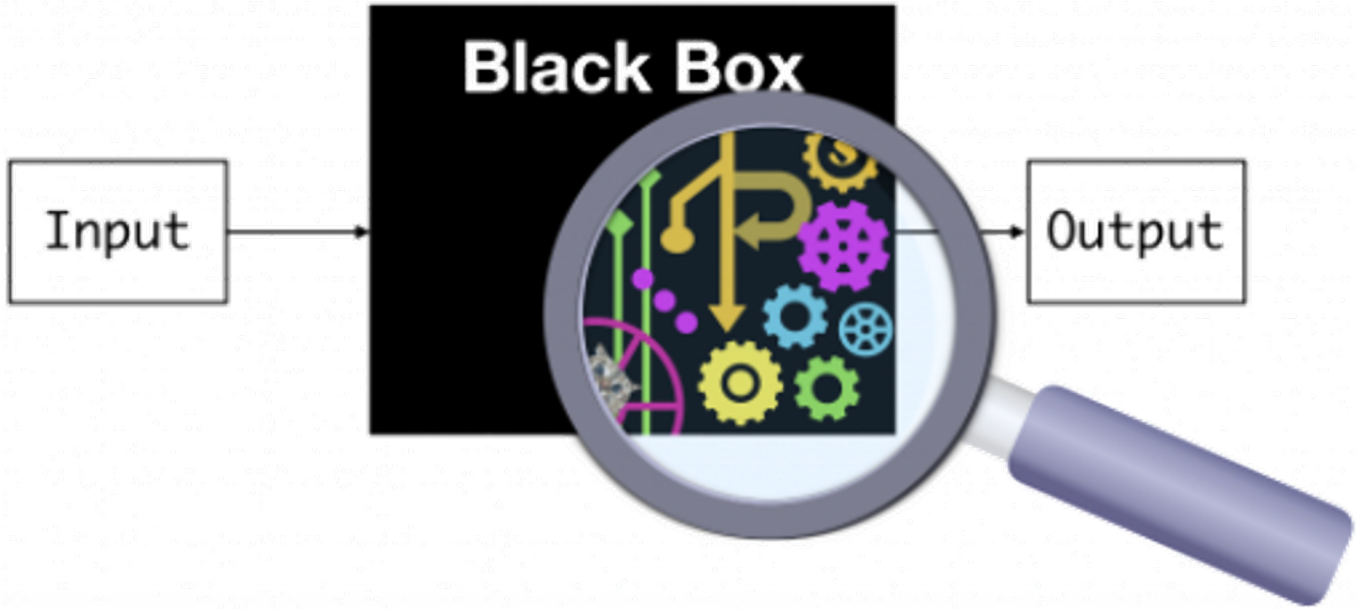
Explainable AI

7th December

UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

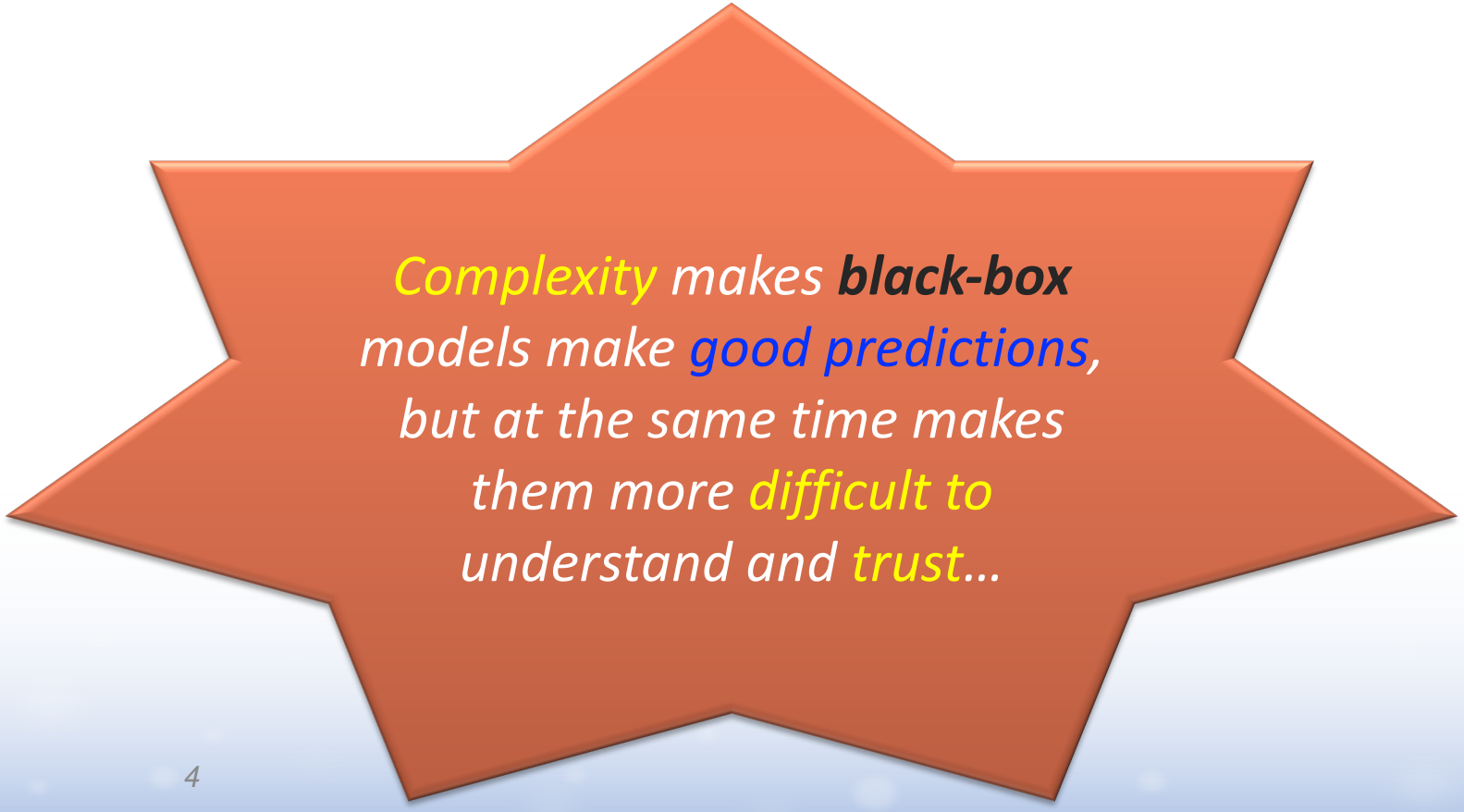


Black Box visual

Artificial Intelligence is BLACK BOX..?

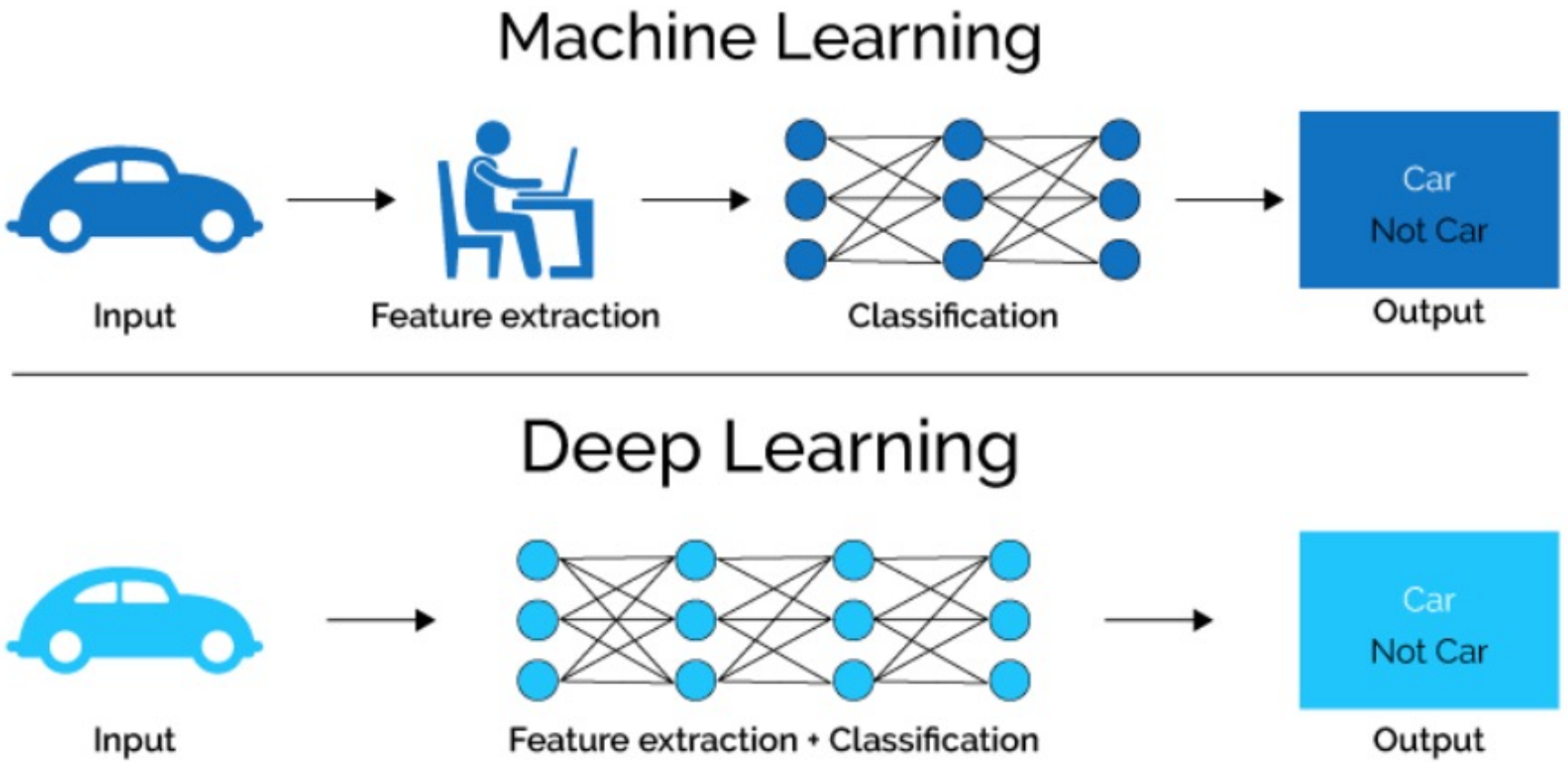
- With the recent development of machine learning and artificial intelligence (AI) technologies, models have become very complex, such as complex deep neural networks (DNNs) and various models being combined.
- That process is largely self-directed and is generally difficult for data scientists, programmers and users to interpret.
- Such complex machine learning models are called black-box models.

- Complexity makes black-box models make good predictions, but at the same time makes them more difficult to understand and trust.
  - The individual algorithms inside the black box model do not reveal their secrets.



Why you need interpretability in addition to accuracy

1. Data scientists want to build models with high accuracy.
2. End users want to know why the model made a particular prediction.
   - Transparency: The system can explain how the model behaves and why it made certain predictions.
   - Clarity (explainability): A system can communicate useful information about its inner workings, such as the patterns a model learns and the results it provides.



https://www.xenonstack.com/blog/log-analytics-deep-machine-learning

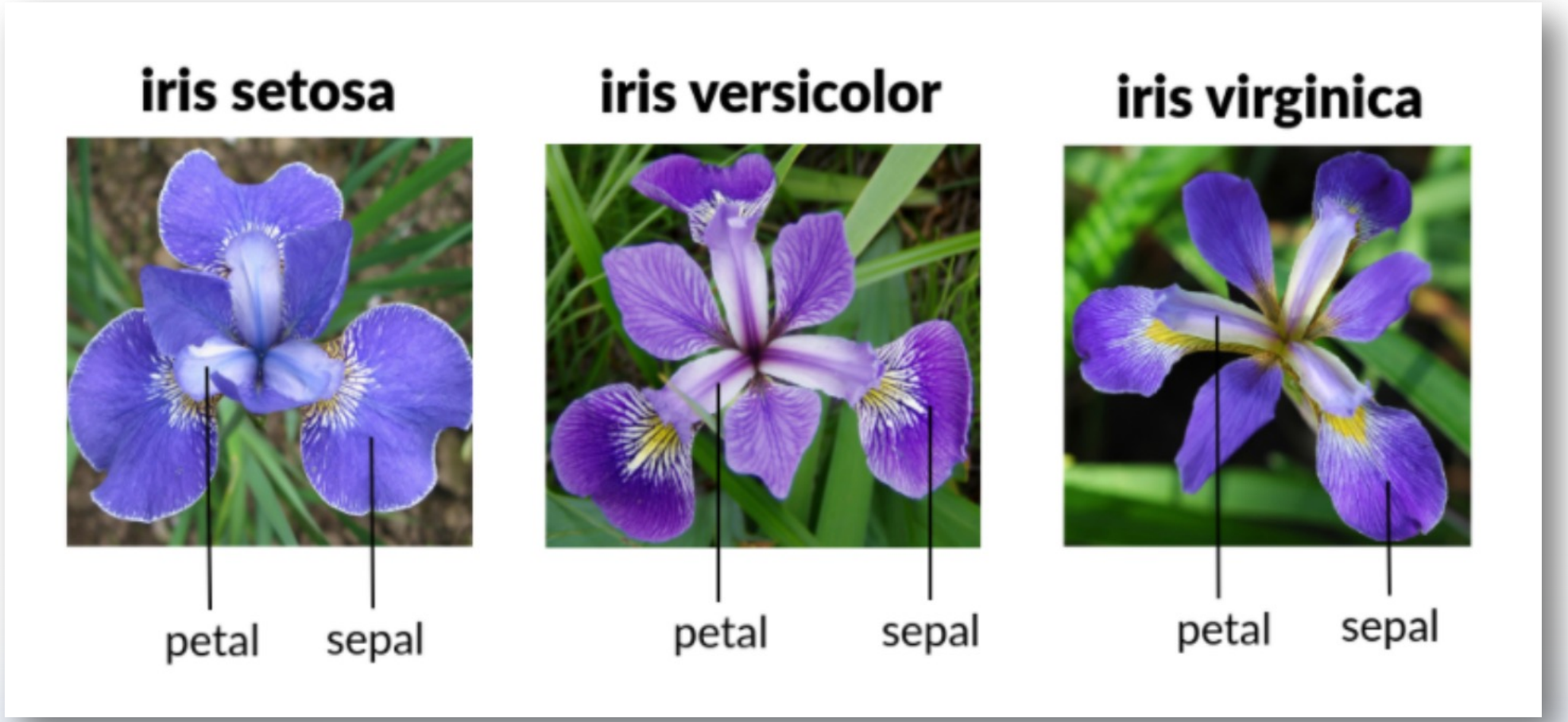
XGBoost: A Scalable Tree Boosting System

- XGBoost is one of many machine learning methods. It is an ensemble method based on a decision tree and a machine learning method based on boosting.
- Getting started with XGBoost
  - The first thing we want to do is install the library which is most easily done via pip. (ex: pip3 install xgboost)

XGBoost: A Scalable Tree Boosting System

- Setting up input data with XGBoost
  - Import Scikit Learn library to get today's input data set.
    - The iris data are data based on statistician R.A Fisher's classification study of irises.
    - The data is structured as follows.
      - ✓ Target data: setosa, versicolor, virginica (3 species or iris)
      - ✓ Feature data: Sepal length, Sepal width, Petal Length, Petal Width

Three species of iris



Setting up input data with XGBoost

- Setting up input data with XGBoost

```
from sklearn import datasets
import xgboost as xgb

iris = datasets.load_iris()
X = iris.data
y = iris.target
```

```
X[:5]
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

```
y[:4]
```

```
array([0, 0, 0, 0])
```

About scikit-learn..

- scikit-learn is a machine learning library that represents python.
- It provides functions that can be applied to various machine learning algorithms such as classification, regression, clustering, and decision trees.
- scikit-learn includes sample datasets to make learning machine learning easy.

Data sets provided by scikit-learn

- load_iris: iris flower data
- load_boston: Boston house price data
- load_diabetes: Diabetes patient data
- load_wine: Wine data
- load_breast_cancer: Wisconsin Breast Cancer Patient Data

Data structure of scikit-learn

- The built-in dataset is composed of a key-value format and has a structure similar to that of a dictionary type.
- Common keys are:
  - Data: Sample data, Numpy array
  - Target: Label data, Numpy array
  - Feature_names: Name of feature data
  - Target_names: Name of label data
  - DESCR: description of data set
  - Filename: Locations of data file (csv)

```
from sklearn import datasets
import xgboost as xgb

iris = datasets.load_iris()
X = iris.data
y = iris.target
```

```
X[:5]
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

```
y[:4]
```

```
array([0, 0, 0, 0])
```

Setting up input data with XGBoost

- Split iris dataset with an 80%-20% for train and test XGBoost model.
- Transform our data into a specific format that XGBoost can handle, named Dmatrix.

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2)
```

```
D_train = xgb.DMatrix(X_train, label=Y_train)
D_test = xgb.DMatrix(X_test, label=Y_test)
```

Defining an XGBoost model

- *max_depth* : maximum depth of the decision trees being trained
- *objective:* the loss function being used
- *num_class:* the number of classes in the dataset

```
param = {
    'eta': 0.3,
    'max_depth': 3,
    'objective': 'multi:softprob',
    'num_class': 3}

steps = 20  # The number of training iterations
```

Training and Testing

```
model = xgb.train(param, D_train, steps)

import numpy as np
from sklearn.metrics import precision_score, recall_score, accuracy_score

preds = model.predict(D_test)
best_preds = np.asarray([np.argmax(line) for line in preds])

print("Precision = {}".format(precision_score(Y_test, best_preds, average='macro')))
print("Recall = {}".format(recall_score(Y_test, best_preds, average='macro')))
print("Accuracy = {}".format(accuracy_score(Y_test, best_preds)))
```

```
Precision = 0.9555555555555556
Recall = 0.9259259259259259
Accuracy = 0.9333333333333333
```

MODIFICATIONS..!

Target & Input Data

```
[ Mode 1: Label & Value Importing ]
# Generating input dataframe (normalized) is successfully finished~!
          0         1    2    3         4    5    6    7         8    9  ...  \
0  0.009091  0.006818  0.0  0.0  0.000000  0.0  0.0  0.0  0.000000  0.0  ...
1  0.006818  0.009091  0.0  0.0  0.000000  0.0  0.0  0.0  0.000000  0.0  ...
2  0.018182  0.000000  0.0  0.0  0.000000  0.0  0.0  0.0  0.000000  0.0  ...
3  0.004049  0.001349  0.0  0.0  0.004049  0.0  0.0  0.0  0.002698  0.0  ...
4  0.011364  0.004546  0.0  0.0  0.000000  0.0  0.0  0.0  0.000000  0.0  ...

   3712      3713      3714  3715      3716      3717      3718      3719  \
0   0.0  0.001398  0.000000   0.0  0.000000  0.000000  0.000000  0.005595
1   0.0  0.000000  0.001398   0.0  0.000000  0.000000  0.000000  0.005595
2   0.0  0.000000  0.000000   0.0  0.000000  0.000000  0.000000  0.001398
3   0.0  0.000000  0.000000   0.0  0.002193  0.000000  0.000000  0.000000
4   0.0  0.001398  0.000000   0.0  0.000000  0.002796  0.001398  0.004196

       3720  target
0  0.009791       1
1  0.009791       1
2  0.015385       1
3  0.010965       1
4  0.006993       1

[5 rows x 3722 columns]
```

Target & Input Data

	UUU_UUU	UUC_UUU	UUA_UUU	UUG_UUU	CUU_UUU	CUC_UUU	CUA_UUU	CUG_UUU	AUU
264	0.009091	0.004546	0.0	0.0	0.002273	0.0	0.000000	0.0	0.0C
265	0.009259	0.006945	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0C
266	0.010345	0.000000	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0C
267	0.011364	0.002273	0.0	0.0	0.002273	0.0	0.000000	0.0	0.0C
268	0.011364	0.004546	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0C
...	...	...	...	...	...	...	...	...	
59677	0.004167	0.000000	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0C
59678	0.004167	0.000000	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0C
59679	0.004167	0.000000	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0C
59680	0.005398	0.001349	0.0	0.0	0.002698	0.0	0.001349	0.0	0.0C
59681	0.004546	0.011364	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0C

59418 rows × 3722 columns

...

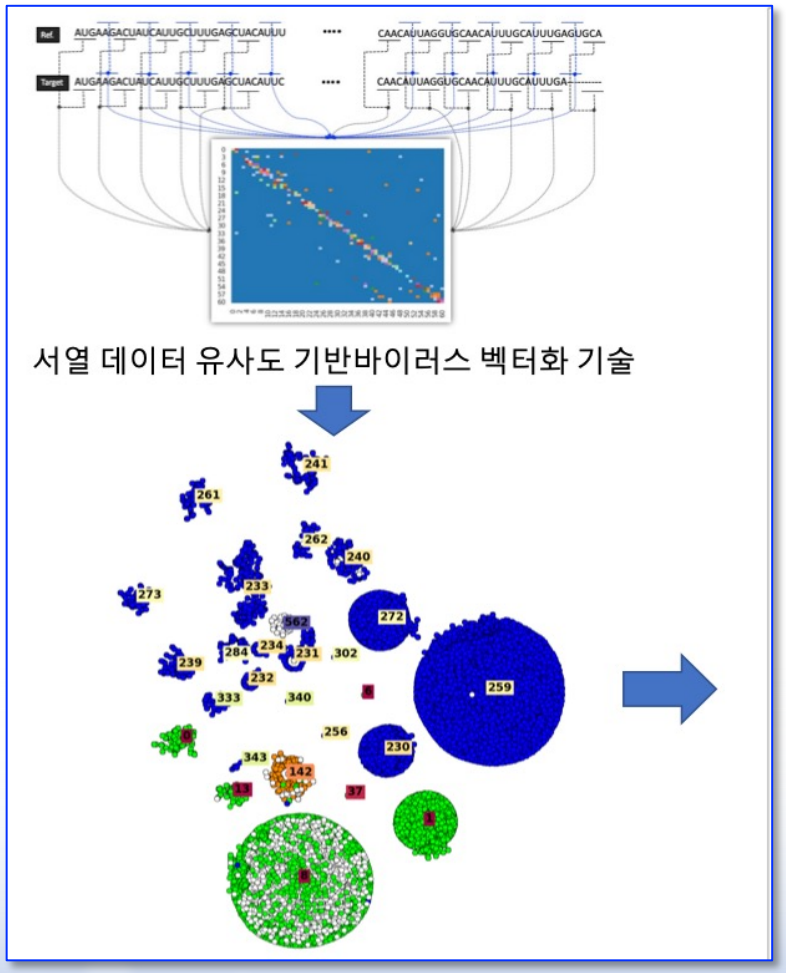
GGA_GGG	GGG_GGG	target
0.006993	0.005595	1
0.005698	0.005698	1
0.003831	0.005747	1
0.008391	0.002796	1
0.008391	0.005595	1
...	...	...
0.000000	0.007143	0
0.000000	0.007143	0
0.000000	0.007143	0
0.000000	0.012532	0
0.005595	0.005595	0

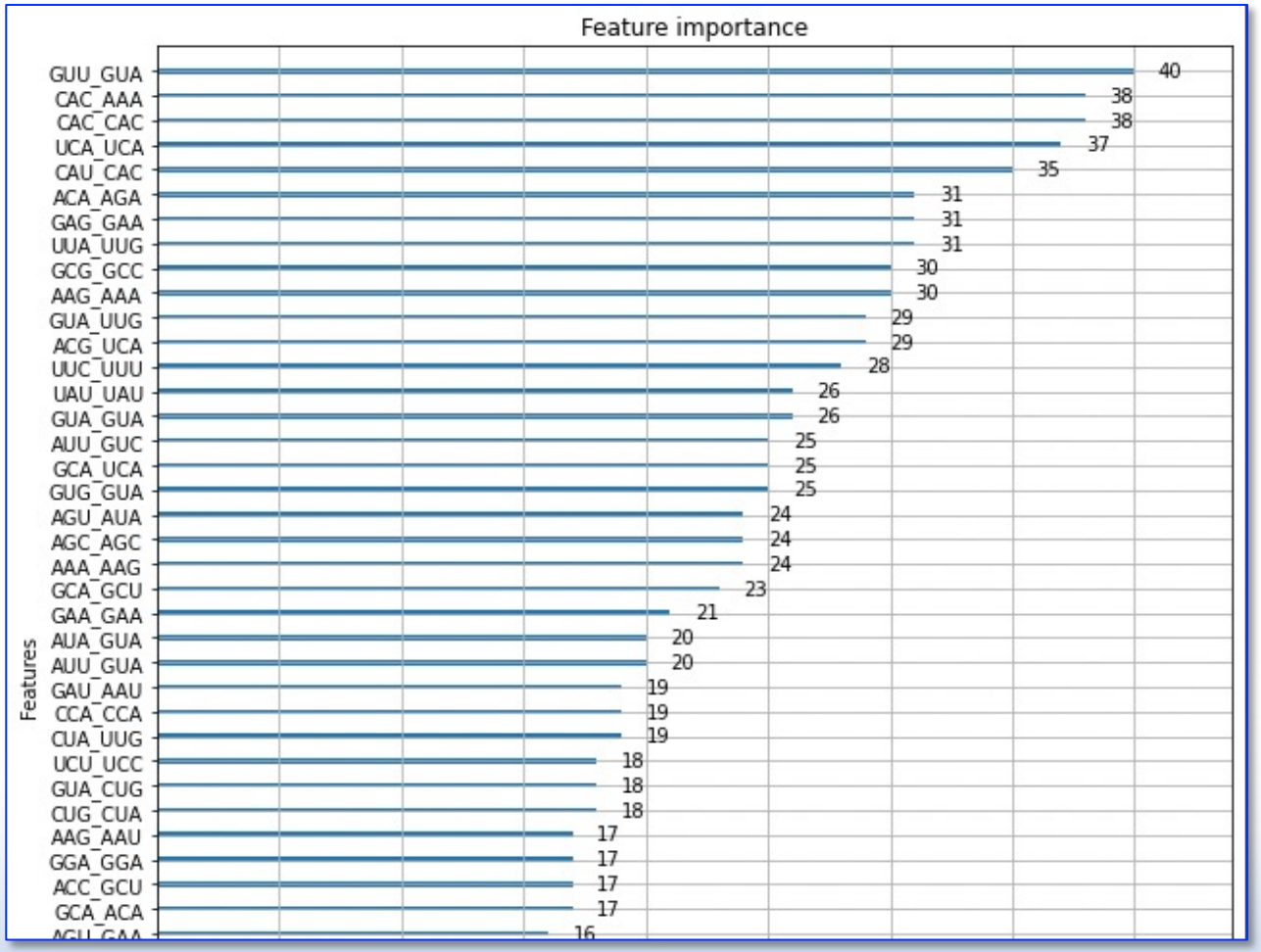
What I want to know is… (example)

- Which codon in each sequence did an important role in predicting the target label..?
  - Label 0: SARS-CoV-1 Delta variants
  - Label 1: SARS-Cov-1 Omicron variants

```
import matplotlib.pyplot as plt
%matplotlib inline

fig, ax = plt.subplots(figsize=(10, 12))
xgb.plot_importance(xgb_model, ax=ax, max_num_features=50)
```





Assignments

Assignments

1. **Try to create** a XGBoost prediction model using IRIS data set.
2. If possible, try to create model using your own data in your major using XGBoost's Feature importance package.
3. 차용준 from KIST and 안재현 from KITECH and 손준호 from KIST, prepare a 5-minute presentation of what these people have done.

THANK YOU