. Results and Analysis]



Class "in"		Predicted	
		Positive	Negative
Actual	Positive	856	170
	Negative	144	3830

Accuracy =	0.9372
Precision =	0.856
Recall =	0.83430799

Class "out"		Predicted	
		Positive	Negative
Actual	Positive	815	105
	Negative	185	3895

Accuracy =	0.942
Precision =	0.815
Recall =	0.88586957

Class "noise"		Predicted	
		Positive	Negative
Actual	Positive	882	142
	Negative	118	3858

Accuracy =	0.948
Precision =	0.882
Recall =	0.86132813

Class "normal"		Predicted	
		Positive	Negative
Actual	Positive	986	7
	Negative	14	3993

Accuracy =	0.9958
Precision =	0.986
Recall =	0.99295065

Class "other"		Predicted	
		Positive	Negative
Actual	Positive	866	171
	Negative	134	3829

Accuracy =	0.939
Precision =	0.866
Recall =	0.83510125

CONFUSION MATRICES FOR EACH CLASS

- Focus: most “in” and “out” classes are predicted as leakages – indicated by the recall values of “in” and “out” classes
- Relatively high accuracy, precision, and recall scores in the “normal” class, but relatively lower scores in the “in” class, especially in recall
- Generally high recall scores, but mispredictions cannot just be overlooked

[Possible Future Directions]

Adjusting Model

Work on adjusting the classification model so that we would have lower mispredictions on actual leakage classes, despite risking having higher mispredictions on non-leakage classes.

XGBoost

- Decision tree ensemble learning algorithm
- Combines several machine learning algorithms to obtain a better model
- Choosing one algorithm over all others was not good enough

References

[1]
OpenAI. (2024). ChatGPT (June 7 version) [Large language model]. OpenAI.

[2]
National Information Society Agency. (2020). Water pipe leak detection dataset. AI Hub. Updated June 2021. Retrieved June 7, 2024, from https://aihub.or.kr/aihubdata/data/view.do?currMenu=115&topMenu=100&aihubDataSe=realm&dataSetSn=138