SMOKE:

Speed, accurate sMoke detection comparison based On Knn, dEcision tree, svm

201915601 양찬우

목차

1. 초록

연기 감지기는 화재의 지표인 연기를 감지하는 장치이다. 연기 감지기는 화재로부터 생명과 재산을 보호하기 위한 중요한 장비로 사용된다. 단 한번의 오작동은 큰 피해를 불러일으킬 수 있으므로, 연기 감지기에게는 높은 탐지 정확성이 요구된다. 연기 탐지기에 AI를 적용하는 일은 연기 탐지기의 정확성을 개선하는 것으로 볼 수 있다. AI는 데이터를 기반으로 작동하며, 데이터가 축적될수록 task에 대한 높은 성능을 보여줄 수 있기 때문이다. 빠른 탐지 속도 또한 연기 탐지기에게 중요한 능력이다. 따라서, 본 보고서에서는 deep learning 모델에 비해 상대적으로 작은 크기를 가진 machine learning 모델이 더 빠른 탐지 속도를 보여줄 수 있다고 가정하고, machine learning 모델을 활용하여 연기 탐지에 대한 binary classification task 및 실험을 진행한다. 실험은 여러 machine learning 모델들의 성능 비교 및 parameter 변화에 따른 성능 변화 등을 보여준다. 이러한 실험들을 통해 어떠한 machine learning 모델이 AI 연기 탐지기에 적용될 수 있는 적절한 모델인지 확인한다. 결과적으로 Decision tree 모델을 AI 연기 탐지기로서 사용하는 것이 가장 적절한 선택인 것으로 드러난다. 더불어, 실험에 사용된 모든 machine learning 모델들이 준수한 성능을 보이면서, 무조건적인 deeplearning 모델의 사용은 지양되어져야 함이 강조된다.

2. 응용 문제 정의

화재는 불에 의한 재난으로, 특히 연기는 화재 초기에 발견되는 주요 신호 중 하나이다. 연기 감지기는 화재의 지표인 연기를 감지하고, 수많은 생명과 재산을 앗아갈 수 있는 화재에 신속히 대응할 수 있도록 돕는 아주 중요한 역할을 한다. 단 한번의 오작동은 큰 피해를 불러일으킬 수 있으므로, 연기 탐지 시스템에게는 높은 탐지 정확성이 요구된다.
기존의 연기 탐지 시스템은 기본적으로 감지 성능에 한계가 있으며, 이로 인해 신속하고 효과적인 대응이 어려워질 수 있다. 이러한 한계와 도전 과제들은 새로운 접근 방식의 필요성을 강조한다.

2.1 기존 연기 탐지 시스템의 한계

현재의 대부분의 연기 탐지 시스템은 전통적인 규칙 기반 접근 방식을 사용하며, 이는 특정한 연기 패턴을 기반으로 동작한다. 그러나 이러한 시스템은 다양한 환경에서 발생하는 연기의 형태와 밀접한 관련이 있어, 변화무쌍한 조건에서는 제대로 동작하지 못하는 경우가 발생할 수 있다. 또한, 거짓 양성이 발생하는 등의 한계도 존재한다.

2.2 AI 기반 연기 탐지 시스템의 필요성과 잠재적 이점

AI 기술의 발전은 연기 탐지 분야에서 새로운 희망을 제시한다. 기계 학습 및 딥러닝 알고리즘을 활용한 AI 기반 연기 탐지기는 환경의 변화에 뛰어난 적응성을 보이며, 기존의 연기 탐지기에 비해 훨씬 정확하고 빠른 연기 감지가 가능할 것으로 기대된다. AI는 데이터를 기반으로 하여, 데이터가 축적될수록 AI를 적용하지 않는 시스템에 비해 더 높은 성능을 보일 수 있음이 수많은 실험들을 통해 증명되어왔기 때문이다. 이러한 기술은 기존 시스템의 한계를 극복하고, 화재 초기에 조치를 취하여 생명과 재산을 보호해줄 적절한 도구로 작용할 것으로 보인다.

따라서, 본 보고서에서는 AI를 활용한 연기 감지기를 모델링하기 위한 binary classification task와 실험을 진행하고자 한다.

3. 모델

AI 연기 탐지 시스템을 구현하는 데에 있어서 어떤 모델을 사용할 것인지에 대한 선택은 매우 중요하다. 모델 선택은 곧 연기 감지기의 성능과 효율성에 큰 영향을 미치기 때문이다.
AI 연기 탐지 시스템을 위한 모델 선택을 위해 먼저 고려해야 하는 것은 deep learning 모델과 machine learning 모델 중 어떤 모델을 선택할 것인가에 대한 문제이다. 본 보고서에서는 machine learning 모델을 선택한다. 그 이유는 아래의 2가지가 존재한다.

(1) 빠른 판단 속도
Deep learning 모델은 학습 및 추론에 많은 계산 리소스가 필요하며, machine learning 모델에 비해 많은 layer를 거쳐야만 추론 결과를 얻을 수 있다. 그 결과로, 빠른 판단 속도가 필요한 연기 감지기임에도 판단 결과를 나타내는 데에 있어 높은 지연 시간이 발생할 수 있다. 반면에, machine learning 모델은 상대적으로 적은 계산 리소스를 필요로 하고, 판단 결과를 얻기 위해 deep learning 모델과 같이 수많은 layer를 거칠 필요가 없으므로, 더 빠른 판단 속도를 제공할 수 있을 것으로 예상한다.

(2) 경량화
Deep learning 모델은 대부분 대용량의 파라미터를 필요로 한다. 이는 관리와 업데이트가 machine learning 모델에 비해 어려워질 수 있음을 의미한다. Machine learning 모델은 상대적으로 작은 크기를 가지면서도 효과적인 학습이 가능하여 연기 감지기의 경량화에 큰 도움을 줄 것으로 예상한다.

3.1 모델 선정

그렇다면 machine learning 모델 중 어떤 모델을 사용하는 것이 좋을 것인가?
AI 연기 탐지 시스템에게 요구되는 성능은 탐지 정확도와 속도이다. 더불어, 사용할 모델은 binary classification task를 진행할 수 있는 모델이어야 한다. 이를 반영하여 본 보고서에서는 아래 3가지 모델들을 실험에 사용할 모델로서 선정한다.

(1) k-Nearest Neighbors(kNN)
kNN은 지도 학습 알고리즘 중 하나로, 새로운 데이터 포인트 주변에 있는 가까운 k개 이웃의 클래스들을 기준으로 새로운 데이터 포인트의 클래스를 결정하는 알고리즘이다. 새로운 데이터 포인트 주변의 가까운 k개의 이웃 데이터들을 알아내기 위해서는 데이터 포인트들 간 거리 계산이 필요하다. 이때의 거리 측정 방법은 매우 다양하며, 본 보고서에서 사용되는 kNN 알고리즘은 데이터 포인트들 간 거리 측정을 위해 euclidean distance를 사용한다.

kNN은 매우 간단하고 직관적인 모델이며, 학습 시간이 상수 시간에 수렴한다는 특징을 가지고 있다. 또한, 기본적으로 높은 판단 정확도를 보일 수 있는 알고리즘이다.
하지만, 추론 시에 모든 training 데이터와의 거리를 계산해야 하므로 O(MN)의 시간 복잡도를 가지고 있다. 여기에서 M은 feature의 개수, N은 example의 개수를 의미한다. 이는 데이터셋이 커지면 커질수록 필요한 연산 시간이 매우 길어질 수 있음을 의미한다. 또한, 데이터 feature에 noise가 존재할 때 판단 정확도가 매우 떨어질 수 있다는 단점이 존재한다. 그러므로 kNN은 간단하면서도 효과적인 모델이지만, 차원이 높고 크기가 큰 데이터셋을 사용할 때에는 적합하지 않을 수 있다. 차원이 높을 때에도 kNN이 적합하지 않을 수 있는 이유는 차원의 저주가 발생할 수 있기 때문이다.

kNN에서 사용되는 parameter는 k로, 추론 시에 새로운 데이터 포인트의 클래스를 결정할 때 몇 개의 이웃 데이터 포인트를 보도록 할 것인지를 나타낸다.

kNN의 학습은 데이터의 저장을 의미한다. kNN의 학습을 위해서는 kd-tree와 같은 자료구조가 이용될 수도 있다. 본 보고서에서 사용되는 kNN 알고리즘은 자료구조를 사용하지 않는 학습을 진행한다.

본 보고서에서 kNN을 선정한 이유는 kNN은 학습이 단순하여 연기 데이터의 축적이 용이하기 때문에, 많은 데이터를 기반으로 정확한 판단을 내릴 수 있도록 도울 것이라고 생각했고, parameter 조정에 따라 빠른 판단 속도 또한 보일 수 있는 알고리즘이 될 수 있을 것이라고 판단하였기 때문이다.

(2) Support Vector Machine(SVM)
SVM 또한 지도 학습 알고리즘으로, 데이터를 고차원 공간으로 매핑하여 최적의 결정 경계를 찾는 방식으로 동작한다. Kernel을 이용하면 명시적으로 고차원 공간으로 매핑하여 product를 하지 않고도 결정 경계를 찾을 수 있다. 이때 최적의 결정 경계는 separator와 데이터 포인트 사이의 width를 의미하는 margin을 가장 크게 만드는 결정 경계이다. 엄밀하게 정의하자면, margin은 separator와 soft vector 사이의 width를 의미한다. Soft vector는 separator와 가장 가까이 위치하는 데이터 포인트를 의미하며, 클래스마다 1개의 soft vector가 존재하게 된다.

SVM은 classification task에 있어 훌륭한 성능과 일반화 능력을 보인다. kNN과는 달리 noise가 존재하는 데이터셋에도 robust한 성능을 보인다는 특징을 갖는다. 또한 kernel trick을 이용하여 고차원 매핑 없이 효율적인 학습을 진행할 수도 있다.

하지만, 여전히 연산 비용이 커서 학습과 추론 시에 속도가 오래 걸릴 수 있다는 단점이 존재한다.

SVM에서 사용되는 parameter는 C로, slack variable을 통해 얼마나 유연한 결정 경계를 만들 것인지를 조절시킨다.

본 보고서에서는 SVM이 높은 탐지 정확도를 보여줄 수 있는 알고리즘이면서도 변화무쌍한 환경에서도 robust한 판단을 내릴 수 있는 알고리즘이라고 판단하여, 실험에 사용할 모델로 선정했다.
실험 시, linear와 kernel svm을 사용하여 실험한다.

(3) Decision tree
Decision tree 또한 지도 학습 알고리즘으로, 데이터의 impurity를 최소화하는 혹은 information gain을 최대화하는 feature들을 기준으로 데이터를 분할하여 트리 구조로 표현하여 추론을 진행한다. Decision tree도 kNN과 마찬가지로 직관적이라는 특징을 가지고 있다. 해석 가능한 모델이라는 점도 특징이다.

Decision tree의 inductive bias는 추론이 모든 feature를 보지 않고 진행된다는 것이다. 그렇기 때문에 decision tree는 매우 빠른 판단 속도를 보여줄 수 있으며 noised feature가 존재한다고 해도 robust한 추론을 보여줄 수 있다.

본 보고서에서는 decision tree가 매우 빠른 판단 속도를 보여줄 수 있는 알고리즘이라는 점에 집중하여 실험에 사용할 모델로 선정했다.

3.2 구현 모델
Numpy로 구현한 모델은 kNN이며 아래와 같이 구현되었다.
구현부는 학습에 필요한 부분과 추론에 필요한 부분으로 나뉜다.

(1) 학습
kNN의 학습은 데이터를 저장하는 것과 같다. 이때, 저장 방식은 kd-tree와 같은 자료구조를 이용할 수 있다. 구현할 때에는 자료구조를 사용하지 않고 구현했다.

```
def fit(self, X, y):
    # just Storing
    self.X = X
    self.y = y

    self._fitted = True
```

(2) 추론

kNN의 추론을 위해서는 새로운 데이터 포인트와 모든 학습 데이터 포인트들 간의 거리 계산이 필요하다. 거리 계산 또한 다양한 선택지가 존재하지만, 구현 시에는 euclidean distance를 사용하여 구현했다.

```python
def euclidean_distance(self, data_x, data_y):
    return np.sqrt(((data_x[:, np.newaxis, :] - data_y) ** 2).sum(axis=2))

def predict(self, test_X):
    if self._fitted is not True:
        raise ValueError("The model should be fitted by training dataset before prediction.")

    probability = self.predict_proba(test_X)

    # predict_proba에서 온 proba로 argmax 해서 making predicted label
    predicted_y = probability.argmax(axis=-1) # (test_X.shape[0], 1)

    return predicted_y

def predict_proba(self, test_X):
    if self._fitted is not True:
        raise ValueError("The model should be fitted by training dataset before prediction.")

    self.distances = self.euclidean_distance(test_X, self.X)

    # 거리 값이 가장 작은 n개의 train 데이터 index를 각 테스트 데이터마다 구하기
    indices = np.argpartition(self.distances, self.n_neighbors, axis=1)[:, :self.n_neighbors]

    # 각 테스트 데이터의 train 데이터 index를 train_y와 mapping 해서 label 빼기
    predicted_class = np.empty_like(indices)
    for i in range(predicted_class.shape[0]):
        pred_class = []
        for j in range(predicted_class.shape[1]):
            pred_class.append(self.y[indices[i][j]])
        predicted_class[i] = np.array(pred_class) # (test_X.shape[0], n_neighbors)

    # 각 테스트 데이터의 클래스 별 proba 연산
    probability = np.empty((test_X.shape[0], 2)) # (test_X.shape[0], num_classes)
    for i in range(probability.shape[0]):
        _predicted_class = predicted_class[i].tolist()
        neg_prob, pos_prob = _predicted_class.count(0) / self.n_neighbors, _predicted_class.count(1) / self.n_neighbors

        probability[i] = np.array([neg_prob, pos_prob])

    return probability
```

kNN을 제외한 SVM과 decision tree 알고리즘은 scikit-learn 라이브러리를 이용하여 구현했다.

4. 데이터

4.1 데이터 분석

본 보고서에서의 실험 진행을 위해 사용하는 데이터셋은 kaggle에 등록된 데이터셋인 “Smoke Detection Dataset”이다. 이 데이터셋에 포함된 데이터의 총 개수는 62630개이며, 0과 1로 구분된 binary label을 갖고 있다. 여기에서 0은 화재가 발생하지 않은 상황이며, 1은 화재가 발생한 상황을 의미한다. 데이터 label에 따른 데이터 개수는 아래 표와 같다.

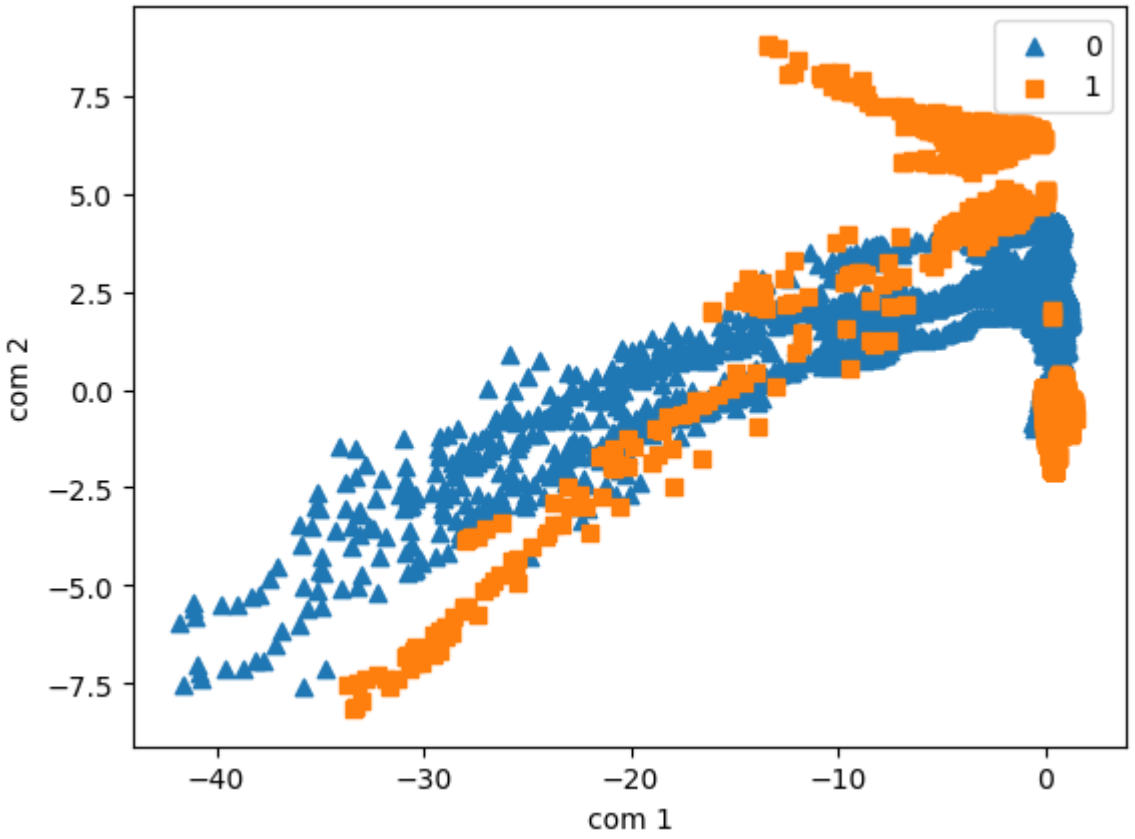
Label 0	Label 1	Total
17873	44757	62630

Label을 제외하고, example 1개가 갖고 있는 feature 종류의 개수는 14개이며, 종류는 아래와 같다.

- UTC
- Temperature[C]
- Humidity[%]
- TVOC[ppb]
- eCO2[ppm]
- Raw H2
- Raw Ethanol
- Pressure[hPa]
- PM 1.0
- PM 2.5
- NC 0.5
- NC 1.0
- NC 2.5
- CNT

모든 feature 값은 int와 float을 통해 표현되어 있다.

이러한 데이터를 normalizing 시키고 pca를 통해 2차원으로 표현한 모습은 아래와 같다.



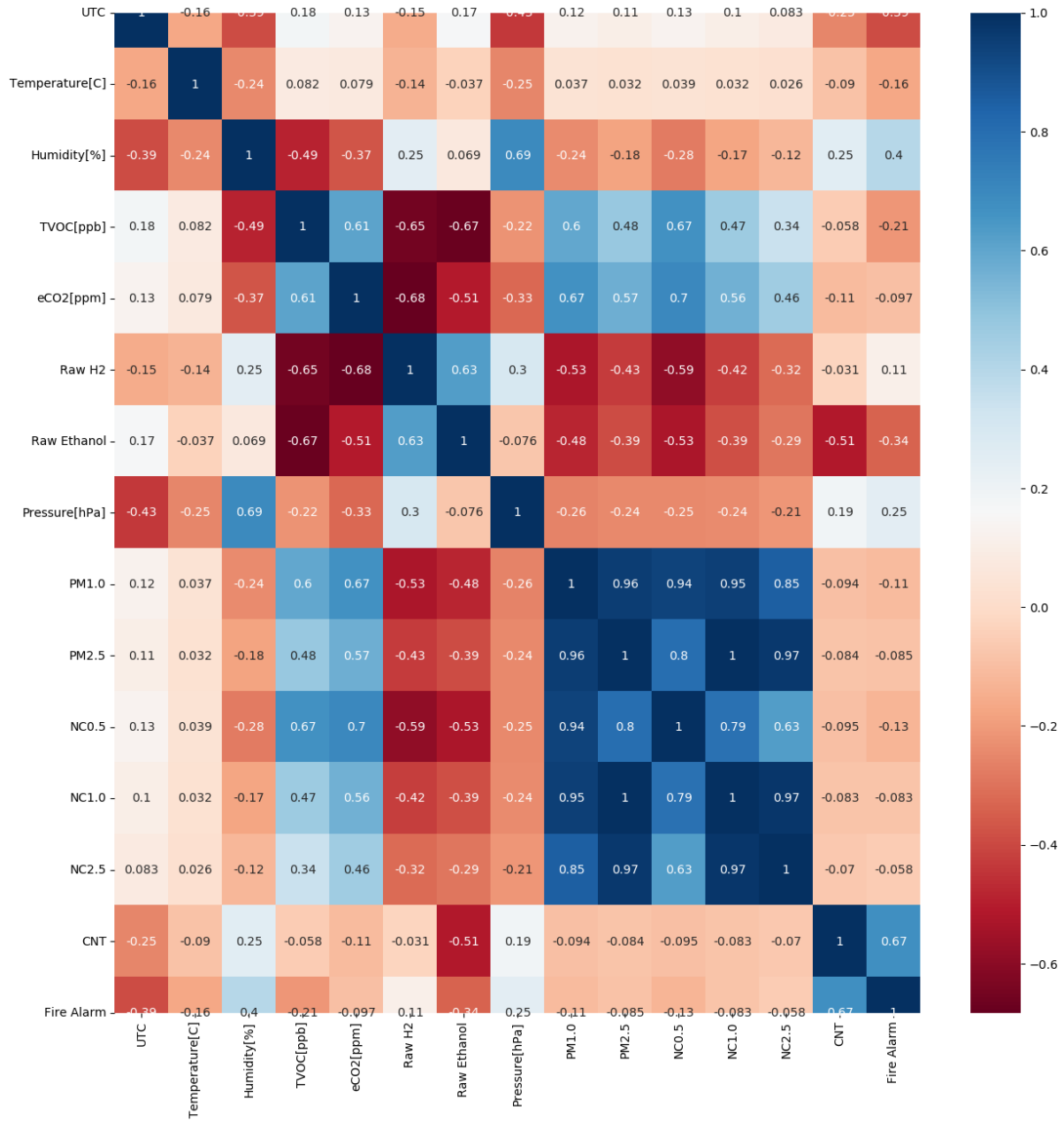
PCA를 통해 데이터를 시각화시켰을 때, 관찰되는 데이터의 분포는 label이 0이든 1이든 간에 비슷한 분포를 띄는 것으로 확인된다. 다만, label 0인 데이터들은 표준편차가 label 1인 데이터 포인트들에 비해 큰 것으로 확인된다. Label이 0인 데이터와 label이 1인 데이터들 간의 상관관계는 데이터 분포만으로는 확인하기 어려운 것으로 보인다.

4.2 데이터 전처리

데이터 전처리를 위해 feature와 label 간의 상관관계를 분석하고, outlier 데이터가 존재하는지 확인한다. 그리고 normalizing을 통해 feature 값들이 zero-mean, unit variance를 갖도록 한다.

4.2.1 상관관계 분석

Feature와 label 간의 상관관계를 분석하기 위해, seaborn 라이브러리를 활용하여 feature들의 heatmap을 시각화시켰으며 그 결과는 아래와 같다.



그리고 feature들과 label 간의 상관관계 값을 나타내보면 아래 표와 같다.

UTC	-0.39
Temperature[C]	-0.16
Humidity[%]	0.4
TVOC[ppb]	-0.2

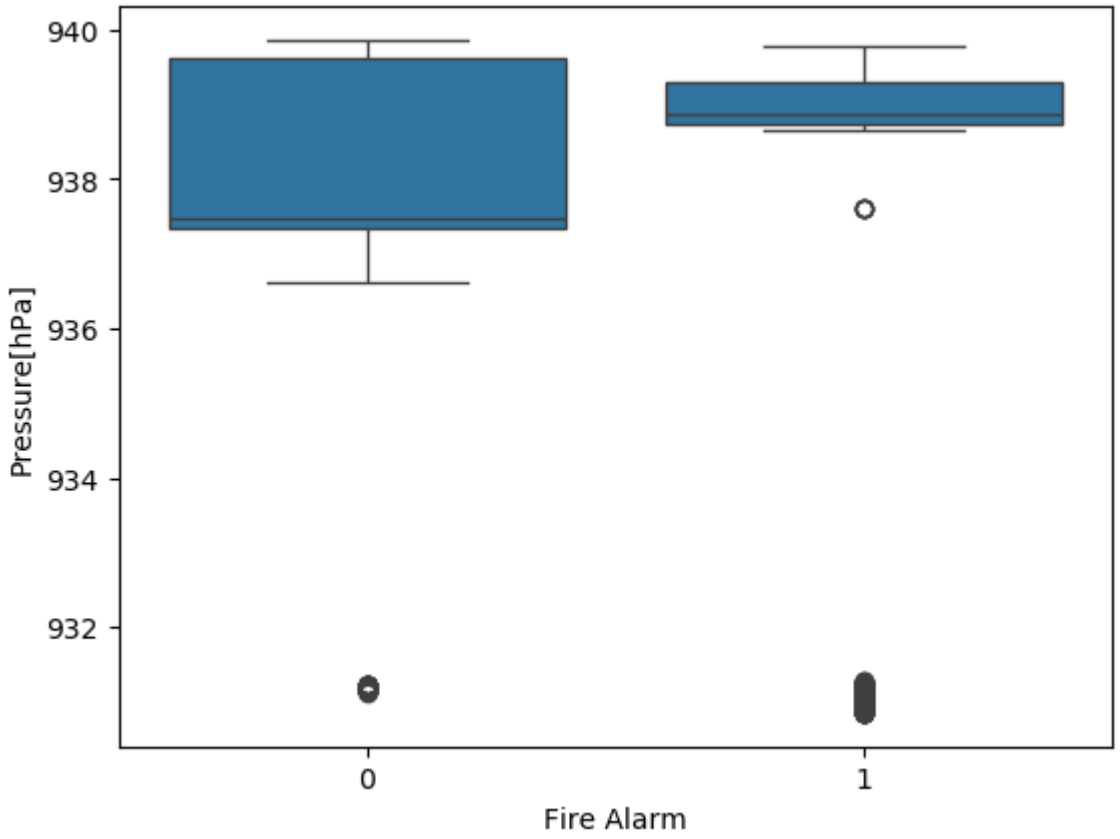
eCO2[ppm]	-0.1
Raw H2	0.1
Raw Ethanol	-0.34
Pressure[hPa]	0.25
PM 1.0	-0.11
PM 2.5	-0.1
NC 0.5	-0.13
NC 1.0	-0.1
NC 2.5	-0.1
CNT	0.67

상관관계 분석 결과를 볼 때, CNT feature를 포함한 총 4가지 feature들만이 label과 양의 상관관계를 갖고 있는 것으로 확인된다.
따라서, 음의 상관관계를 갖고 있는 나머지 10가지 feature들은 모두 제거하기로 결정하였다.
음의 상관관계인 feature들을 제거하는 것은 모델의 연산 속도를 높이면서도, 추론 정확도를 높이는 데에 기여할 것이라고 생각했기 때문이다.

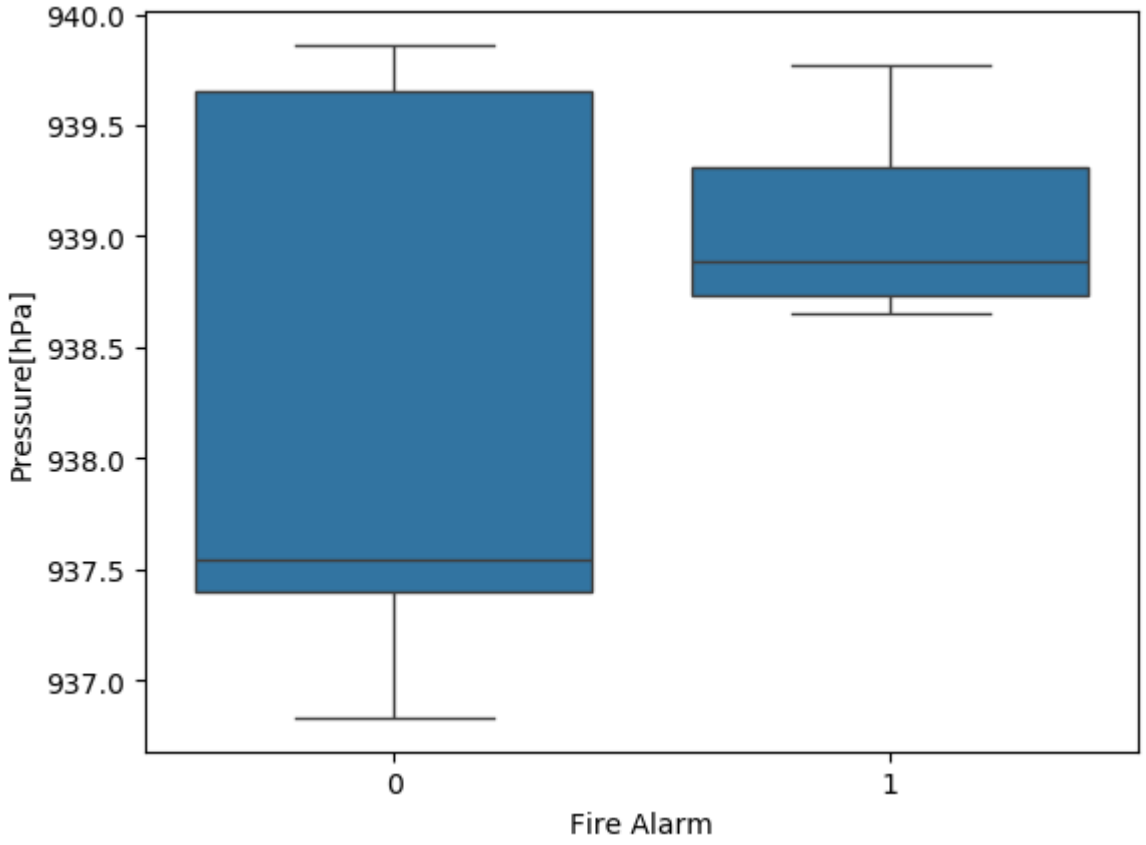
4.2.2 Outlier 분석

Outlier는 일반적인 데이터 집합에서 크게 벗어난 값을 가진 데이터를 말하며 특이값 혹은 극단값으로 불린다. Outlier는 대부분의 데이터 포인트들이 가지고 있는 특성과는 다른 특성을 가지고 있기 때문에 데이터의 일반적인 분포나 패턴을 왜곡시킬 수 있다.
이러한 특징 때문에 outlier는 모델의 부정확한 예측을 발생시킬 수 있으며, 특히 outlier에 민감한 알고리즘의 경우 성능의 하락을 초래할 수 있다.
따라서, label과 양의 상관관계를 갖고 있는 4개의 feature들이 outlier를 포함하고 있는지 확인한 다음, outlier 제거를 진행했다. Outlier는 CNT feature를 제외한 3개 feature들에서 확인되었으며, 그 3개 feature들이 갖고 있던 outlier를 제거했다.

아래는 outlier가 확인된 3개 feature들 중 하나인 Pressure[hPa] feature가 가지고 있던 outlier를 boxplot을 통해 나타낸 것이다.



그리고 확인된 outlier를 제거한 후의 모습은 아래와 같다.



4.2.3 Normalizing

데이터의 feature 값들의 scale 차이가 커지게 되면, 특히 kNN과 같이 데이터 포인트들의 거리를 측정해야 하는 경우 잘못된 거리 값이 측정될 수 있으며 이는 모델의 잘못된 예측을 초래시킬 수 있다.
따라서, normalizing을 진행하여 데이터의 값들을 zero-mean, unit variance로 scaling 한다.

```
def normalizing(train_data, val_data, test_data):
    mu = np.mean(train_data, axis=0)
    sigma = np.std(train_data, axis=0)

    # normalizing (zero-mean)
    train_data = (train_data - mu) / sigma
    val_data = (val_data - mu) / sigma
    test_data = (test_data - mu) / sigma

    return train_data.to_numpy(), val_data.to_numpy(), test_data.to_numpy()
```

4.3 데이터 분리

본 데이터셋은 training, evaluation, test를 위해 데이터가 분리되어 있지 않은 데이터셋이다.
그래서 모델 학습을 위한 training data, 모델 평가 및 hyperparameter 조절을 위한 evaluation data 그리고 모델 성능을 보여주는 test data으로 분리해주어야 했다. 각 용도를 위해 분리한 모습은 아래와 같다.
총 데이터의 개수가 58030개가 된 이유는 outlier 제거가 진행되었기 때문이다.

용도	개수	비율
Training	43523	75%
Evaluation	8705	15%
Test	5802	10%

위와 같이 training 데이터의 개수를 75%의 비율로 설정한 이유는, 최대한 많은 데이터를 학습에 사용함으로써 모델이 일반화 능력을 갖도록 만들고자 했기 때문이다. 그리고 나머지 25%를 차지하는 evaluation과 test 데이터는 비슷한 비율을 가지면서도 evaluation이 좀 더 많은 비율을 갖도록 했다. 그 이유는 evaluation 데이터를 통해 hyperparameter를 설정할 때, 최대한 일반화 되면서도 성능을 최고로 높여줄 수 있는 hyperparameter 값을 찾을 수 있도록 하길 원했기 때문이다.

5. 실험

5.1 평가 지표

세 가지 모델의 연기 감지기 시스템에 대한 적합도를 판단하기 위해 2가지 평가 지표를 사용한다.
2가지 평가 지표는 accuracy와 모델이 예측을 도출하기까지 걸리는 시간이다. 연기가 발생했을 때, 가장 정확하고 가장 빠르게 화재 판단을 내릴 수 있는 모델은 어떤 모델인지 평가할 수 있도록 이와 같은 2가지 평가 지표를 사용하기로 결정했다.
Accuracy는 직접 구현한 confusion matrix를 활용하여 평가를 진행하였으며, 속도의 경우는 line_profiler 라이브러리를 활용하여 평가를 진행하였다.

5.2 Hyperparameter

3가지 모델 중에서 hyperparameter를 고려한 모델은 kNN뿐이다. 다른 모델 또한 hyperparameter가 존재하지만 결과적으로 나타난 성능을 확인해 보았을 때, 초기 설정값에서 바로 최고 성능이 확인되어 hyperparameter를 수정하지 않아도 상관 없다는 판단을 내렸기 때문이다.
다만, 성능 평가 전에 kNN의 hyperparameter는 validation dataset을 이용한 cross validation을 통해서 설정되었다. Hyperparameter인 k = 1부터 101까지 k 값을 조정하여, validation dataset에 대해 최고 성능을 낸 k 값은 1로 나타났기 때문에, 성능 실험 시 사용할 hyperparameter k는 1로 설정하기로 결정했다. 또한, 최고 성능 확인과는 별개로 hyperparameter 변화에 따른 kNN의 성능 변화를 실험해보기 위해 hyperparameter k를 40000으로 설정하였을 때의 성능을 추가적으로 확인했다.

5.3 실험

5.3.1 Accuracy

아래 표는 각 모델을 사용하여 test dataset에 대한 accuracy를 측정했을 때의 결과를 정리한 것이다.

모델	Accuracy
kNN(k=40000)	0.7413
kNN(k=1)	1.0
Decision tree	1.0
SVM(linear)	0.9998
SVM(kernel)	1.0

3가지 모델 모두 거의 완벽한 성능을 보이는 것으로 확인된다. 더할 나위 없는 성능이 3가지 모델 모두에게서 확인되었으므로, 현재 주어진 데이터 상으로는 3개 모델 모두 연기 감지기로서 훌륭한 역할을 해낼 수 있는 모델이라고 말할 수 있을 것이다. 다만, k가 40000인 경우 underfitting으로 인해 성능이 많이 떨어지는 결과가 나타났다.

5.3.2 속도

아래 3개의 사진은 속도를 측정했을 때의 결과를 보여주고 있는 사진이다.
위에서부터 차례로 kNN(k=40000), kNN(k=1), decision tree, SVM(kernel) 모델의 predict 시의 속도 측정 결과 사진이다.

```
Total time: 169.927 s
File: run.py
Function: knn_predict at line 72

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    72                                           @profile
    73                                           def knn_predict(model, test_dataset):
    74         1         16.1     16.1      0.0      model.fit(train_dataset, train_y)
    75         1  169926688.6    2e+08    100.0      print(model.predict(test_dataset))
```

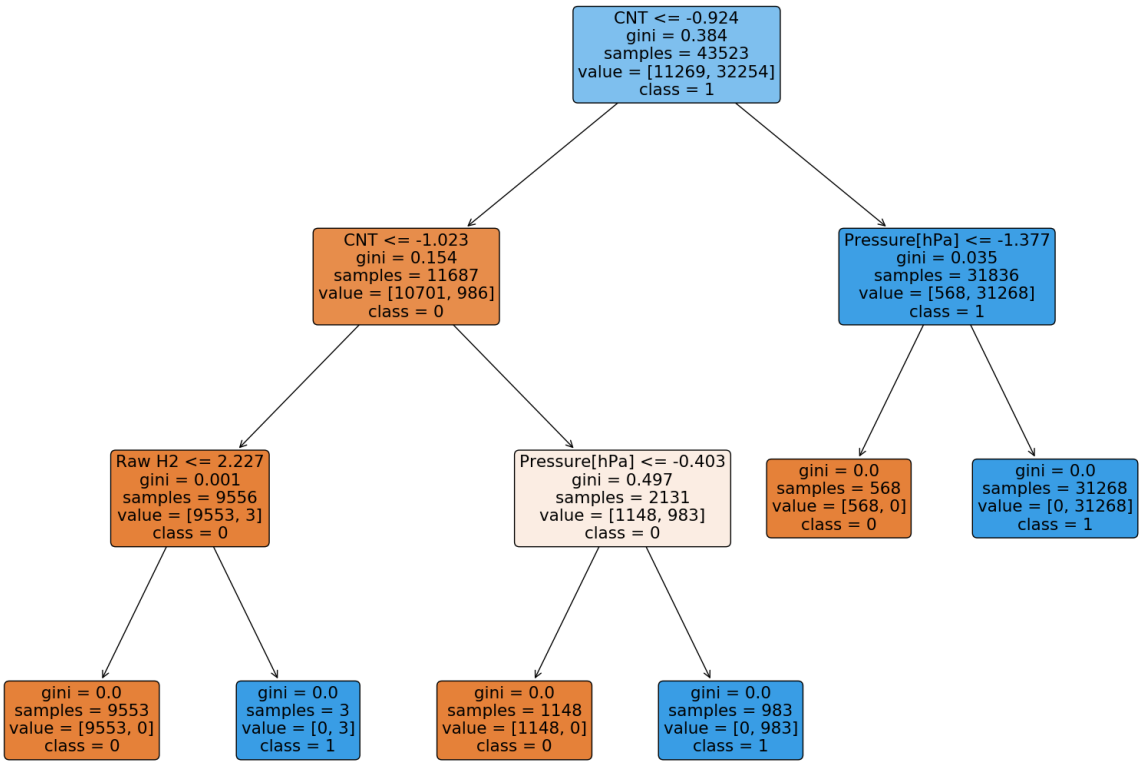
```
Total time: 9.92746 s
File: run.py
Function: knn_predict at line 72

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    72                                           @profile
    73                                           def knn_predict(model, test_dataset):
    74         1          4.0      4.0      0.0      model.fit(train_dataset, train_y)
    75         1    9927459.9    1e+07    100.0      print(model.predict(test_dataset))
```

```
Total time: 0.0299656 s
File: run.py
Function: tree_predict at line 91

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    91                                           @profile
    92                                           def tree_predict(model, test_dataset):
    93         1      29433.9  29433.9     98.2      model.fit(train_dataset, train_y)
    94         1        531.7    531.7      1.8      print(model.predict(test_dataset))
```

```
Total time: 1.75503 s
File: run.py
Function: svm_predict at line 130

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   130                                           @profile
   131                                           def svm_predict(model, test_dataset):
   132         1    1556631.9    2e+06     88.7      model.fit(train_dataset, train_y)
   133         1     198399.9 198399.9     11.3      model.predict(test_dataset)
```

Decision tree와 svm 그리고 kNN과의 속도 성능 차이는 굉장히 큰 것으로 확인되었다. 학습 단계인 fit 단계에서 kNN은 거의 시간이 측정되지 않을 정도로 빨라서 다른 모델들과 비교했을 때 학습 속도면에서는 가장 빠르지만, 추론을 위한 연산 과정에서 시간이 많이 걸리는 것으로 확인되었다. 특히, 연산 시 확인할 이웃의 수가 많은 경우에는 연산 시간이 굉장히 늘어나게 됨을 확인할 수 있다. 그리고 svm과 decision tree의 판단 속도를 비교했을 때에는 decision tree가 월등히 빠른 것으로 나타났다. 아무래도, 모든 feature를 고려하지 않고 중요한 feature들만 고려하여 판단을 한다는 decision tree의 inductive bias가 빠른 판단 속도를 내는 데에 결정적인 역할을 한 것으로 보인다.

아래는 추론을 위해 구성된 decision tree를 시각화한 모습이다.



kNN과 다른 모델들의 판단 속도 차이가 매우 큰 것은 sklearn 라이브러리 사용 유무에서도 영향을 받지 않았을까하는 생각이 들어, sklearn으로 구현한 kNN의 속도를 확인해 보았다. 그 결과는 아래 이미지와 같다.

```
Total time: 0.23634 s
File: run.py
Function: knn_predict at line 76

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    76                                           @profile
    77                                           def knn_predict(model, test_dataset):
    78         1      31485.0  31485.0     13.3      model.fit(train_dataset, train_y)
    79         1     204854.9 204854.9     86.7      print(model.predict(test_dataset))
```

sklearn 라이브러리를 사용하여 구현한 kNN with kd-tree(k=1)는 직접 구현한 kNN보다 훨씬 빠른 판단 속도를 보여주었다. 하지만, 여전히 decision tree의 판단 속도보다는 느린 것으로 확인되었다.

5.3 결과

Accuracy 실험과 속도 실험을 종합해보자면 아래의 표와 같이 정리할 수 있다.

모델	Accuracy	속도(학습 + 추론)

kNN(k=40000)	0.7413	170s
kNN(k=1)	1.0	9.927s
Decision tree	1.0	0.03s
SVM(kernel)	1.0	1.76s

실제 적용을 위해서라면 또 추가적인 사항들을 고려해야 할 것이기 때문에, 실제적인 적용에 있어서는 다른 모델이 사용될 수 있지만, 실험한 결과를 바탕으로 하였을 때에는 decision tree가 “가장 정확하고 빠른” 판단을 도출하는 모델인 것으로 확인되었다. 추가적으로, hyperparameter k의 조절을 통해서 kNN의 경우 이웃의 수에 따라서 성능 차이가 크게 달라질 수 있음을 확인할 수 있었다. 이를 통해, hyperparameter는 cross validation과 같은 과정을 통해서 신중하게 결정되어야 함을 알 수 있다.

6. 보완점

위와 같은 실험을 진행하면서 2가지 부분에 대해 아쉬움을 느꼈고, 다음에 실험을 진행하게 된다면 이 2가지를 보완하여 실험을 진행하면 더 좋을 것 같다는 생각이 들었다. 그 2가지는 아래와 같다.

(1) kd-tree를 이용한 kNN 모델 구현
Kd-tree라는 자료구조를 이용하여 kNN 모델을 구현할 수 있다. Kd-tree를 이용하게 된다면, kNN은 학습 시간이 빠른 것이 특징인데, 이 특징은 그대로 가져가면서 추론을 위한 연산 시간을 자료구조를 이용해 줄일 수 있게 된다. sklearn 라이브러리로 이를 구현하여 판단 속도를 확인하기는 했지만, 이는 직접 구현했을 때의 속도보다는 빠르게 측정될 가능성이 높기에, 직접 구현한 kNN with kd-tree를 가지고 여러 비교를 해보았으면 더 좋은 실험이 되지 않았을까 하는 생각이 들었다.

(2) SVM, decision tree 직접 구현
SVM과 decision tree 또한 직접 구현하여, 직접 구현한 모델끼리 성능을 비교해보았으면 더 흥미로운 실험을 진행할 수 있지 않았을까하는 생각이 들었다.

7. 결론

위의 실험을 바탕으로 연기 탐지 데이터셋을 활용하여 가장 좋은 연기 탐지기 모델로서의 역할을 할 수 있는 모델은 decision tree인 것으로 확인되었다. 실제적인 적용에 있어서는 고려해야 하는 여러가지 요소들이 있을 것이기 때문에, 실제 적용 모델은 달라질 수 있을 가능성이 존재한다. 최종 선택 모델은 decision tree이지만, 주목해야 할 부분은 실험에 사용된 모델 모두 100%의 accuracy를 보여주었다는

점이라고 생각한다. Deeplearning 모델을 사용한다고 해도 물론 좋은 성능을 보여줄 가능성이 높지만, 추론 속도까지 고려를 해본다면 실험에 사용된 3가지 모델은 모두 우수한 성능을 보여준 것이라고 생각된다. 이를 통해, 무조건적인 deeplearning 모델 사용은 지양하는 것이 좋고, 상황을 적절하게 분석하여 다양한 모델을 적재적소에 사용하는 것이 중요함을 알 수 있었다.