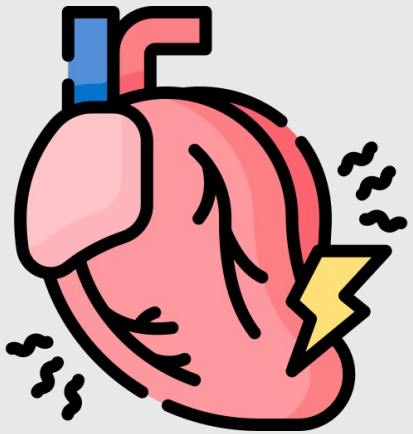
2024 Data Structures

Using Three Tree-Based Ensemble Models to Assess the Risk of Myocardial Infarction

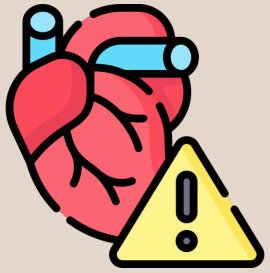
세가지 트리 기반 앙상블 모델을 활용한 심근경색 위험도 측정

Ewha Womans University
Department of Data Science
2392032 Cheong Jiyoung
2392038 Han Kaeun

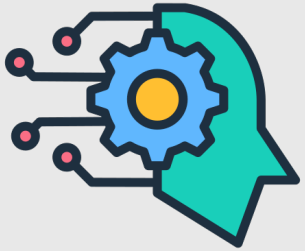


Contents / 목차

Introduction / 주제 설명



Myocardial Infarction (심근경색)
= Sudden & Life-threatening Disease
-> What if we could easily and frequently check its risk ?



Using tree-based AI Models
to predict simplified myocardial infarction (심근경색)

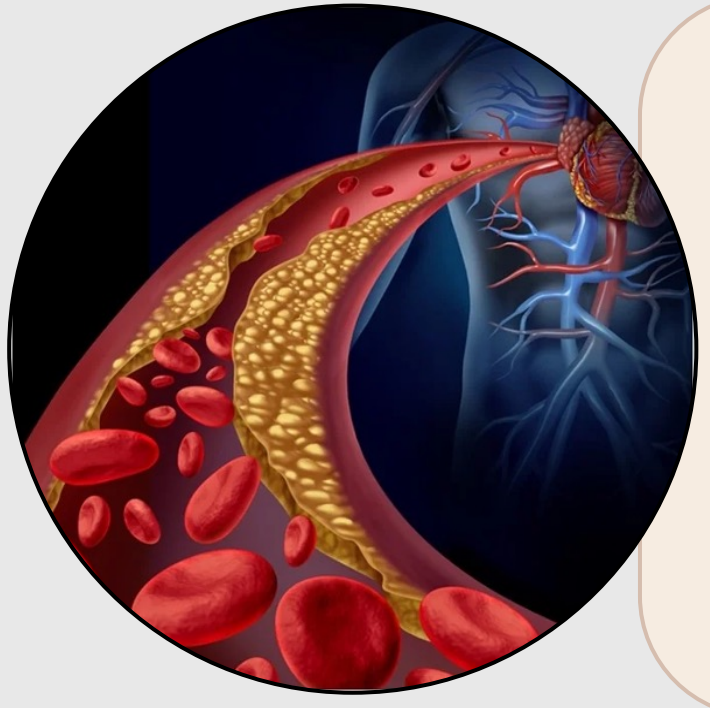
Introduction / 주제 설명

Daily medical check & Manage everyday lifestyle

Effects :
Make Body Healthier
& Protect us from Myocardial Infarction !!

Motivation / 동기

The Danger of Disease : Myocardial Infarction (심근경색)



Sudden chest pain without early symptoms
Clotted blood blocks blood vessels
33% death rate upon arrival at the hospital
5-10% survival rate with immediate medical care

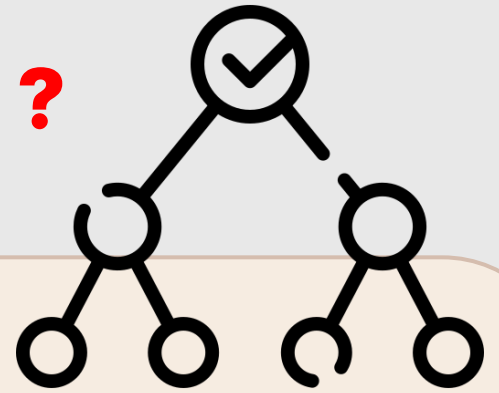
Primary factors : Cholesterol, Blood Pressure, Age

[Picture Reference]

[서울대학병원 N 의학정보]

Motivation / 동기

Q. Why did we choose tree-based models ?

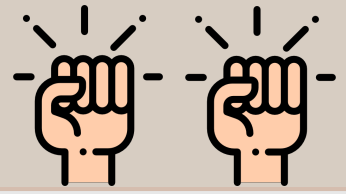


A.

- **Designate various decision criteria to each node**
- **Direction from parent node to child node**
  -> Easy to understand decision making process
- **Searching is fast in trees (Time complexity : O(log n))**
- **High classification accuracy and easy data visualization**

Role / 역할 분담

Together



**Searching topic, Data Discovery,
Model Reviews, Conclusion, Developments**

Jiyoung

Data Download & Bringing SAS data
Data Preprocessing (Data Scaling, Feature Selection)
Methods : Decision Tree, XGBoosting
Source Making : Motivation, Decision Tree, Gradient Boosting, XGBoosting

Kaeun

PI, Data Preprocessing (Data Sampling, Data Cleaning)
Methods : Random Forest / Gradient Boosting
**Primarily Making Slides : Role, Data Information, Data Preprocessing,
Ensemble Learning, Random Forest, Before Application**

Data Information / 데이터 정보

[국민건강영양조사]

국민건강영양조사 원시자료 2019 – 2021 (제 8기) 기본 DB

Contents : 검진조사, 건강설문조사, 영양조사

2019 : 8110 Data / 912 Variables

2020 : 7359 Data / 859 Variables

2021 : 7090 Data / 864 Variables

Columns = 행(세로) = Variables = Features

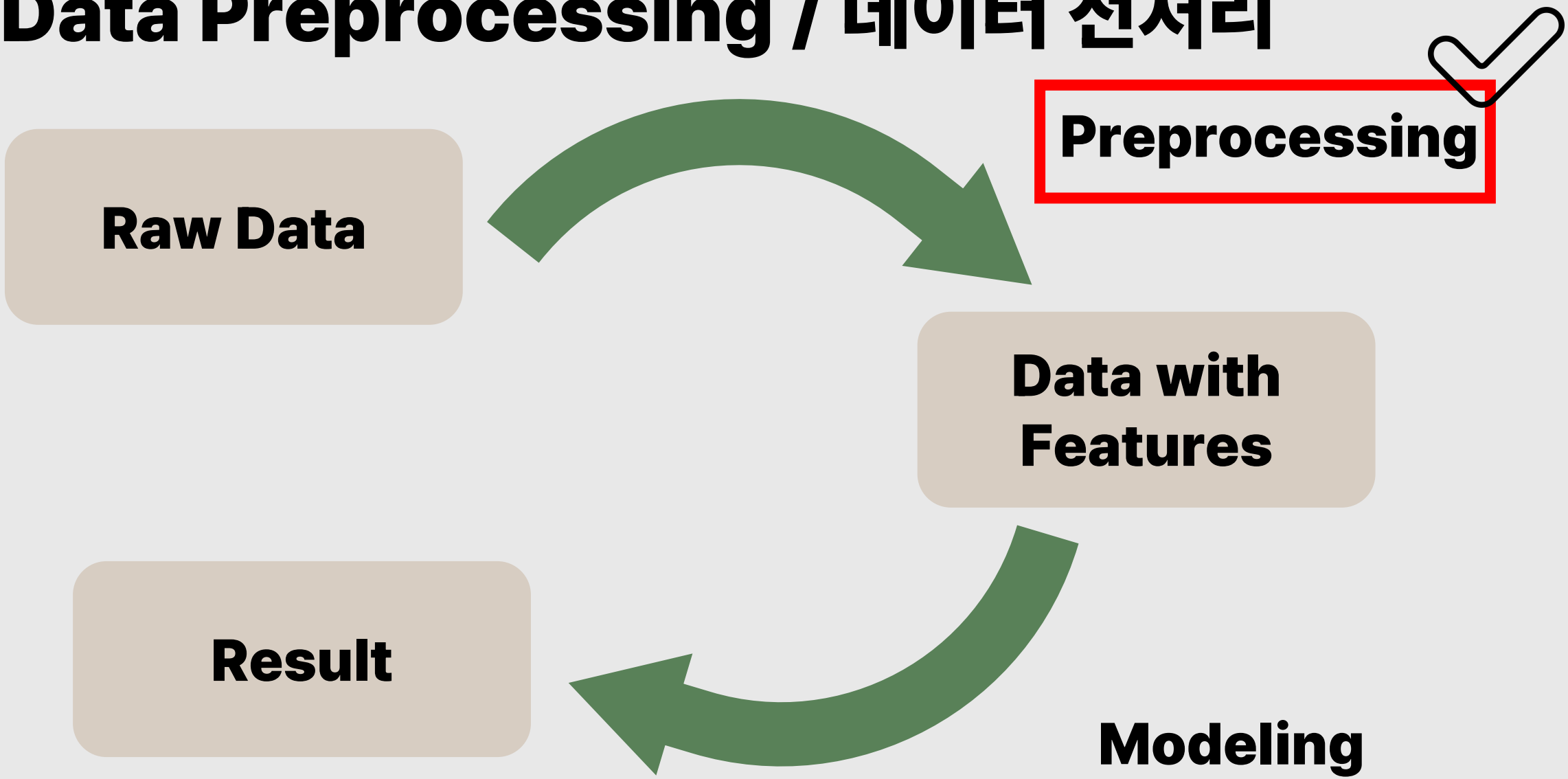
```
hn21_s.head()
```

	sex	age	DI1_dg	DI2_dg	DE1_dg	DI3_dg	DF2_dg	DI1_pr	DI2_pr	DE1_pr	...	HE_HLfh1	HE_HLfh2	HE_IHDfh1	HE_IHDfh2	HE_STRfh1	HE_STRfh2	HE_DMfh1	HE_DMfh2
0	1.0	61.0	1.0	0.0	0.0	0.0	1.0	1.0	8.0	8.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	1.0	57.0	1.0	0.0	0.0	0.0	0.0	1.0	8.0	8.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	2.0	39.0	0.0	0.0	0.0	0.0	0.0	8.0	8.0	8.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	2.0	19.0	0.0	0.0	0.0	0.0	0.0	8.0	8.0	8.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	1.0	70.0	0.0	0.0	0.0	0.0	0.0	8.0	8.0	8.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 33 columns

Rows = 열(가로) = Data = Observations = Samples

Data Preprocessing / 데이터 전처리



Data Preprocessing / 데이터 전처리

1. Data Cleaning :
Missing Values(NULL) + Uninformative values(모름/무응답)
We Discarded rows with missing and uninformative values.

2. Data Scaling : Standardization
(Rescaling different variables to a consistent range)
Rescaling each variable average 0, variance 1.

We used StandardScaler() provided by Scikit-learn.

Data Preprocessing / 데이터 전처리

3. Feature Selection : Selecting important variables

Raw(Original) Data :
Include many variables (both informative, uninformative)
-> Need to reduce variables for model training !!

Criteria we used in selecting important variables :

원시자료이용지침서 (국민건강영양조사)
Medical Information about Myocardial Infarction(심근경색)

[Medical Information Reference1] [Medical Information Reference2]

Data Preprocessing / 데이터 전처리

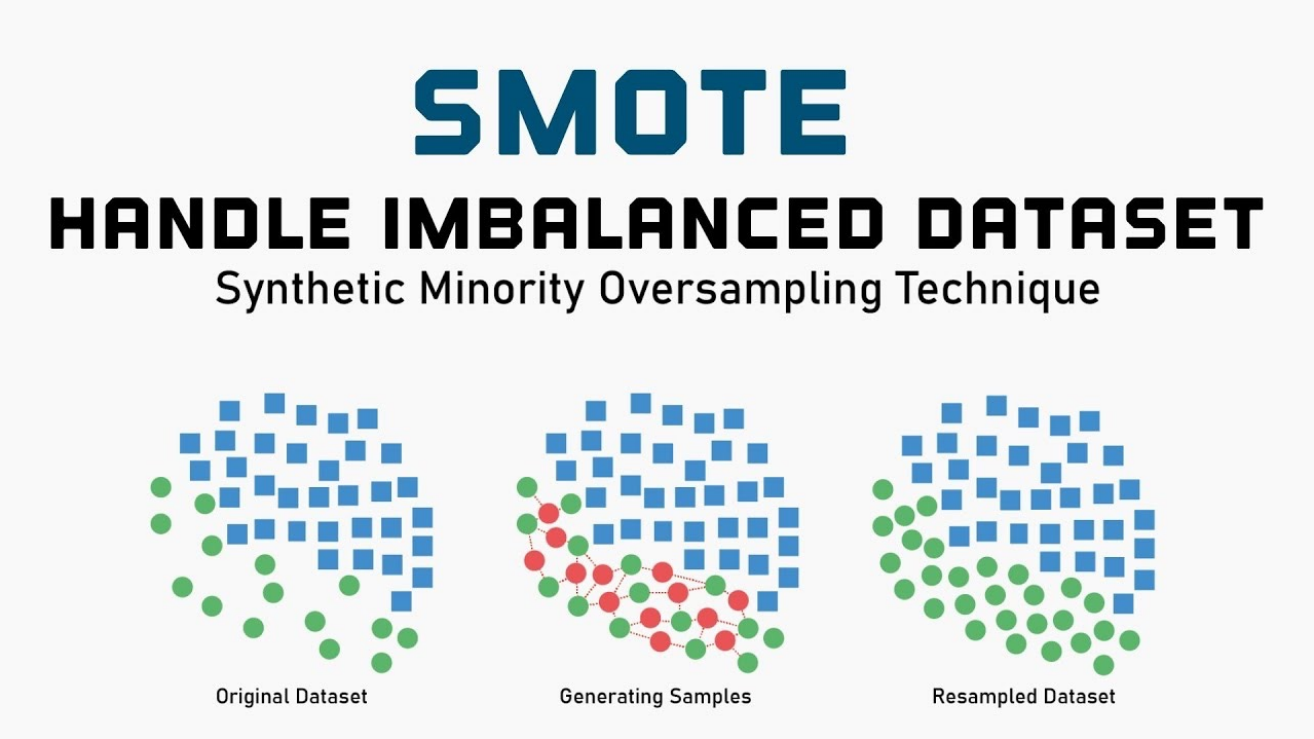
4. Data Sampling : Oversampling – SMOTE

	14215	54	348
Classes of Y(심근경색) :	NA(0)	SAFE(1)	DANGER(2)

We observed **Class Imbalance** in the dataset.
Class imbalance : Observations are **unevenly** distributed.
-> This may cause bias during model training.
-> Need to increase sample data in **minority** classes !!

Data Preprocessing / 데이터 전처리

4. Data Sampling : Oversampling
SMOTE (Synthetic Minority Oversampling Technique)



[Picture Reference]

In Our Data

Before : 0 1 2

14215 54 348

After : 0 1 2

14215 14215 14215

Data Preprocessing / 데이터 전처리

Introduction of Major Variables:

Total : 32 Variables

X (Independent Variables)

'age'	연령	Numerical
'DI1_pr'	고혈압 현재 유병여부	Categorical
'DE1_pr'	당뇨병 현재 유병여부	Categorical
'HE_glu'	공복혈당	Numerical
'HE_chol'	총콜레스테롤	Numerical
'HE_sbp'	수축기 혈압	Numerical
'HE_dbp'	이완기 혈압	Numerical

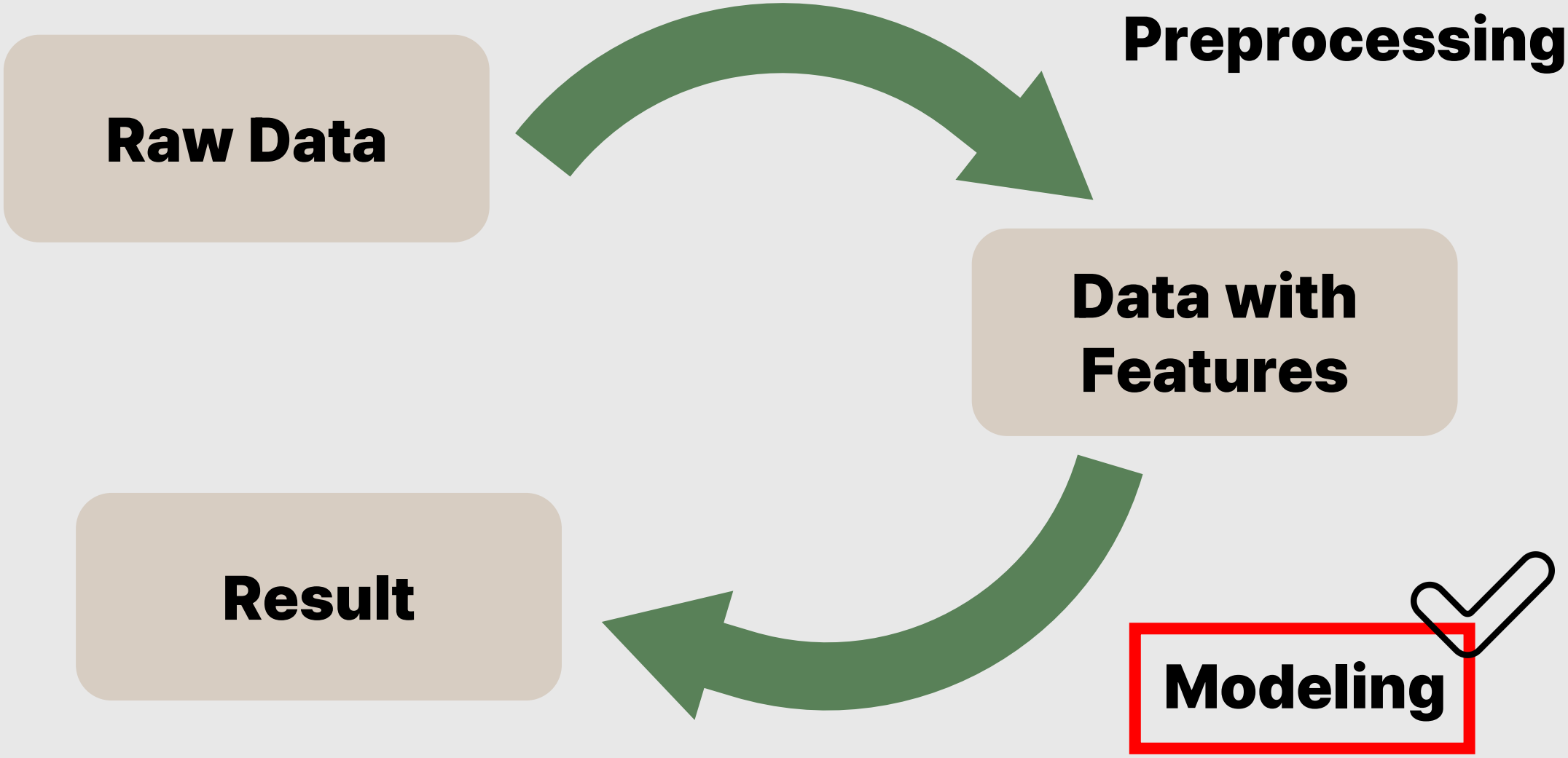
Data Information / 데이터 정보

Introduction of Major Variables:

Y (Dependent Variable)

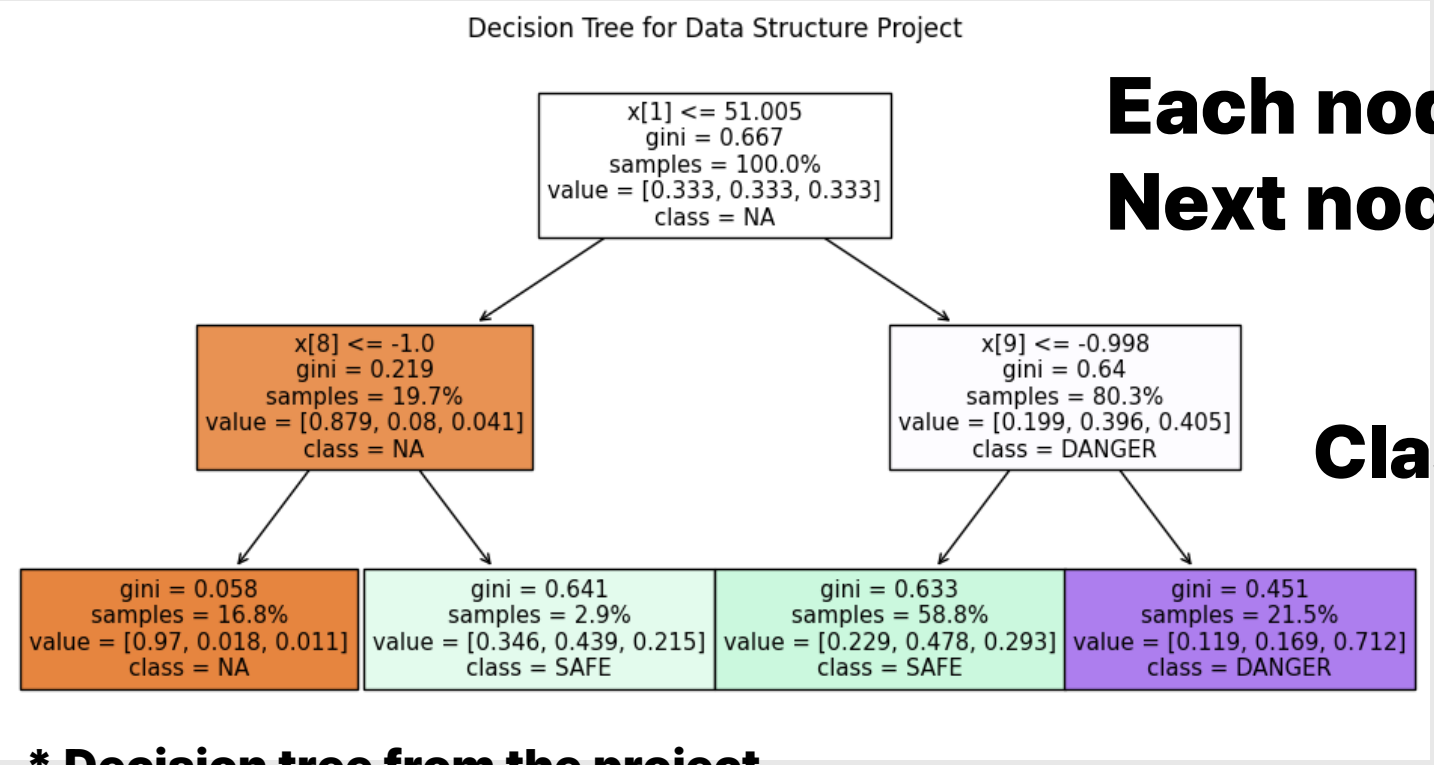
'DI4_pr'	심근경색 또는 협심증 현재 유병여부	Categorical

Methods / 방법론


Raw Data
Preprocessing
Data with Features
Modeling
Result

Methods / 방법론 – Decision Tree

Separates Data according to specific criteria(questions)
Similar to human's decision-making process

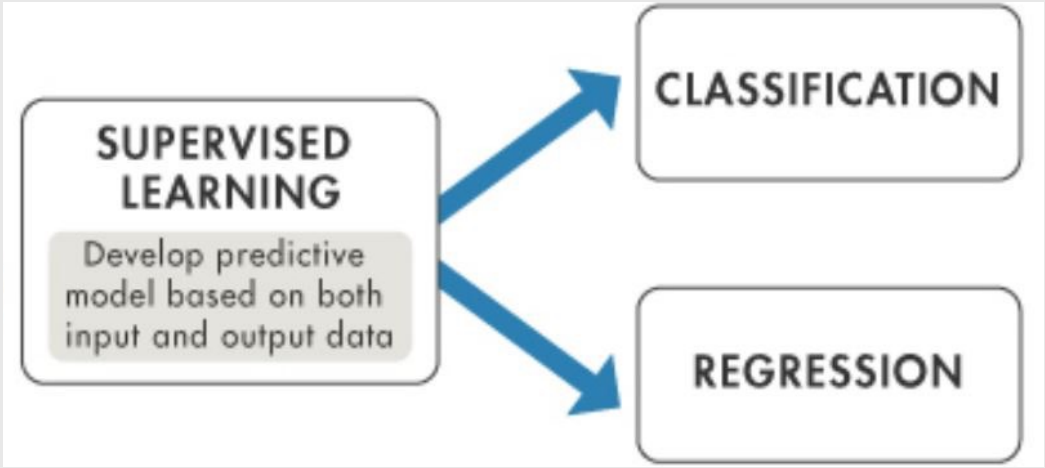


Each node contain split criterion
Next node is decided through it

Class = [NA, SAFE, DANGER]

*** Decision tree from the project**

Methods / 방법론 – Decision Tree



Decision Tree

- **Supervised Learning Method**

Supervised Learning :
Learning method including both X and Y Variables
Classification :
Predicting categorical output on given data
Regression :
Predicting numerical output on given data

[Picture Reference]

Methods / 방법론 – Decision Tree

Pros

Simple and effective for classifying
: especially decision for medical field
White-box model : easy to interpret for human

Cons

Easy to be overfitted
(next page)

Methods / 방법론 – Decision Tree

Overfitting : Model is too precisely trained on given data, which may result in bad prediction to new data.

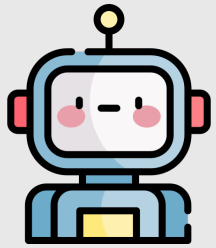
Common problem in decision trees (too much classified)

Solution : Ensemble Models ! (training multiple decision trees)

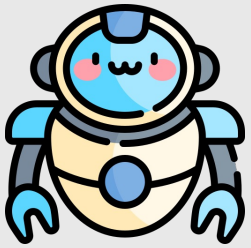
Methods / 방법론 – Ensemble Learning

Ensemble

“ A group of Weak Learners can come together to form a Strong Learner ”

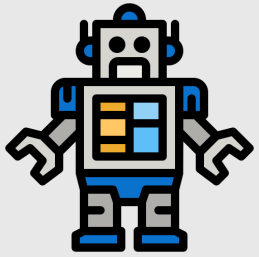


Weak Learner 1

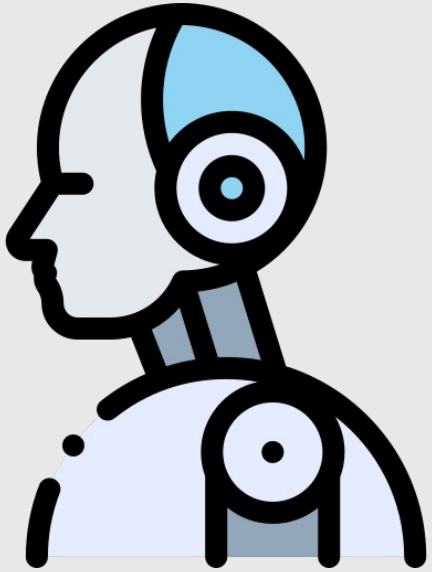


Weak Learner 2

Weak Learner 3



Weak Learner 4



Strong Learner

Methods / 방법론 – Ensemble Learning

Ensemble

Create multiple classifiers
+ Combine their predictions
=> a more Accurate Final Prediction

Bagging — **Boosting** — **Stacking**

Random Forest

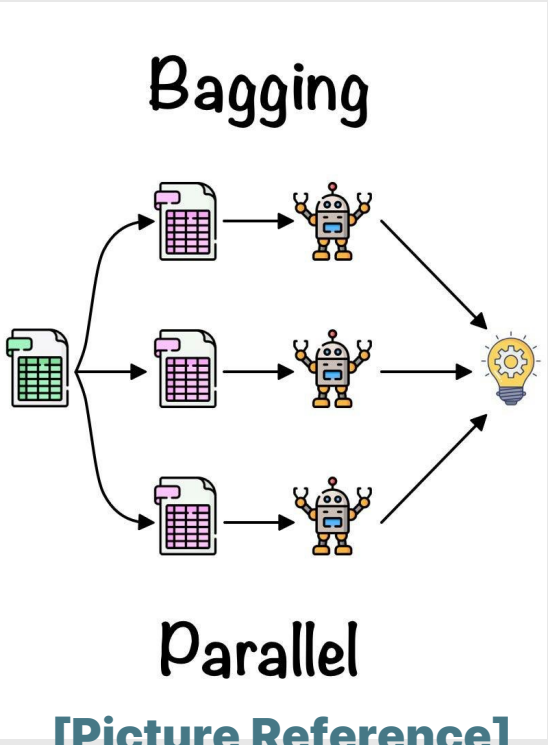
Gradient Boosting
XGBoosting

Methods / 방법론 – Ensemble Learning

Bagging

Bootstrap Aggregating = Bagging
Aggregate the bootstrap sampling !

- **Bootstrap Sampling** :
  Random Sampling with Replacement
- **Train individual models on each dataset**
- **Aggregation** :
  - **Classification : Majority Voting**
  - **Regression : Average of Outputs**



[Picture Reference]

Related Model : Random Forest

Methods / 방법론 – Ensemble Learning

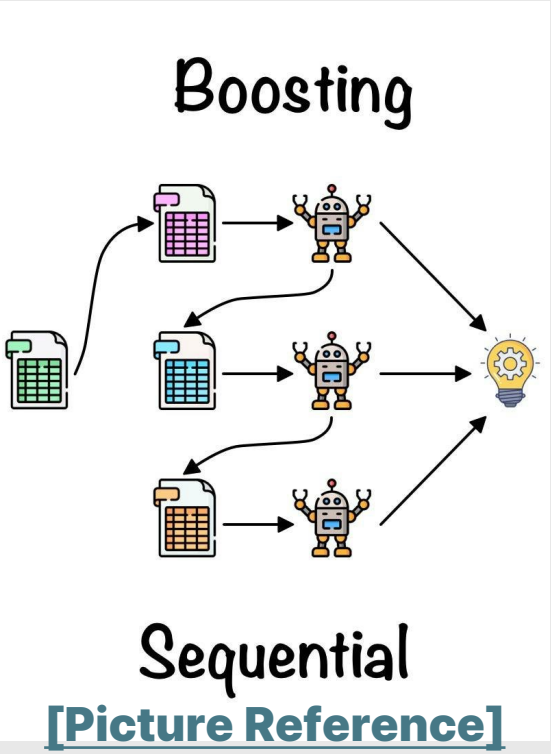
Boosting

Iterative Learning

Update the models by focusing on the errors of previous iterations

- **Increase the weight of misclassified data**
- **Focus more on difficult cases**
- **Combine these weak learners linearly**

Related Model :
Gradient Boosting, XGBoosting



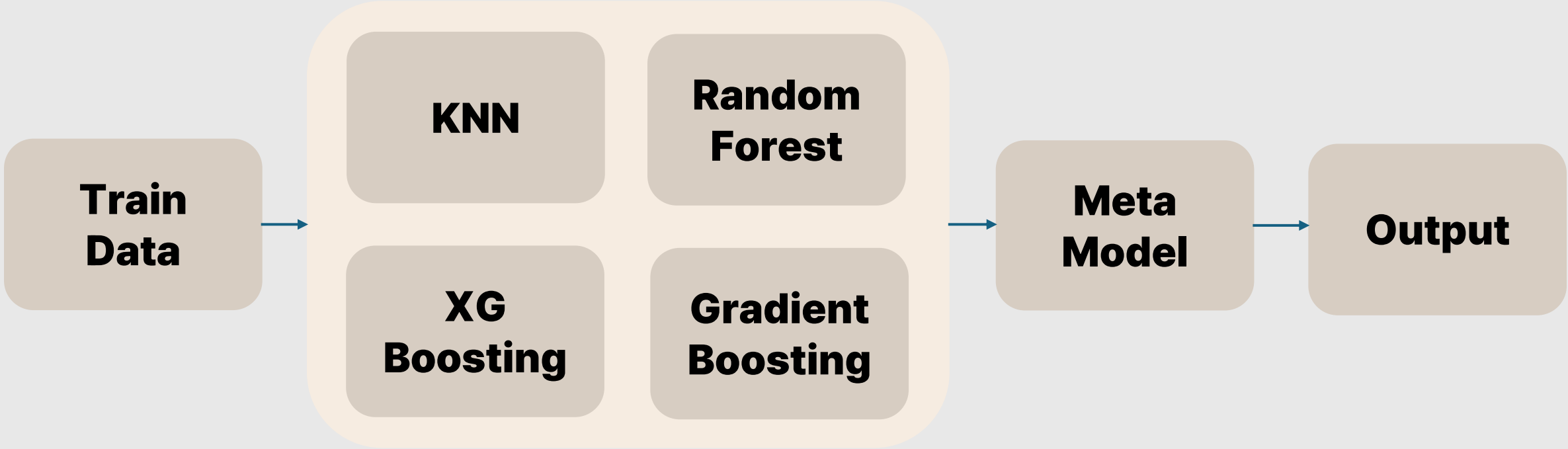
[Picture Reference]

Methods / 방법론 – Ensemble Learning

Stacking

Meta Learning

Combining predictions from Multiple Models (meta-model)



Methods / 방법론 – Ensemble Learning

Pros

Accuracy and Stability
Reduced overfitting problem
: Often vulnerable in decision tree algorithm

Cons

Interpretability
: Hard to interpret the decision-making of models
Computational Complexity

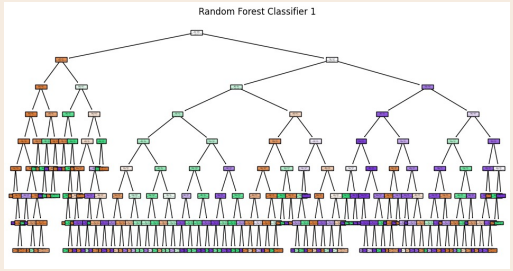
Methods / 방법론 – Random Forest

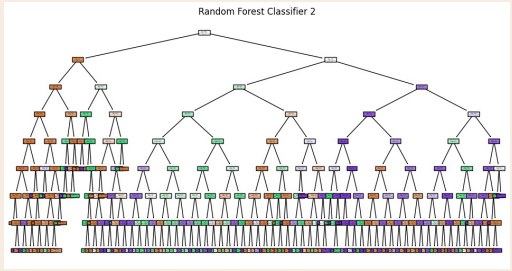
Ensemble - Bagging

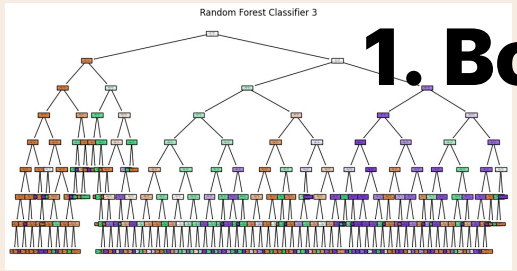
Random Data Sampling + Multiple Decision Trees
=> " Random Forest "

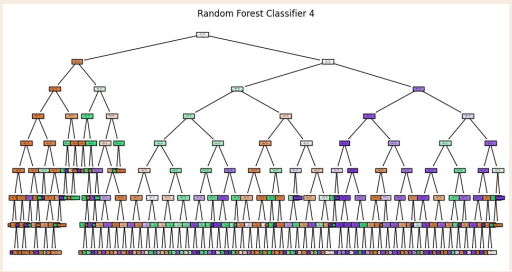
1. **Create Random Samples - Bootstrap Sampling**
2. **Train multiple decision trees**
3. **Aggregate them in a single model**
4. **Predict the output**

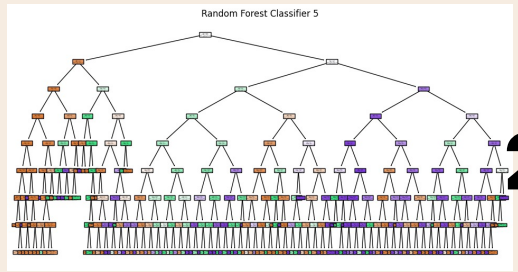
Methods / 방법론 – Random Forest











1. Bootstrap sampling

2. Multiple Decision Trees

***Trees from the project**

3. Aggregating

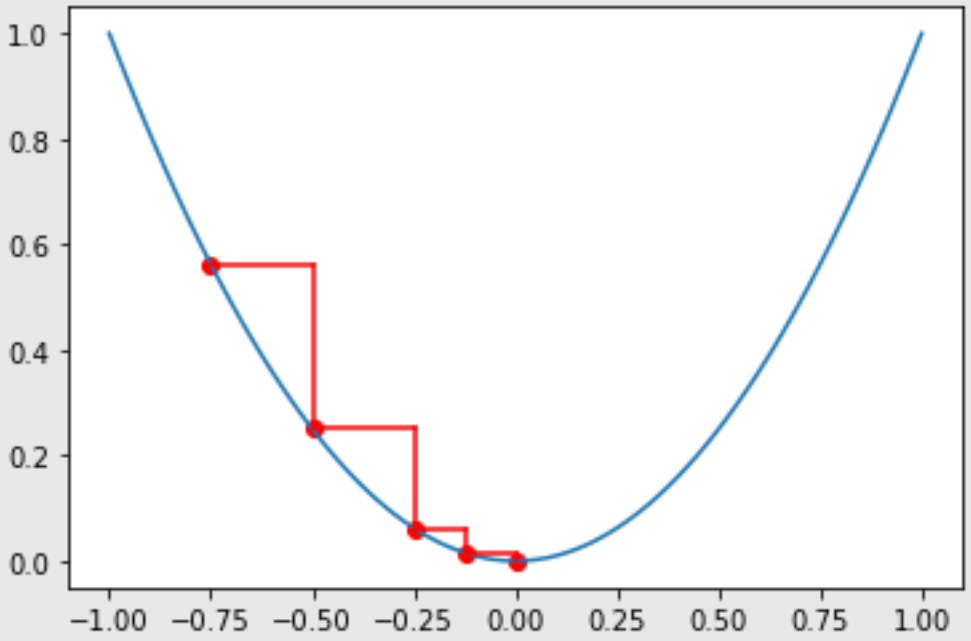
4. Prediction Output

Methods / 방법론 – Gradient Boosting

Ensemble – Boosting

Use gradient descent sequentially for enhancing previous model

- **Gradient Descent : Moving towards smaller gradients to locate the point with the minimum gradient**

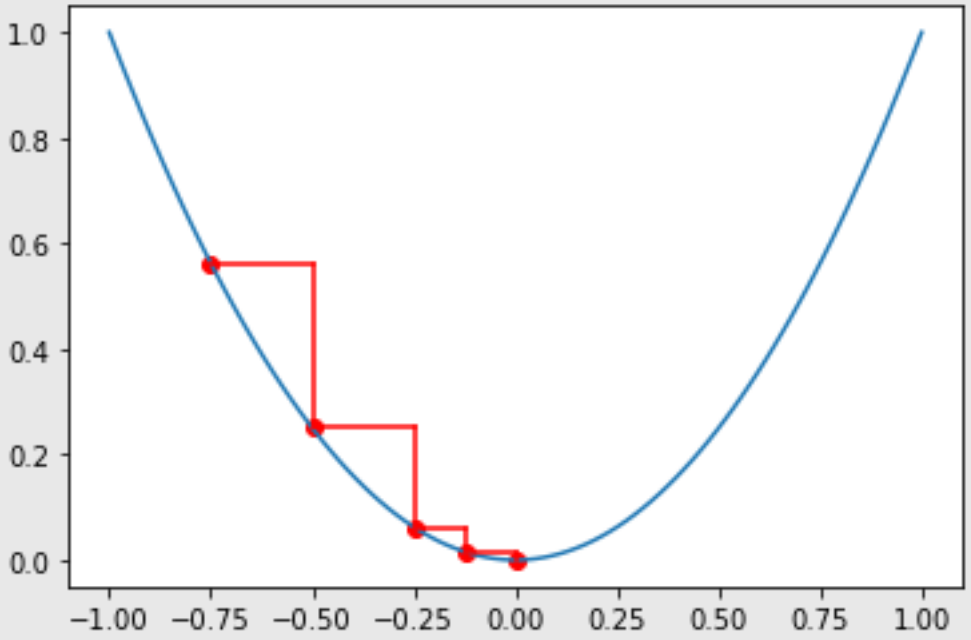


[Picture Reference]

Methods / 방법론 – Gradient Boosting

- **Cost Function: Function that shows how well the model predicted**

Keep updates errors through gradient descent by reducing the value of cost function

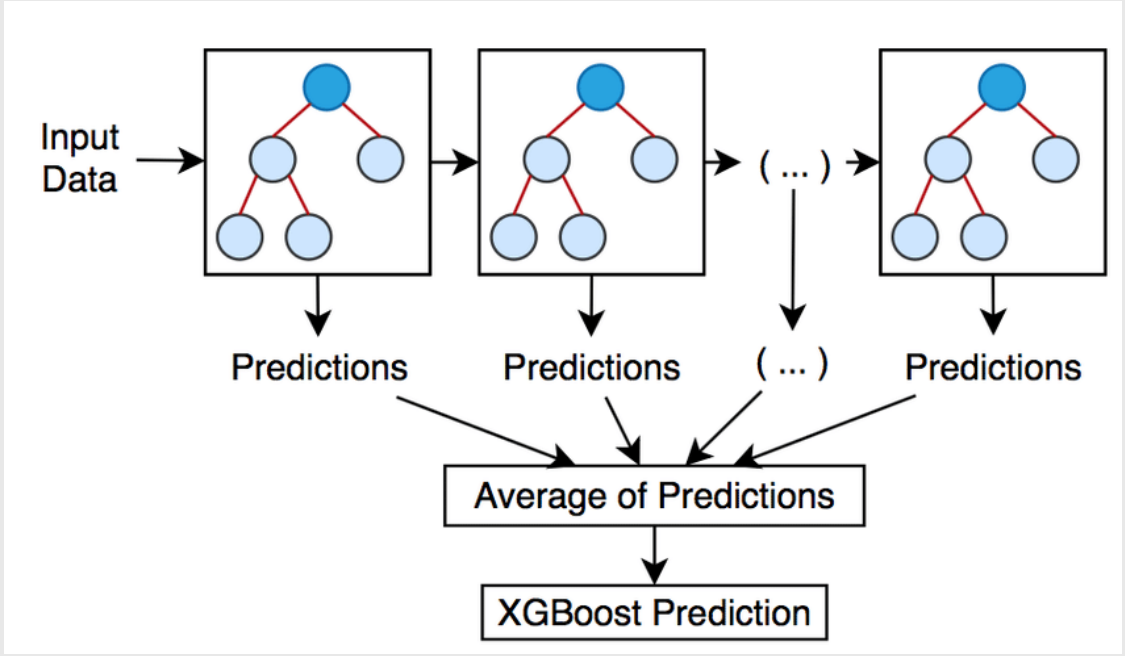


[Picture Reference]

Methods / 방법론 – XGBoosting

Ensemble – Boosting

Like gradient boosting
- Less overfitted, and thus
- Better precision



[Picture Reference]

Thanks to Regularization
Regularization
: Adjust the importance of each factor to enhance fitness

Before Application : Modeling

Using Scikit-learn (Python Programming Library)

Common Modeling Structure :

```
model = DecisionTreeClassifier() -> 모델 선언
model.fit(train_x, train_y) -> 모델 훈련
model.predict(test_x) -> 모델 예측
```

Split data into train and test -> Train the data with model -> Use this model to predict y variables of test dataset

Before Application : Modeling

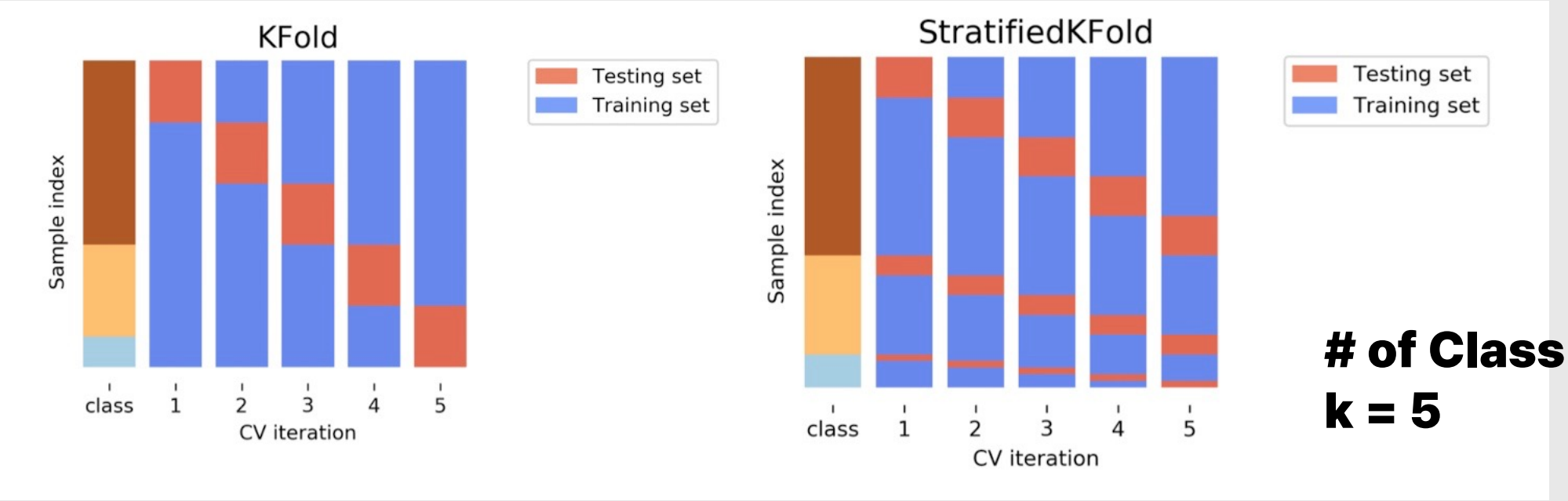
Using Scikit-learn (Python Programming Library)

Classifiers we used in Scikit-learn:

DecisionTreeClassifier()
RandomForestClassifier()
GradientBoostingClassifier()
XGBClassifier()

Those are the classifiers to build each predictive models.

Before Application : Stratified k-fold



# of Class : 3
k = 5

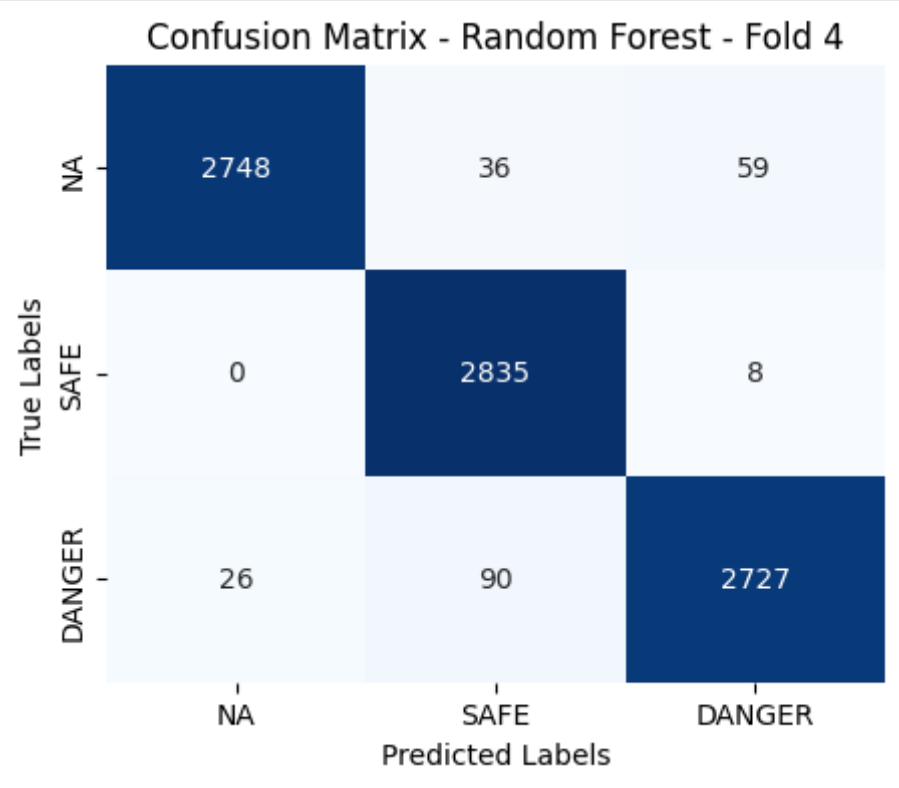
[Picture Reference]

k-fold : Simply divides data into number of folds of equal size
But ! All data may not be evenly distributed across each fold
Stratified k-fold : Divides data evenly across classes

Before Application : Metrics

F1-score : Harmonic mean of Precision and Recall

Confusion matrix : Tool for evaluating classification models using actual and predicted classes matching



Precision :

Actual [Class] out of Model Predicted [Class]

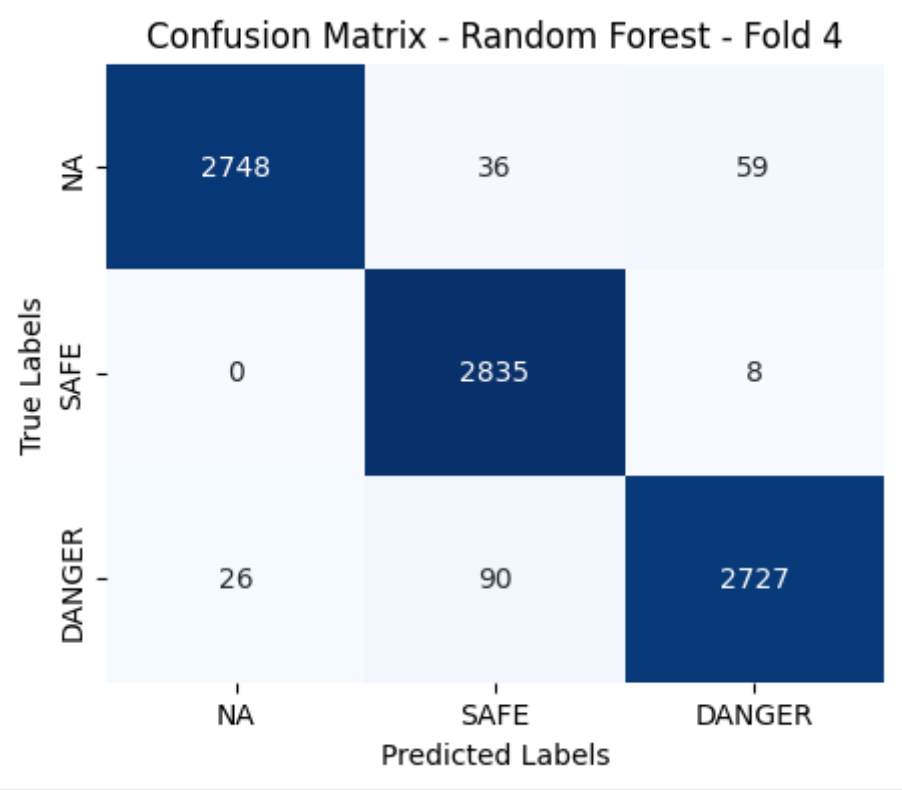
Precision of Class [Danger] : 2727 / (2727+8+59) = 0.976

***Confusion matrix from the project**

Before Application : Metrics

Recall : Model Predicted as [Class] out of Actual [Class]

Accuracy : Predicted Correctly out of Total



Recall of Class [Danger] : 2727 / (26+90+2727) = 0.959

Accuracy : (2748+2835+2727) / 8529 = 0.974

***Confusion matrix from the project**

Application / 적용 – Decision Tree

```python
k_fold = StratifiedKFold(n_splits=5)  # k = 5

dt_f1_scores = []
accuracies = []
precisions = []
recalls = []

for fold, (train_idx, test_idx) in enumerate(k_fold.split(X_train_over, y_train_over), 1):
    train_x, train_y = X_train_over[train_idx], y_train_over[train_idx]
    test_x, test_y = X_train_over[test_idx], y_train_over[test_idx]

    dt = DecisionTreeClassifier()
    dt.fit(train_x, train_y)
    pred_y = dt.predict(test_x)
```

Stratified k-fold

Modeling

```python
# Confusion Matrix 시각화
plt.figure(figsize=(5, 4))
sns.heatmap(confusion_matrix(test_y, pred_y), annot=True, cmap='Reds', fmt='d', cbar=False,
            xticklabels=['NA', 'SAFE', 'DANGER'], yticklabels=['NA', 'SAFE', 'DANGER'])
plt.title(f"Confusion Matrix - Decision Tree - Fold {fold}")
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

Confusion Matrix

Application / 적용 – Decision Tree

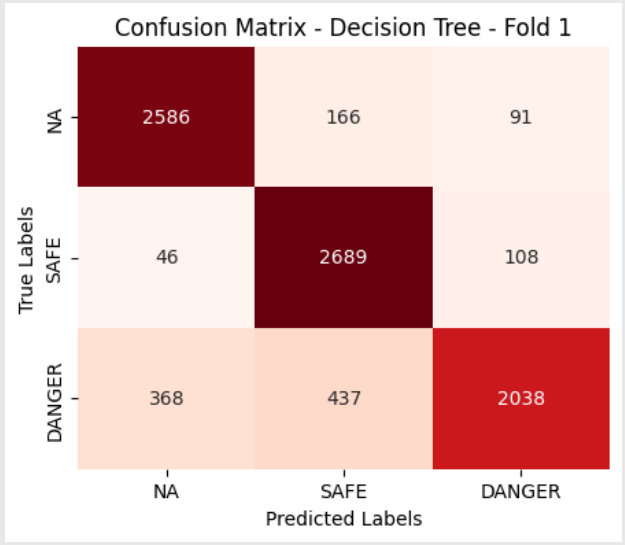
```
    # 각 fold에서 f1-score를 출력
    print(f"Fold {fold}:")
    print(f"  F1-score: {dt_f1:.4f}")
    print(f"  Accuracy: {accuracy:.4f}")
    print(f"  Precision:")
    for j in range(len(precision)):
        print(f"    Class {j}: {precision[j]:.4f}")
    print(f"  Recall:")
    for j in range(len(recall)):
        print(f"    Class {j}: {recall[j]:.4f}")
    print()
```

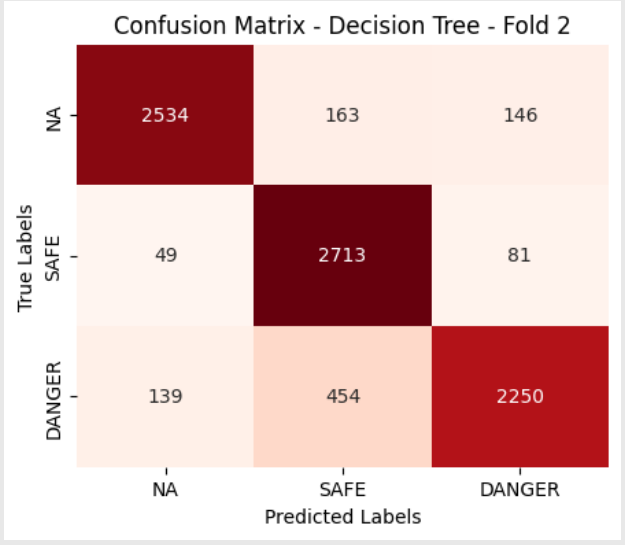
Printing Metrics for each fold

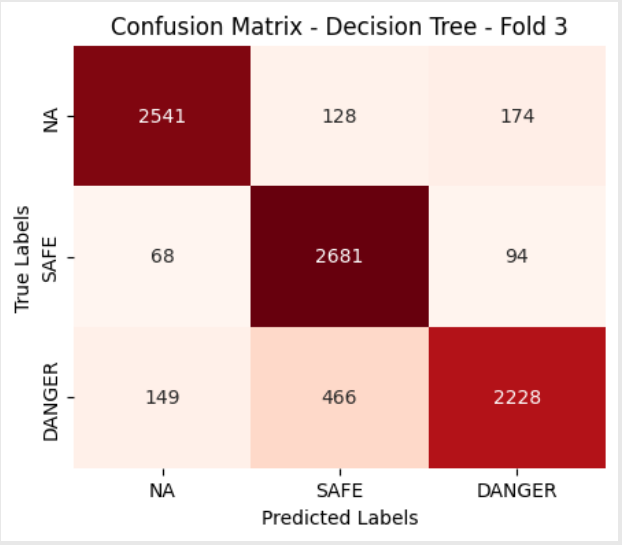
```
# 평균 f1-score, accuracy, precision, recall 계산 및 출력
avg_f1 = np.mean(dt_f1_scores)
avg_accuracy = np.mean(accuracies, axis=0)
avg_precision = np.mean(precisions, axis=0)
avg_recall = np.mean(recalls, axis=0)

print("Decision Tree")
print(f"  Average F1-score: {avg_f1:.4f}")
print(f"  Average Accuracy: {avg_accuracy:.4f}")
print("  Average Precision:")
for j in range(len(avg_precision)):
    print(f"    Class {j}: {avg_precision[j]:.4f}")
print("  Average Recall:")
for j in range(len(avg_recall)):
    print(f"    Class {j}: {avg_recall[j]:.4f}")
```

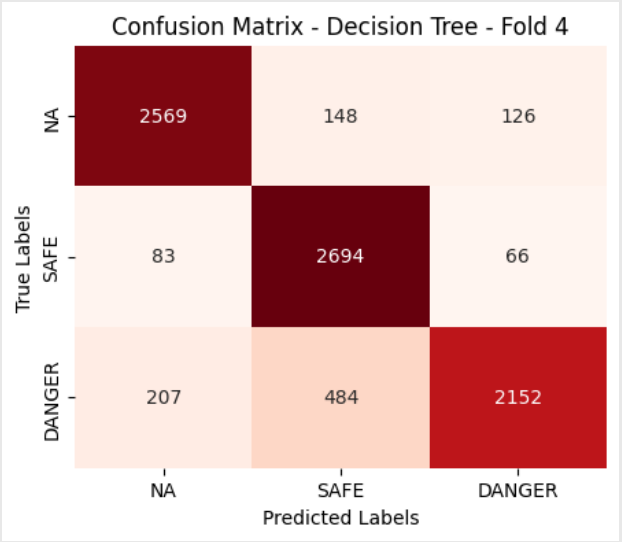
Printing Average of Metrics across folds

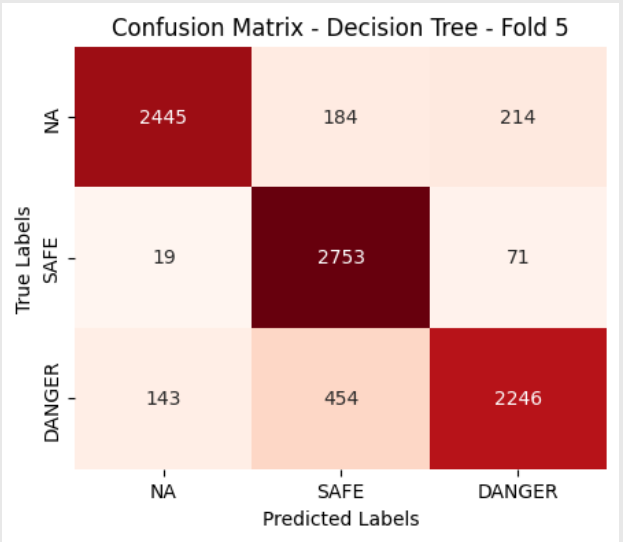
Application / 적용 – Decision Tree







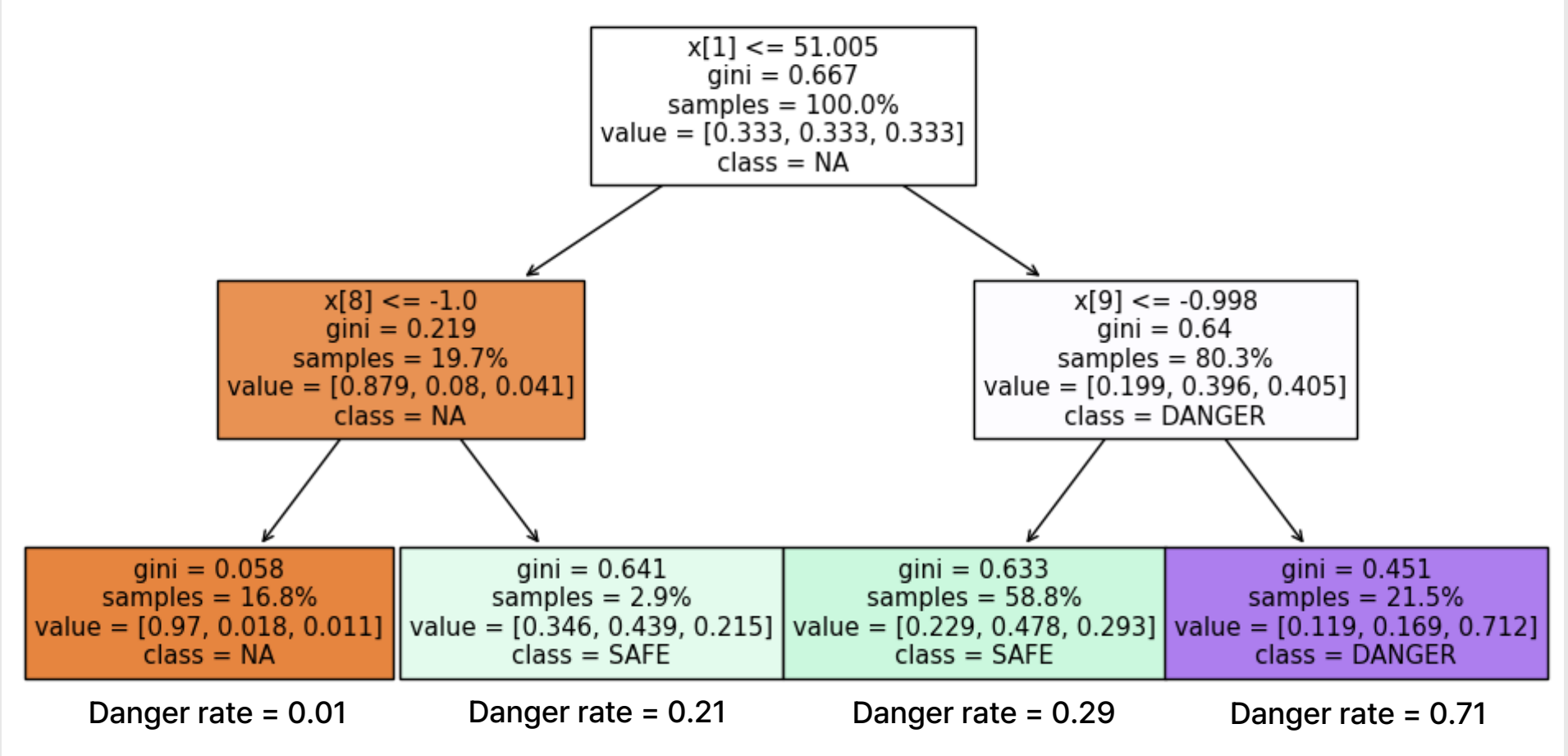





```
Decision Tree
  Average F1-score: 0.8704
  Average Accuracy: 0.8704
  Average Precision:
    Class 0: 0.9101
    Class 1: 0.8144
    Class 2: 0.9035
  Average Recall:
    Class 0: 0.8917
    Class 1: 0.9518
    Class 2: 0.7678
```

Application / 적용 – Decision Tree

Calculating Danger rate when depth = 2 (for visual clarity)



Application / 적용 – Random Forest

```
k_fold = StratifiedKFold(n_splits=5) # k = 5

rf_f1_scores = []
rf_accuracies = []
rf_precisions = []
rf_recalls = []

for fold, (train_idx, test_idx) in enumerate(k_fold.split(X_train_over, y_train_over), 1):
    print(f'Fold {fold}:')

    x_train, y_train = X_train_over[train_idx], y_train_over[train_idx]
    x_test, y_test = X_train_over[test_idx], y_train_over[test_idx]

    scaler = StandardScaler()
    x_train = scaler.fit_transform(x_train)
    x_test = scaler.transform(x_test)

    # Random Forest Classifier
    rf = RandomForestClassifier()
    rf.fit(x_train, y_train)
    y_pred_rf = rf.predict(x_test)
```

Stratified k-fold

Modeling

```
# Confusion Matrix 시각화
plt.figure(figsize=(5, 4))
sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, cmap='Blues', fmt='d', cbar=False,
            xticklabels=['NA', 'SAFE', 'DANGER'], yticklabels=['NA', 'SAFE', 'DANGER'])
plt.title(f"Confusion Matrix - Random Forest - Fold {fold}")
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

Confusion Matrix

Application / 적용 – Random Forest

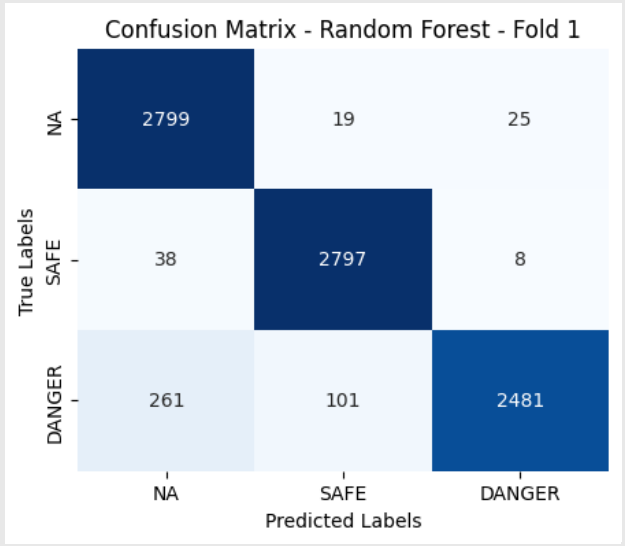
```
# 각 fold에서 f1-score를 출력
print(f"Fold {fold}:")
print(f"  F1-score: {rf_f1:.4f}")
print(f"  Accuracy: {rf_accuracy:.4f}")
print(f"  Precision:")
for j in range(len(rf_precision)):
    print(f"    Class {j}: {rf_precision[j]:.4f}")
print(f"  Recall:")
for j in range(len(rf_recall)):
    print(f"    Class {j}: {rf_recall[j]:.4f}")
print()
```

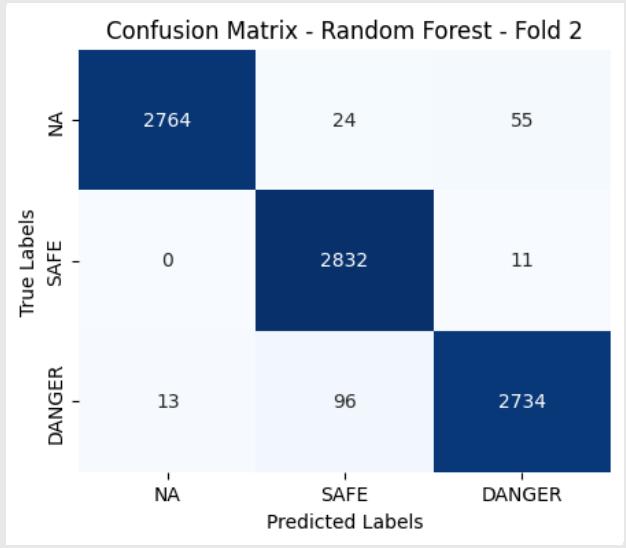
Printing Metrics for each fold

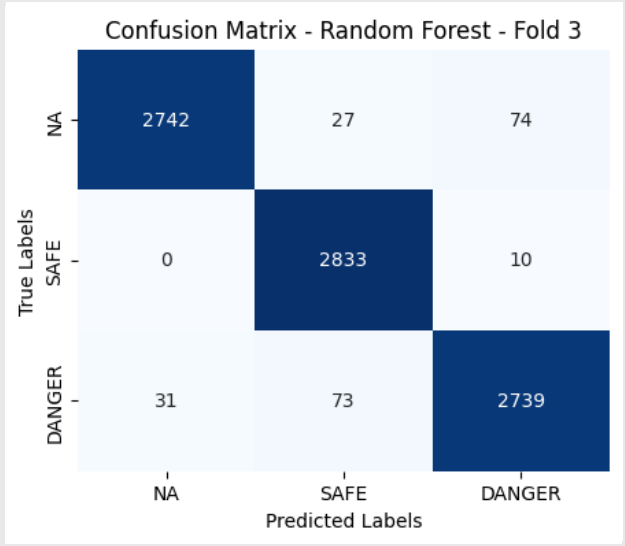
```
# 평균 F1-score, Accuracy, Precision, Recall 계산 및 출력
rf_avg_f1 = np.mean(rf_f1_scores)
rf_avg_accuracy = np.mean(rf_accuracies)
rf_avg_precision = np.mean(rf_precisions, axis=0)
rf_avg_recall = np.mean(rf_recalls, axis=0)

# 결과 출력
print("Random Forest:")
print(f"  Average F1-score: {rf_avg_f1:.4f}")
print(f"  Average Accuracy: {rf_avg_accuracy:.4f}")
print("  Average Precision:")
for j in range(len(rf_avg_precision)):
    print(f"    Class {j}: {rf_avg_precision[j]:.4f}")
print("  Average Recall:")
for j in range(len(rf_avg_recall)):
    print(f"    Class {j}: {rf_avg_recall[j]:.4f}")
print()
```

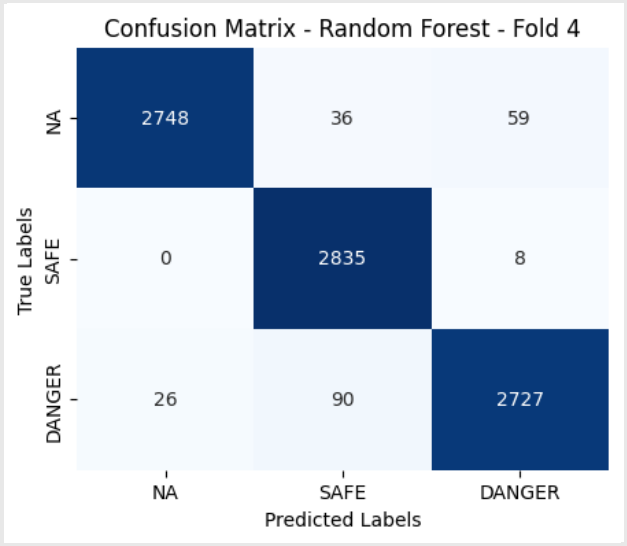
Printing Average of Metrics across folds

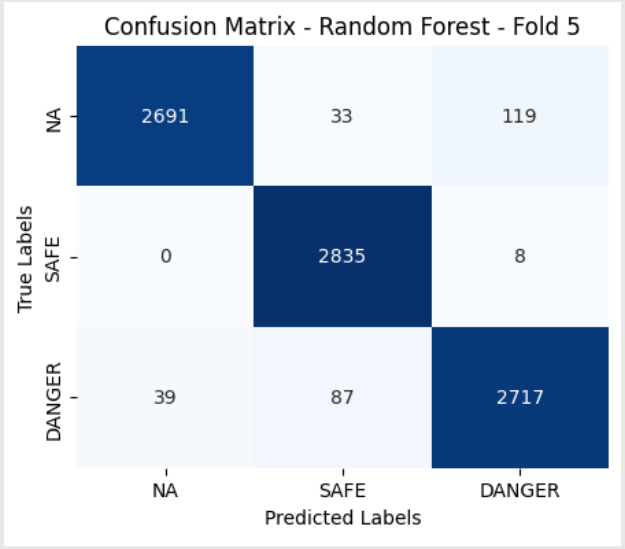
Application / 적용 – Random Forest












```
Random Forest:
  Average F1-score: 0.9679
  Average Accuracy: 0.9679
  Average Precision:
    Class 0: 0.9728
    Class 1: 0.9602
    Class 2: 0.9730
  Average Recall:
    Class 0: 0.9669
    Class 1: 0.9942
    Class 2: 0.9425
```

Application / 적용 – Gradient Boosting

```
k_fold = StratifiedKFold(n_splits=5) # k = 5
```

Stratified k-fold

```
gb_f1_scores = []
gb_accuracies = []
gb_precisions = []
gb_recalls = []

for fold, (train_idx, test_idx) in enumerate(k_fold.split(X_train_over, y_train_over), 1):
    print(f'Fold {fold}:')

    x_train, y_train = X_train_over[train_idx], y_train_over[train_idx]
    x_test, y_test = X_train_over[test_idx], y_train_over[test_idx]

    scaler = StandardScaler()
    x_train = scaler.fit_transform(x_train)
    x_test = scaler.transform(x_test)

    # Gradient Boosting Classifier
    gb = GradientBoostingClassifier()
    gb.fit(x_train, y_train)
    y_pred_gb = gb.predict(x_test)
```

Modeling

```
# Confusion Matrix 시각화
plt.figure(figsize=(5, 4))
sns.heatmap(confusion_matrix(y_test, y_pred_gb), annot=True, cmap='Greens', fmt='d', cbar=False,
            xticklabels=['NA', 'SAFE', 'DANGER'], yticklabels=['NA', 'SAFE', 'DANGER'])
plt.title(f"Confusion Matrix - Gradient Boosting - Fold {fold}")
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

Confusion Matrix

Application / 적용 – Gradient Boosting

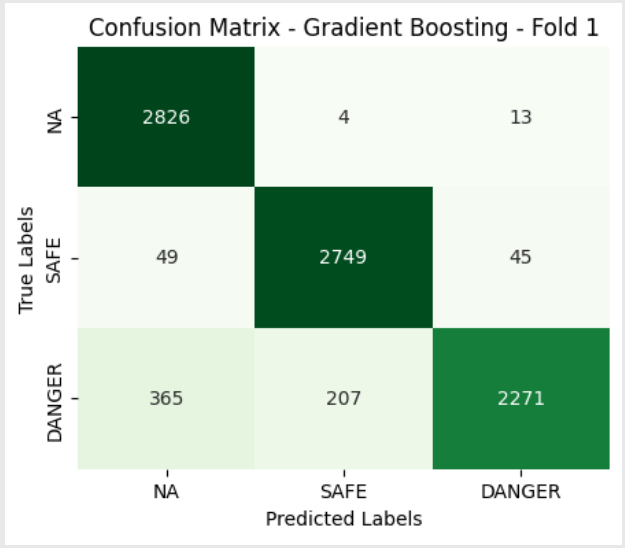
```
    # 각 fold에서 f1-score를 출력
    print(f"Fold {fold}:")
    print(f"  F1-score: {gb_f1:.4f}")
    print(f"  Accuracy: {gb_accuracy:.4f}")
    print(f"  Precision:")
    for j in range(len(gb_precision)):
        print(f"    Class {j}: {gb_precision[j]:.4f}")
    print(f"  Recall:")
    for j in range(len(gb_recall)):
        print(f"    Class {j}: {gb_recall[j]:.4f}")
    print()
```

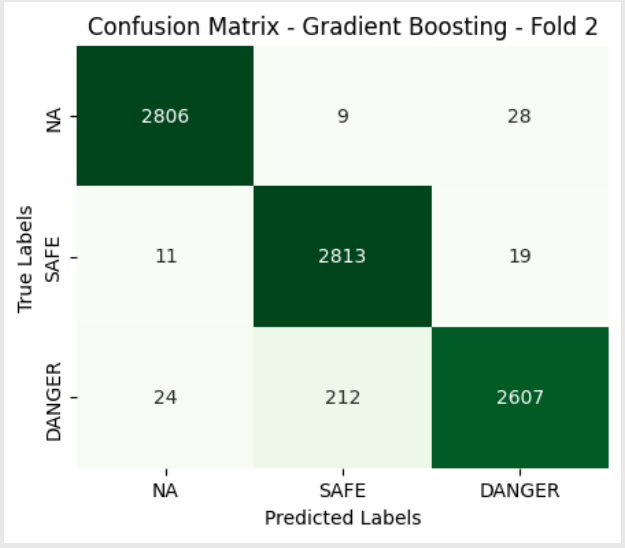
Printing Metrics for each fold

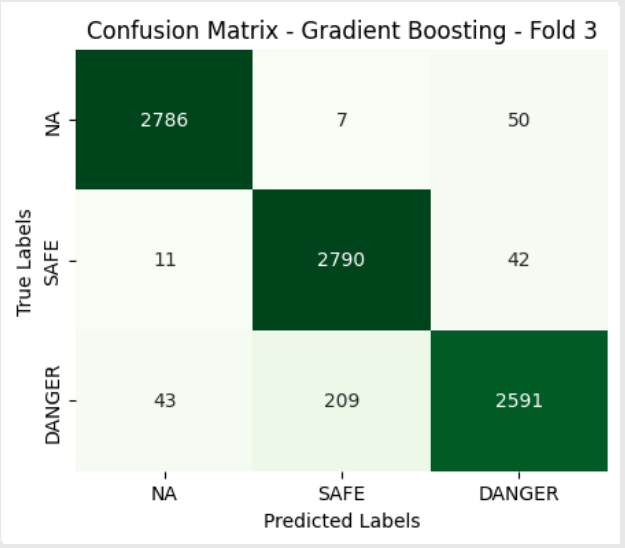
```
# 평균 F1-score, Accuracy, Precision, Recall 계산 및 출력

gb_avg_f1 = np.mean(gb_f1_scores)
gb_avg_accuracy = np.mean(gb_accuracies)
gb_avg_precision = np.mean(gb_precisions, axis=0)
gb_avg_recall = np.mean(gb_recalls, axis=0)

# 결과 출력
print("Gradient Boosting:")
print(f"  Average F1-score: {gb_avg_f1:.4f}")
print(f"  Average Accuracy: {gb_avg_accuracy:.4f}")
print("  Average Precision:")
for j in range(len(gb_avg_precision)):
    print(f"    Class {j}: {gb_avg_precision[j]:.4f}")
print("  Average Recall:")
for j in range(len(gb_avg_recall)):
    print(f"    Class {j}: {gb_avg_recall[j]:.4f}")
print()
```

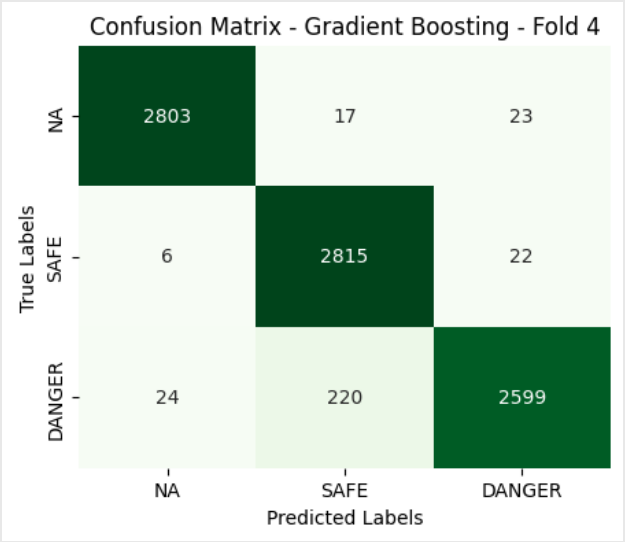
Printing Average of Metrics across folds

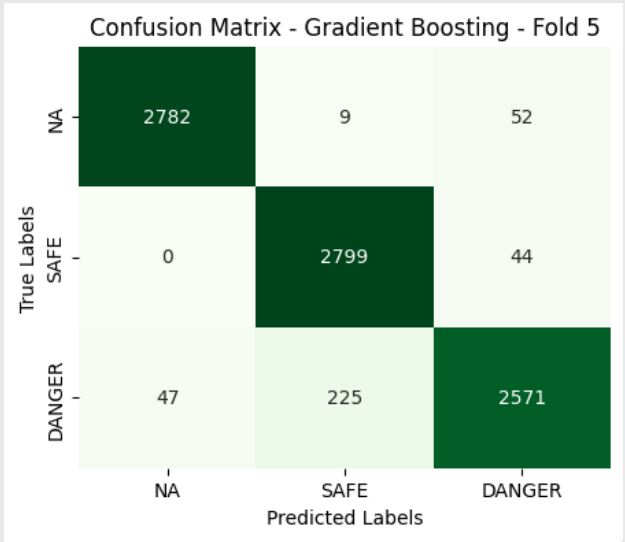
Application / 적용 – Gradient Boosting












```
Gradient Boosting:
  Average F1-score: 0.9522
  Average Accuracy: 0.9522
  Average Precision:
    Class 0: 0.9627
    Class 1: 0.9258
    Class 2: 0.9740
  Average Recall:
    Class 0: 0.9851
    Class 1: 0.9825
    Class 2: 0.8891
```

Application / 적용 –XGBoosting

```
k_fold = StratifiedKFold(n_splits=5) # k = 5

xgb_f1_scores = []
xgb_accuracies = []
xgb_precisions = []
xgb_recalls = []

for fold, (train_idx, test_idx) in enumerate(k_fold.split(X_train_over, y_train_over), 1):
    print(f'Fold {fold}:')

    x_train, y_train = X_train_over[train_idx], y_train_over[train_idx]
    x_test, y_test = X_train_over[test_idx], y_train_over[test_idx]

    scaler = StandardScaler()
    x_train = scaler.fit_transform(x_train)
    x_test = scaler.transform(x_test)

    # XGBoost Classifier
    xgb = XGBClassifier()
    xgb.fit(x_train, y_train)
    y_pred_xgb = xgb.predict(x_test)
```

Stratified k-fold

Modeling

```
# Confusion Matrix 시각화
plt.figure(figsize=(5, 4))
sns.heatmap(confusion_matrix(y_test, y_pred_xgb), annot=True, cmap='Reds', fmt='d', cbar=False,
            xticklabels=['NA', 'SAFE', 'DANGER'], yticklabels=['NA', 'SAFE', 'DANGER'])
plt.title(f"Confusion Matrix - XGBoost - Fold {fold}")
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

Confusion Matrix

Application / 적용 –XGBoosting

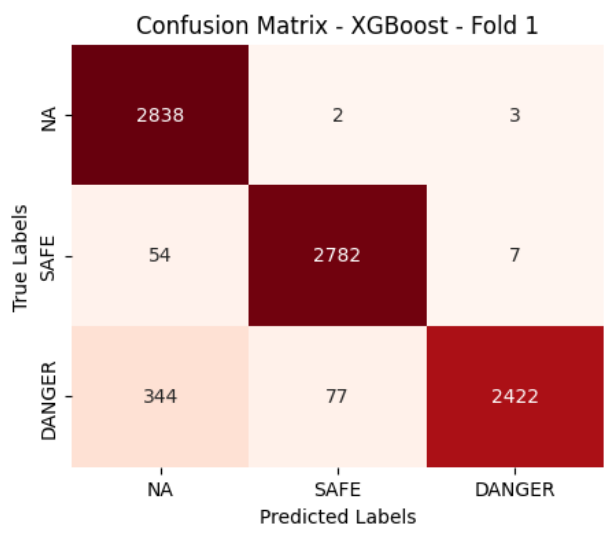
```
    # 각 fold에서 f1-score를 출력
    print(f"Fold {fold}:")
    print(f"  F1-score: {xgb_f1:.4f}")
    print(f"  Accuracy: {xgb_accuracy:.4f}")
    print(f"  Precision:")
    for j in range(len(xgb_precision)):
        print(f"    Class {j}: {xgb_precision[j]:.4f}")
    print(f"  Recall:")
    for j in range(len(xgb_recall)):
        print(f"    Class {j}: {xgb_recall[j]:.4f}")
    print()
```

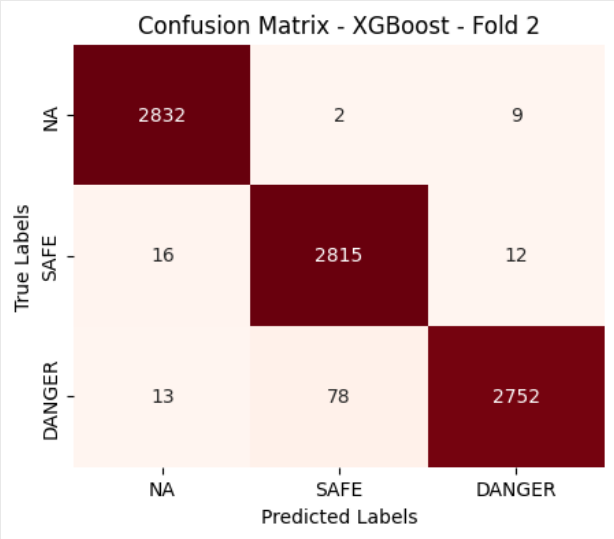
Printing Metrics for each fold

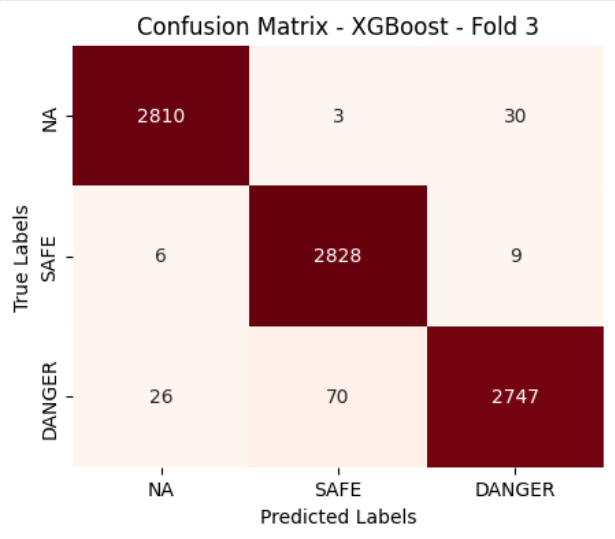
```
# 평균 F1-score, Accuracy, Precision, Recall 계산 및 출력
xgb_avg_f1 = np.mean(xgb_f1_scores)
xgb_avg_accuracy = np.mean(xgb_accuracies)
xgb_avg_precision = np.mean(xgb_precisions, axis=0)
xgb_avg_recall = np.mean(xgb_recalls, axis=0)

# 결과 출력
print("XGBoost:")
print(f"  Average F1-score: {xgb_avg_f1:.4f}")
print(f"  Average Accuracy: {xgb_avg_accuracy:.4f}")
print("  Average Precision:")
for j in range(len(xgb_avg_precision)):
    print(f"    Class {j}: {xgb_avg_precision[j]:.4f}")
print("  Average Recall:")
for j in range(len(xgb_avg_recall)):
    print(f"    Class {j}: {xgb_avg_recall[j]:.4f}")
print()
```

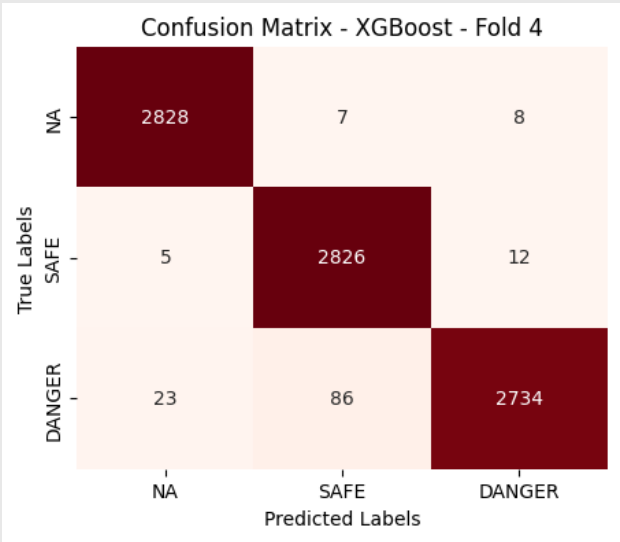
Printing Average of Metrics across folds

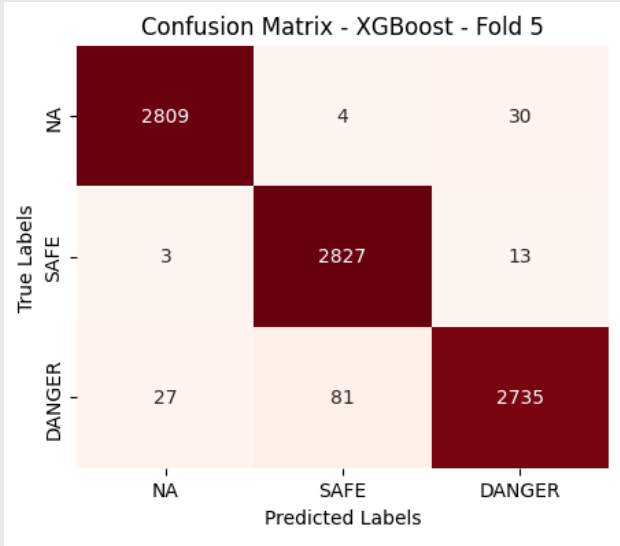
Application / 적용 –XGBoosting












```
XGBoost:
  Average F1-score: 0.9751
  Average Accuracy: 0.9751
  Average Precision:
    Class 0: 0.9670
    Class 1: 0.9717
    Class 2: 0.9903
  Average Recall:
    Class 0: 0.9931
    Class 1: 0.9904
    Class 2: 0.9420
```

Conclusion / 결론

1. Model Performance

Single Decision Tree << Ensemble Models

Comparing F1-score

Decision Tree	Random Forest	Gradient Boosting	XGBoosting
0.8704	0.9679	0.9522	0.9751

Gradient Boosting, XGBoosting took long time calculating

Conclusion / 결론

2. Further Idea Proposal

Our Goal :

- **Society More aware of the danger of Myocardial Infarction**
- **Manage Disease & Live Healthier**

How it works ? :

- **Users take blood and our system analyze it**
- **If the danger rate is high, it will be notified to the user**

Developments / 발전가능성

1. Using Various Hyperparameter :
   - To prevent overfitting
   - Advancing model usage

2. Evaluation Metrics were very high.
   - Need to check for overfitting

3. Hard to view danger rate in models
   - Considering Interpretability of ensemble tree-based models
   - Don’t know if the danger rate is calculated properly

References / 참고문헌

[Medical References]

- 서울대학교병원 N 의학정보; 심근경색 (http://www.snuh.org/health/nMedInfo/nView.do?category=DIS&medid=AA000335)
- 서울아산병원; 급성 심근경색증 (https://www.amc.seoul.kr/asan/healthinfo/disease/diseaseDetail.do?contentId=32111)
- 국민건강영양조사 제 8기 Raw Data, 이용지침서 (https://knhanes.kdca.go.kr/knhanes/main.do)
- Child's Heart Disease (https://m.post.naver.com/viewer/postView.nhn?volumeNo=15348174&memberNo=38360564)
- 서울아산병원; 혈색소 (https://www.amc.seoul.kr/asan/mobile/healthinfo/management/managementDetail.do?managementId=119)

References / 참고문헌

[Basic Coding References]

- Bringing SAS data into Python (https://rsas.tistory.com/m/368)
- Machine Learning Model Application; Course Materials of 'Data Science Foundation' Lab 03
- Data Preprocessing; Course Materials of 'Data Science Foundation' Data Preprocessing

[Model References – Decision Tree]

- https://ko.wikipedia.org/wiki/%EA%B2%B0%EC%A0%95_%ED%8A%B8%EB%A6%AC_%ED%95%99%EC%8A%B5%EB%B2%95
- https://themainstreamseer.blogspot.com/2013/01/introduction-to-classification.html
- National Library of Medicine (https://pubmed.ncbi.nlm.nih.gov/12182209/)

References / 참고문헌

[Model References – Ensemble Learning]

- https://zephyrus1111.tistory.com/245
- https://libertegrace.tistory.com/entry/Classification-2-%EC%95%99%EC%83%81%EB%B8%94-%ED%95%99%EC%8A%B5Ensemble-Learning-Voting%EA%B3%BC-Bagging
- https://data-analysis-science.tistory.com/61
- https://encord.com/blog/what-is-ensemble-learning/

[Model References – Random Forest Classifier]

- https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/

References / 참고문헌

[Model References – Gradient Boost Classifier]

- https://zephyrus1111.tistory.com/224
- https://www.robotstory.co.kr/raspberry/?board_page=5&vid=63
- https://velog.io/@regista/%EB%B9%84%EC%9A%A9%ED%95%A8%EC%88%98Cost-Function-%EC%86%90%EC%8B%A4%ED%95%A8%EC%88%98Loss-function-%EB%AA%A9%EC%A0%81%ED%95%A8%EC%88%98Objective-Function-Ai-tech

[Model References – XGBoost Classifier]

- https://zephyrus1111.tistory.com/232
- https://docs.aws.amazon.com/ko_kr/sagemaker/latest/dg/xgboost-HowItWorks.html
- https://nanunzoey.tistory.com/entry/%EA%B3%BC%EC%A0%81%ED%95%A9Overfitting%EA%B3%BC-%EA%B7%9C%EC%A0%9CRegularization

References / 참고문헌

[Picture References]

- Flaticon (https://www.flaticon.com/)
- Blood Clotting (https://www.news-medical.net/health/Blood-Clotting-Process.aspx)
- XGBoosting (https://www.researchgate.net/figure/XGBoost-model-Source-Self_fig2_350874464)