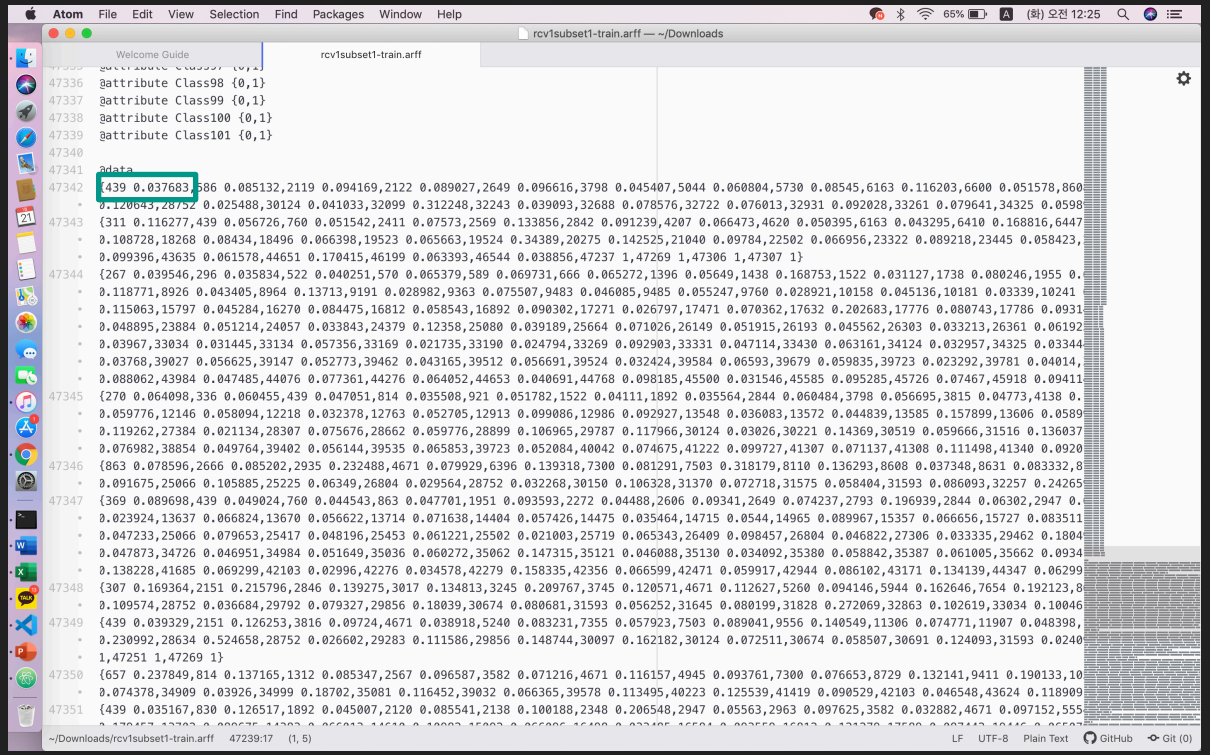
Reuters dataset

Team A
박성환, 남현지, 이고은, 송지현

1.문제분석

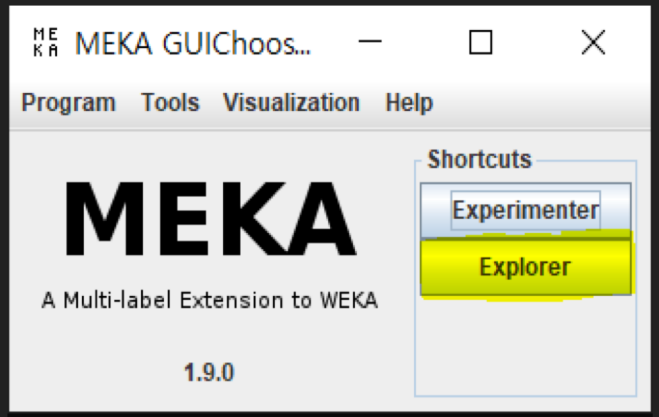


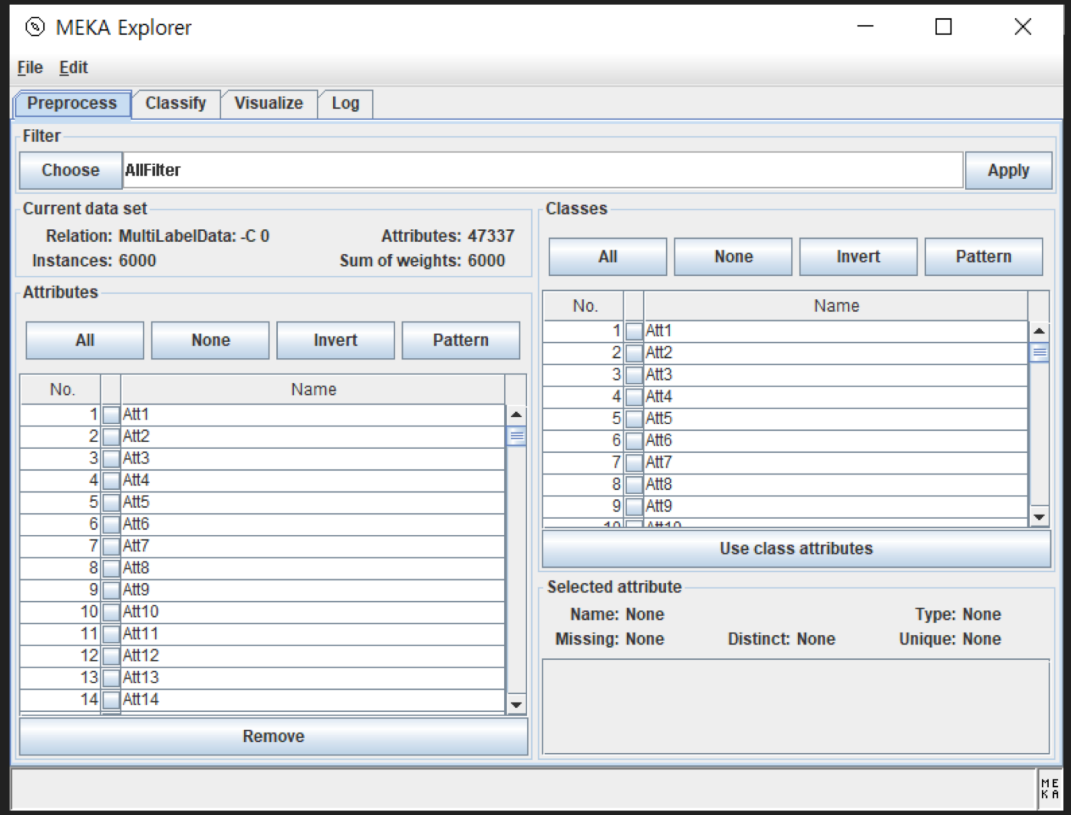
{숫자 (공백) 숫자,
숫자 (공백) 숫자,
.
.
.
}
{숫자 (공백) 숫자,
숫자 (공백) 숫자,
.
.
.
}
.
.
.

	Att1	Att2	Att3	Att4	...	Class98	Class99	Class100	Class101
0	0	0.0	0.0	0.0	...	0	0	0	0
1	0	0.0	0.0	0.0	...	0	0	0	0
2	0	0.0	0.0	0.0	...	0	0	0	0
3	0	0.0	0.0	0.0	...	0	0	0	0
4	0	0.0	0.0	0.0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...
5995	0	0.0	0.0	0.0	...	0	0	0	0
5996	0	0.0	0.0	0.0	...	0	0	0	0
5997	0	0.0	0.0	0.0	...	0	0	0	0
5998	0	0.0	0.0	0.0	...	0	0	0	0
5999	0	0.0	0.0	0.0	...	0	0	0	0

[6000 rows x 47337 columns]

instance : 6000, numeric : 47236, label : 101


MEKA GUIChoos...
Program
Tools
Visualization
Help
MEKA
A Multi-label Extension to WEKA
1.9.0
Shortcuts
Experimenter
Explorer


MEKA Explorer
File
Edit
Preprocess
Classify
Visualize
Log
Filter
Choose
AllFilter
Apply
Current data set
Relation: MultiLabelData: -C 0
Attributes: 47337
Instances: 6000
Sum of weights: 6000
Attributes
All
None
Invert
Pattern
No.
Name
1 Att1
2 Att2
3 Att3
4 Att4
5 Att5
6 Att6
7 Att7
8 Att8
9 Att9
10 Att10
11 Att11
12 Att12
13 Att13
14 Att14
Remove
Classes
All
None
Invert
Pattern
No.
Name
1 Att1
2 Att2
3 Att3
4 Att4
5 Att5
6 Att6
7 Att7
8 Att8
9 Att9
Use class attributes
Selected attribute
Name: None
Type: None
Missing: None
Distinct: None
Unique: None

```python
class_sum_list=[]
for i in range(1,102):
  class_name = "Class"+str(i)
  class_data = pandas_file[class_name]
  class_sum = class_data.sum()
  class_sum_list.append(class_sum)
print(class_sum_list)
print(class_sum_list[4])
```

```
[120, 197, 196, 527, 764, 328, 285, 247, 140,
764
```

```python
top_10_index=[]
for _ in range (10):
  Maxnum = max(class_sum_list)
  Maxnum_index = class_sum_list.index(Maxnum)
  top_10_index.append(Maxnum_index)
  class_sum_list[Maxnum_index]=0
print(top_10_index)
```

```
[32, 33, 70, 69, 4, 98, 59, 3, 97, 58]
```

LogReg

```python
import torch

class CustomDataset(torch.utils.data.Dataset):
  def __init__(self, filename, index):
    data = pd.read_csv(filename, sep=',')
    data = data.to_numpy()
    
    self.x_data=data[:, :47236]
    self.y_data = data[:, 47236+index]
    self.length = len(data)

  def __getitem__(self, index):
    return self.x_data[index], self.y_data[index]

  def __len__(self):
    return self.length
```

```python
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler

train_dataset_1 = CustomDataset('drive/My Drive/rcv1subset1-train.csv', 33)
test_dataset_1 = CustomDataset('drive/My Drive/rcv1subset1-test.csv', 33)

X_tr = train_dataset.x_data
y_tr = train_dataset.y_data
X_ts = test_dataset.x_data
y_ts = test_dataset.y_data
```

```python
from sklearn.metrics import (accuracy_score,precision_score, recall_score, f1_score, log_loss)

normalizer = MinMaxScaler(feature_range=(0,1))
normalizer.fit(X_tr)
X_tr_normalized = normalizer.transform(X_tr)
X_ts_normalized = normalizer.transform(X_ts)

clf = LogisticRegression(max_iter = 5000)
parameters = {'penalty':['l2'],
              'C': [10e-5,10e-3, 10e-2, 10e-1, 10e0, 10e1, 10e2, 10e3, 10e5]}
gridsearch = GridSearchCV(clf, parameters, scoring = 'accuracy', cv = 5)
gridsearch.fit(X_tr_normalized,y_tr)
print(f'gridsearch.best_params_={gridsearch.best_params_}')

best_clf = gridsearch.best_estimator_

y_pred = best_clf.predict(X_ts_normalized)
test_acc = accuracy_score(y_ts, y_pred)
print(f'test_acc={test_acc}')
```

```
gridsearch.best_params_={'C': 1000.0, 'penalty': 'l2'}
test_acc=0.5603333333333333
```

Random Forest

```python
# Instantiate a model object
clf = RandomForestClassifier()

# Set a search range
parameters = {'n_estimators': [100, 150, 200],
              'criterion': ['gini', 'entropy']}
# Find the best hyperparameters using GridSearchCV
gridsearch = GridSearchCV(clf, parameters, scoring = 'accuracy', cv = 5)
gridsearch.fit(X_tr, y_tr)

# Show the best hyperparameters
print(f'gridsearch.best_params_ = {gridsearch.best_params_}')

# The best model is stored in 'best_clf'
best_clf = gridsearch.best_estimator_
best_clf
```

```python
y_pred = best_clf.predict(X_ts)
test_acc = accuracy_score(y_ts, y_pred)
print(f'test_acc = {test_acc}')
```

[32] test_acc = 0.056

[33] test_acc = 0.604

[70] test_acc = 0.780333

[69] test_acc = 0.713666

[4] test_acc = 0.95

[98] test_acc = 0.94866

[59] test_acc = 0.947333

[3] test_acc = 0.834666

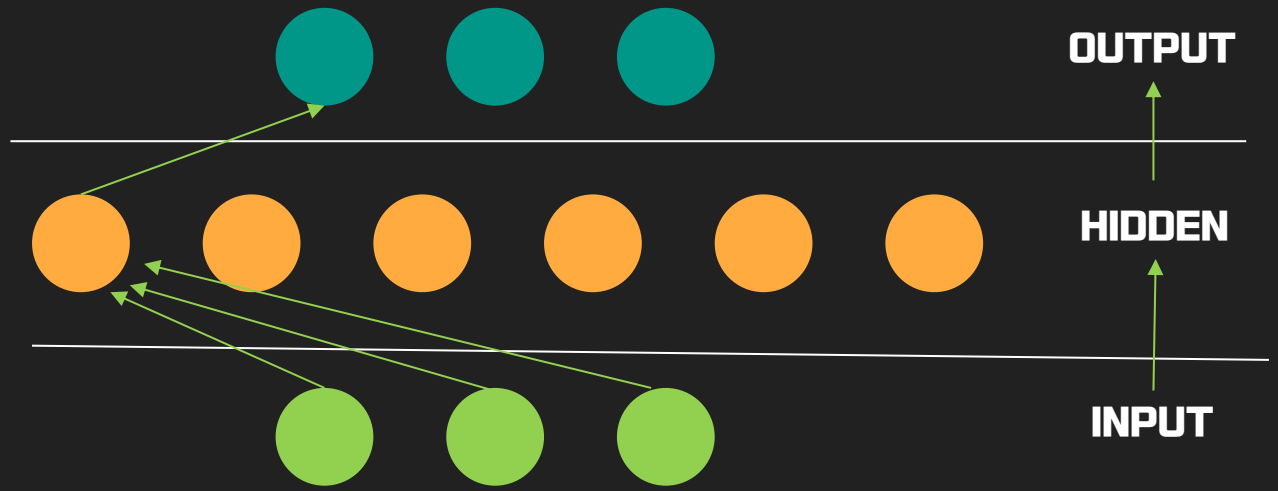
[97] test_acc = 0.878

[58] test_acc = 0.863

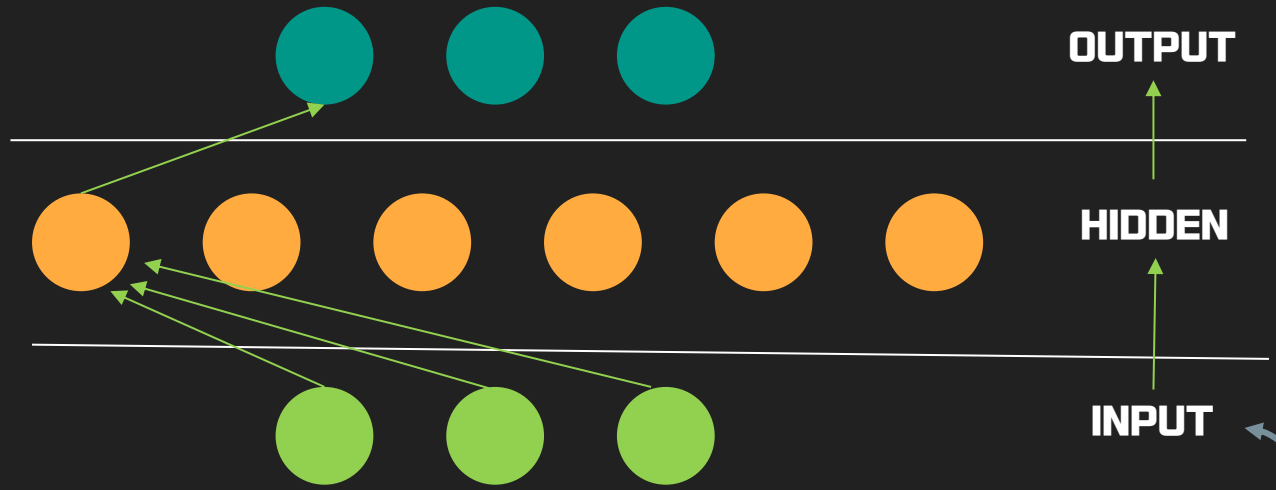
MLP

MLP = multi layer perceptron

MLP = multi layer perceptron

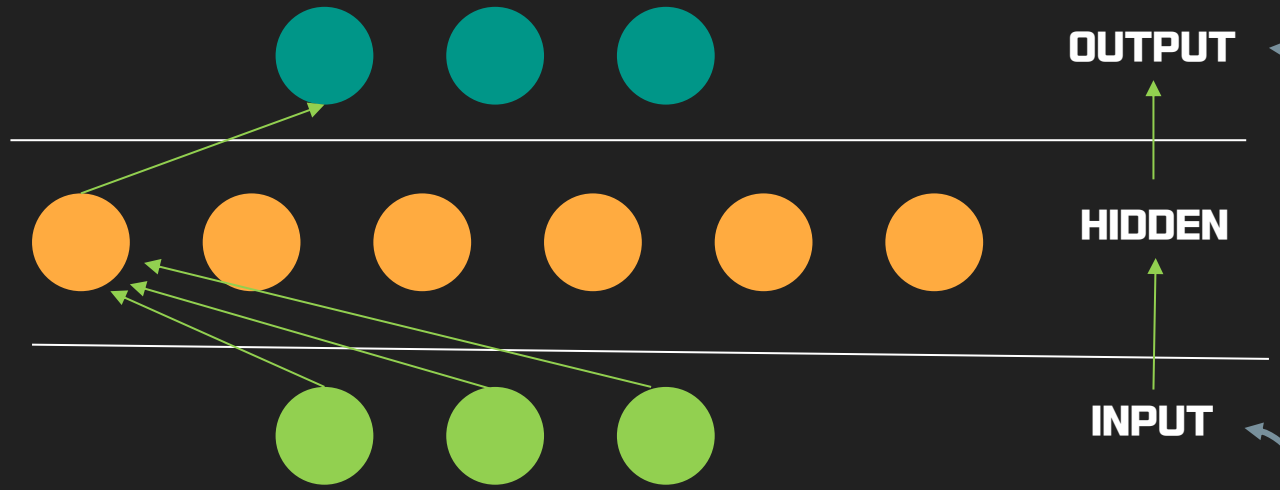


MLP = multi layer perceptron




```
input_size = 47236
num_classes = 2
batch_size = 100
learning_rate = 0.001
```

MLP = multi layer perceptron




```
input_size = 47236
num_classes = 2
batch_size = 100
learning_rate = 0.001
```

input size
x
256
x
128
x
num_classes

```
82 class FFNet(nn.Module):
83   def __init__(self, input_size, num_classes):
84     super(FFNet, self).__init__()
85     self.fc1 = nn.Linear(input_size, 256)
86     self.relu = nn.ReLU()
87     # self.fc2 = nn.Linear(512, 512)
88     # self.fc3 = nn.Linear(512, 256)
89     self.fc4 = nn.Linear(256, 128)
90     self.fc5 = nn.Linear(128, num_classes)
91
92   def forward(self, x):
93     out = self.fc1(x)
94     out = self.relu(out)
95     # out = self.fc2(out)
96     # out = self.relu(out)
97     # out = self.fc3(out)
98     # out = self.relu(out)
99     out = self.fc4(out)
100    out = self.relu(out)
101    out = self.fc5(out)
102    return out
```

```python
class CustomDataset(torch.utils.data.Dataset):

# 우리가 class 생성시에 해줄것은
# self.len 에 길이를 넣어주고
# self.x_data에 x 데이터를
# self.y_data에 y 데이터를 넣어주고
# 인덱스에 따라 i 번째 데이터를 잘 리턴해주면 됩니다.

  def __init__(self, filename, index):
    # 1. initialize file paths or a list of file names
    # 데이터셋의 전처리를 해주는 부분

    data = pd.read_csv(filename, sep=',')
    data = data.to_numpy()

    x_data=data[:, :47236]
    self.x_data = torch.from_numpy(x_data.astype('float'))
    y_data = data[:, 47236+index]
    self.y_data = torch.from_numpy(y_data.astype('int64'))
    self.length = len(data)

  def __getitem__(self, index):
    return self.x_data[index], self.y_data[index]

  def __len__(self):
    return self.length

```

csv.file

dataframe(pandas)

numpy

tensor

Model 1(32)

Accuracy of the network on the 1000 test inputs: 56.03333333333333%

Model 2(33)

Accuracy of the network on the 1000 test inputs: 57.266666666666666%

Model 3(70)

Accuracy of the network on the 1000 test inputs: 75.7%

---

xgboost

: Extreme gradient boosting

DT → RF → Boosting

Tree구조 앙상블 연속 앙상블

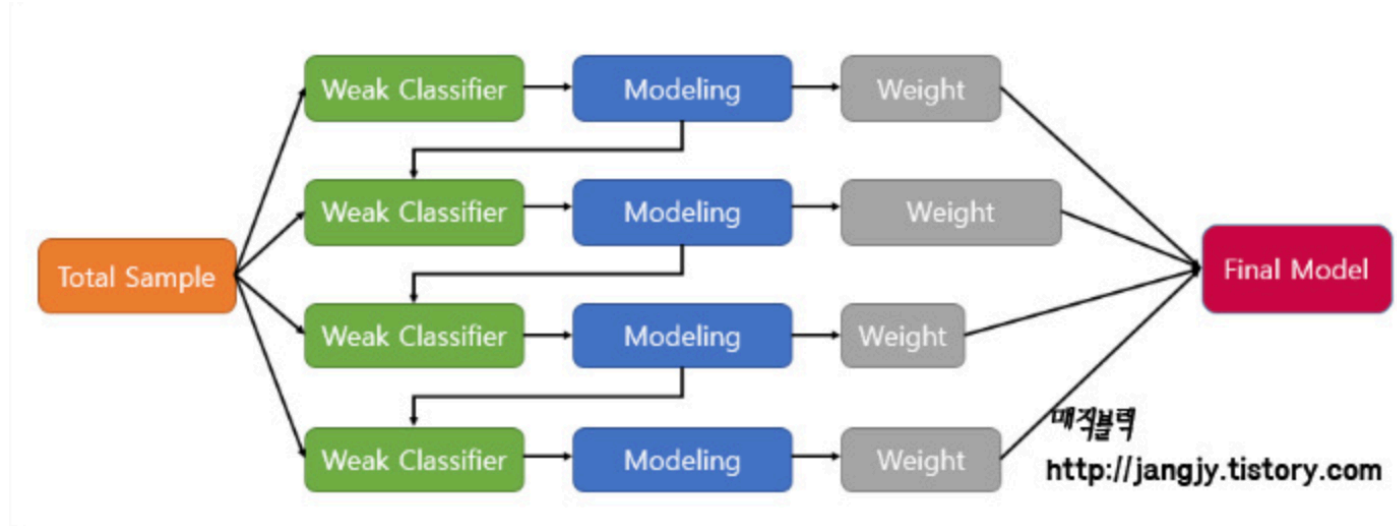


그림 2. boosting의 system flow

```
xg_reg = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.3, learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 10)
```

```
xg_reg.fit(X_tr,y_tr)

preds = xg_reg.predict(X_ts)
```

```
rmse = np.sqrt(mean_squared_error(y_ts, preds))
print("RMSE: %f" % (rmse))
```

```
RMSE: 0.560028
```

[parameter]

n-estimator

max_depth

learning rate