Dongui Bogam

What to include

- Project Proposal
- Updates
- Methods
- Data
- Results & Visualization
- Conclusion (Recap / Takeaways)

Dongui Bogam

Topic | Health Data Analysis

Description | Personal healthcare system managing users' nutritive condition based on their input of food/health supplement intakes.
We expect outputs of the user's current health status and health supplement recommendations

Main Methods | Research, Data Acquisition, Feature Selection, Predictive analysis

Team Member | 신익규, 신승균, 이아현

Updates

Identify the relevant variables

⇓

Clean and process the data

⇓

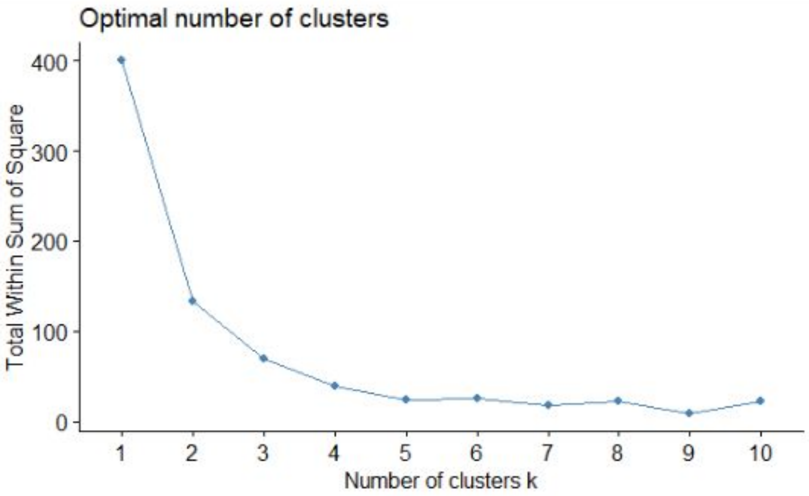
Conduct statistical analysis

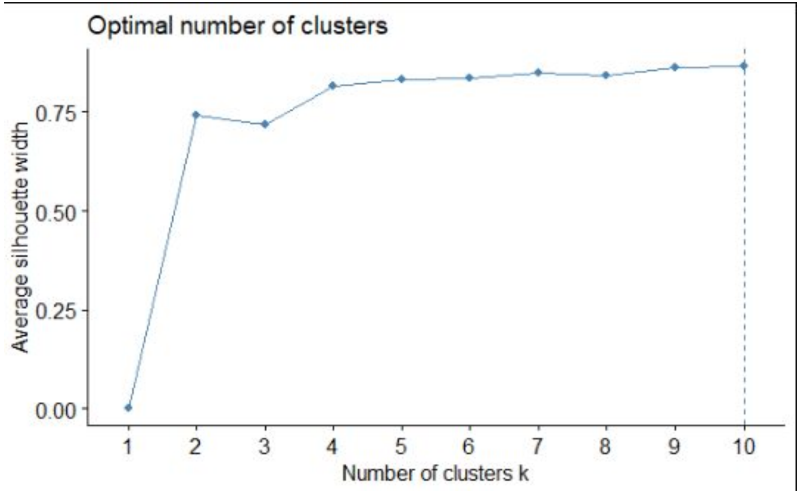
⇓

Develop a risk estimation tool

Back & forth with cleaning and processing the data

Recommendation



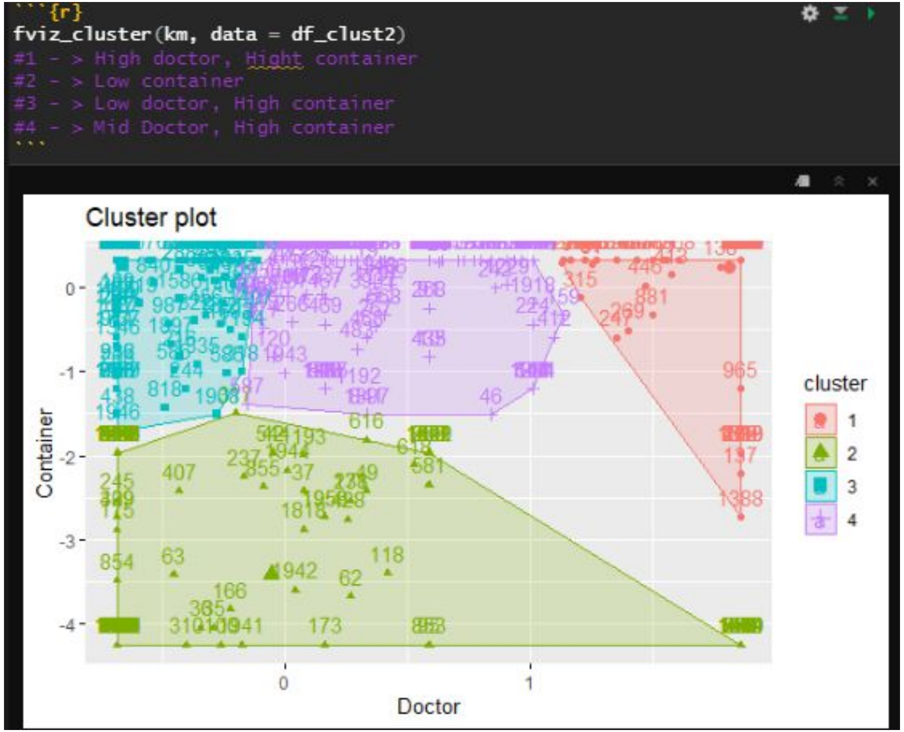


필터링 - 군집화 - 추천 → 군집화 - 필터링 - 추천 순으로 변경

인풋에 따라 달라지는 최적 K 값 방지

그래프에 따라서 최적 K 값은 4로 선택

Recommendation



앞에서 찾은 최적 K = 4로 군집화

Recommendation

```r
#Function to filter data
if (length(Nut) == 1){ #Number of Nutrients need (Multi or Not)
  df_fil = dat_final %>%
  filter(Target == Group,
         Type == Input_type,
         multi == 0) %>%
  filter(`Ing Name` == Nut)
} else{
  df_fil = dat_final %>%
    filter(Target == Group,
           Type == Input_type,
           multi == 1) %>%
    filter(`Ing Name` %in% Nut) %>%
    group_by(ID) %>%
    mutate(N_filter = n()) %>%
    filter(N_filter == length(Nut))

}

clust = dat_final$cluster[dat_final$Name == Nutrients][1]
Final_recom = df_fil %>%
  filter(cluster == clust) %>%
  arrange(desc(N))

Final_recom[1, ]$Name
```
```
[1] "DEFAULT MULTIVITAMIN / MULTIMINERAL"
```

군집 선택 기준 :

기존에 먹는 영양제가 있는 경우,

복용 중인 영양제와 같은 군집에 속하는 영양제 중 **가장 많은 사람들이 찾는** 영양제로 추천

Further Improvement

복용 목적

연령대

성별 등

으로 인풋 세분화 후 보다 정확한 추천

Final Dataset_Liver

Demo

Liver

RDI

```
Data columns (total 13 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   SEQN                        3473 non-null   int64
 1   BMXWT                       3473 non-null   float64
 2   BMXBMI                      3473 non-null   float64
 3   BMXWAIST                    3473 non-null   float64
 4   LBXGH                       3473 non-null   float64
 5   LBXGLU                      3473 non-null   int64
 6   LBXTR                       3473 non-null   int64
 7   LBDLDLN                     3473 non-null   int64
 8   water_soluble_vitamins_sum  3473 non-null   int64
 9   fat_soluble_vitamins_sum    3473 non-null   int64
 10  major_minerals_sum          3473 non-null   int64
 11  trace_minerals_sum          3473 non-null   int64
 12  MCQ160L                     3473 non-null   int64
```

Final Dataset_Cardiovascular

Demo

Cardio Vascular

RDI

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12409 entries, 0 to 12408
Data columns (total 18 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   BMXWT                        12409 non-null  float64
 1   BMXWAIST                     12409 non-null  float64
 2   BMXBMI                       12409 non-null  float64
 3   LBXTC                        12409 non-null  int64
 4   LBXIN                        12409 non-null  float64
 5   LBXGH                        12409 non-null  float64
 6   LBXTR                        12409 non-null  int64
 7   LBDLDLN                      12409 non-null  int64
 8   water_soluble_vitamins_sum   12409 non-null  int64
 9   fat_soluble_vitamins_sum     12409 non-null  int64
 10  major_minerals_sum           12409 non-null  int64
 11  ALQ130                       12409 non-null  int64
 12  SMQ                          12409 non-null  int64
 13  BPXOPLS                      12409 non-null  float64
 14  DIQ010                       12409 non-null  int64
 15  PAQ706                       12409 non-null  float64
 16  URXUMS                       12409 non-null  float64
 17  HEART                        12409 non-null  float64
dtypes: float64(9), int64(9)
memory usage: 1.7 MB
```

Final Dataset_Liver

	BMXWT	BMXBMI	BMXWAIST	LBXGH	LBXGLU	LBXTR	LBDLDLN	water_soluble_vitamins_sum	fat_soluble_vitamins_sum	major_minerals_sum	trace_minerals_sum	MCQ160L
count	3473.000000	3473.000000	3473.000000	3473.000000	3473.000000	3473.000000	3473.000000	3473.000000	3473.000000	3473.000000	3473.000000	3473.000000
mean	83.846300	29.951109	101.002966	5.860841	113.148575	107.519436	110.627124	-0.009214	-2.830118	-0.787504	-0.981284	0.050389
std	22.445219	7.366641	17.256111	1.138301	37.434179	70.566097	36.239897	3.258439	1.327849	2.091180	1.139946	0.218777
min	39.600000	15.400000	63.200000	2.800000	47.000000	10.000000	14.000000	-7.000000	-4.000000	-4.000000	-4.000000	0.000000
25%	67.900000	24.800000	88.800000	5.300000	96.000000	60.000000	86.000000	-2.000000	-4.000000	-2.000000	-1.000000	0.000000
50%	80.600000	28.700000	99.500000	5.600000	103.000000	89.000000	107.000000	1.000000	-3.000000	-2.000000	-1.000000	0.000000
75%	96.000000	33.700000	111.800000	6.000000	115.000000	133.000000	133.000000	3.000000	-2.000000	1.000000	0.000000	0.000000
max	210.800000	82.000000	178.000000	14.900000	451.000000	780.000000	359.000000	5.000000	2.000000	4.000000	1.000000	1.000000

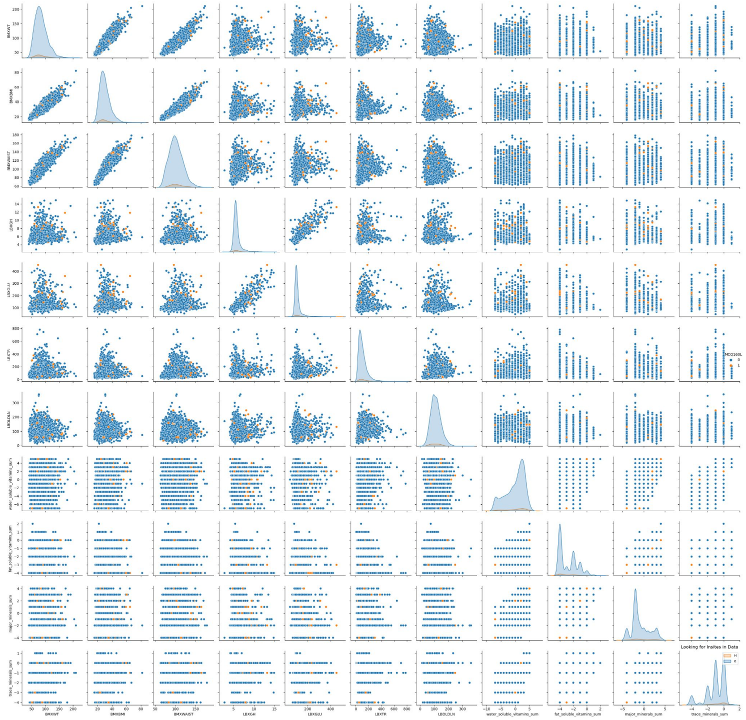
Descriptive Statistics

Final Dataset_Liver

	BMXWT	BMXBMI	BMXWAIST	LBXGH	LBXGLU	LBXTR	LBDLDLN	water_soluble_vitamins_sum	fat_soluble_vitamins_sum	major_minerals_sum	trace_minerals_sum	MCQ160L
BMXWT	1.000000	0.891104	0.898872	0.176365	0.180693	0.157723	-0.007659	0.054901	-0.021464	-0.015674	0.062594	0.008235
BMXBMI	0.891104	1.000000	0.908330	0.209515	0.192484	0.168085	0.006847	-0.047056	-0.030518	-0.070379	-0.034834	0.019597
BMXWAIST	0.898872	0.908330	1.000000	0.266344	0.251934	0.224130	0.012253	-0.021565	-0.047174	-0.068998	0.018617	0.041646
LBXGH	0.176365	0.209515	0.266344	1.000000	0.852273	0.229452	0.004495	-0.045834	-0.039291	-0.062676	0.001963	0.058813
LBXGLU	0.180693	0.192484	0.251934	0.852273	1.000000	0.264493	-0.007970	-0.016657	-0.046051	-0.042145	0.010788	0.067031
LBXTR	0.157723	0.168085	0.224130	0.229452	0.264493	1.000000	0.193415	0.049156	-0.070419	0.008402	0.056733	0.036754
LBDLDLN	-0.007659	0.006847	0.012253	0.004495	-0.007970	0.193415	1.000000	-0.041201	0.005369	-0.035758	-0.001121	-0.006239
water_soluble_vitamins_sum	0.054901	-0.047056	-0.021565	-0.045834	-0.016657	0.049156	-0.041201	1.000000	0.379064	0.659722	0.716439	0.002268
fat_soluble_vitamins_sum	-0.021464	-0.030518	-0.047174	-0.039291	-0.046051	-0.070419	0.005369	0.379064	1.000000	0.506032	0.261243	-0.010638
major_minerals_sum	-0.015674	-0.070379	-0.068998	-0.062676	-0.042145	0.008402	-0.035758	0.659722	0.506032	1.000000	0.541787	0.045210
trace_minerals_sum	0.062594	-0.034834	0.018617	0.001963	0.010788	0.056733	-0.001121	0.716439	0.261243	0.541787	1.000000	0.035483
MCQ160L	0.008235	0.019597	0.041646	0.058813	0.067031	0.036754	-0.006239	0.002268	-0.010638	0.045210	0.035483	1.000000

Correlation Matrix

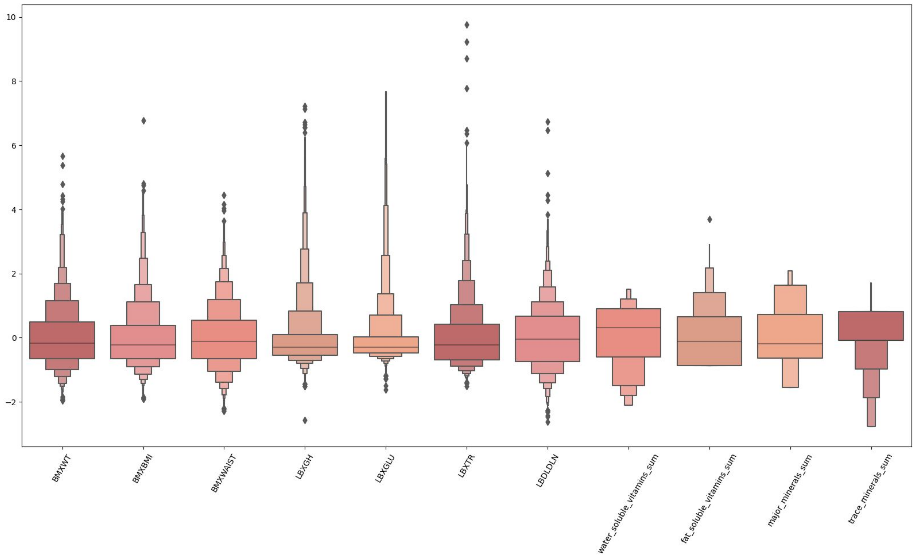
Final Dataset



Pairplot

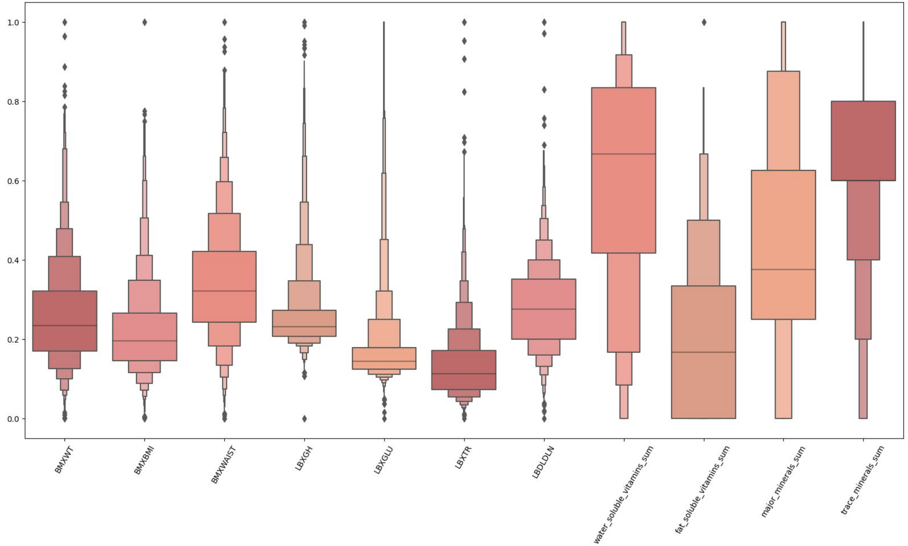
Methods - Standard Scaler

	count	mean	std	min	25%	50%	75%	max
BMXWT	6596.0	0.257500	0.131340	0.0	0.169977	0.233645	0.321262	1.0
BMXBMI	6596.0	0.220966	0.115194	0.0	0.145646	0.195195	0.265766	1.0
BMXWAIST	6596.0	0.339415	0.148396	0.0	0.242160	0.320557	0.420732	1.0
LBXGH	6596.0	0.262830	0.102255	0.0	0.206612	0.231405	0.272727	1.0
LBXGLU	6596.0	0.175264	0.107518	0.0	0.123762	0.143564	0.178218	1.0
LBXTR	6596.0	0.133890	0.088713	0.0	0.072727	0.112987	0.171429	1.0
LBDLDLN	6596.0	0.280341	0.106965	0.0	0.200000	0.275362	0.350725	1.0
water_soluble_vitamins_sum	6596.0	0.583245	0.276465	0.0	0.416667	0.666667	0.833333	1.0
fat_soluble_vitamins_sum	6596.0	0.191480	0.219546	0.0	0.000000	0.166667	0.333333	1.0
major_minerals_sum	6596.0	0.425921	0.274985	0.0	0.250000	0.375000	0.625000	1.0
trace_minerals_sum	6596.0	0.619618	0.224037	0.0	0.600000	0.600000	0.800000	1.0



Methods - MinMax Scaler

	count	mean	std	min	25%	50%	75%	max
BMXWT	6596.0	0.257500	0.131340	0.0	0.169977	0.233645	0.321262	1.0
BMXBMI	6596.0	0.220966	0.115194	0.0	0.145646	0.195195	0.265766	1.0
BMXWAIST	6596.0	0.339415	0.148396	0.0	0.242160	0.320557	0.420732	1.0
LBXGH	6596.0	0.262830	0.102255	0.0	0.206612	0.231405	0.272727	1.0
LBXGLU	6596.0	0.175264	0.107518	0.0	0.123762	0.143564	0.178218	1.0
LBXTR	6596.0	0.133890	0.088713	0.0	0.072727	0.112987	0.171429	1.0
LBDLDLN	6596.0	0.280341	0.106965	0.0	0.200000	0.275362	0.350725	1.0
water_soluble_vitamins_sum	6596.0	0.583245	0.276465	0.0	0.416667	0.666667	0.833333	1.0
fat_soluble_vitamins_sum	6596.0	0.191480	0.219546	0.0	0.000000	0.166667	0.333333	1.0
major_minerals_sum	6596.0	0.425921	0.274985	0.0	0.250000	0.375000	0.625000	1.0
trace_minerals_sum	6596.0	0.619618	0.224037	0.0	0.600000	0.600000	0.800000	1.0



Methods

NON-TREE BASED ALGORITHMS:

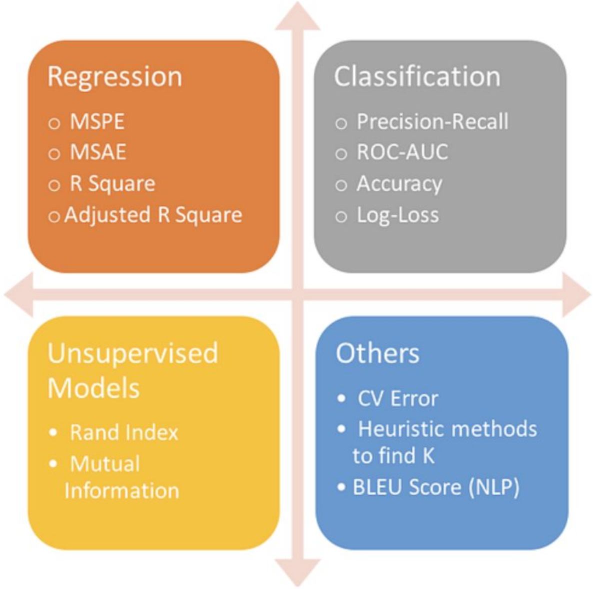
Logistic Regression
Naive Bayes
SVM(Support Vector Machines)

TREE BASED ALGORITHMS:

Decision tree Classifier
Random Forest Classifier
XGBoost

Results & Visualization

- **Accuracy**
- **Precision** (P)
- **Recall** (R)
- **F1 score** (F1)
- **Area under the ROC** *(Receiver Operating Characteristic)* **curve (AUC)**
    - Widely used metric for skewed binary classification tasks in the industry



Results & Visualization

Random Forest Classifier

```
Forest_reg = RandomForestClassifier(n_estimators=500, random_state=123123)
Forest_reg.fit(X_train, y_train)
y_pred = Forest_reg.predict(X_test)
Forest_reg.score(X_test, y_test)
```

```
0.9994946942900454
```

```
metrics.precision_score(y_test, y_pred, average='weighted', labels=np.unique(y_pred))
```

```
0.999495188235021
```

```
metrics.f1_score(y_test, y_pred, average='weighted', labels=np.unique(y_pred))
```

```
0.9994946857651343
```

Results & Visualization

F1 value = 0? Why?

```
In [139]: print(classification_report(y_test, y_pred))

              precision    recall  f1-score   support

           0       0.95      1.00      0.97      1651
           1       0.00      0.00      0.00        86

    accuracy                           0.95      1737
   macro avg       0.48      0.50      0.49      1737
weighted avg       0.90      0.95      0.93      1737

/Users/tom/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classification.py:13
Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted sam
arameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/tom/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classification.py:13
Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted sam
arameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/tom/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classification.py:13
Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted sam
arameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

In [147]: metrics.precision_score(y_test, y_pred, average='weighted', labels=np.unique(y_pred))

Out[147]: 0.9504893494530801

In [148]: metrics.f1_score(y_test, y_pred, average='weighted', labels=np.unique(y_pred))

Out[148]: 0.974616292798111
```

Sample # disparities between each label

```
True label counts:
1     1022
0      957
Name: MCQ160L, dtype: int64

Predicted label counts:
0     1496
1      483
dtype: int64

Predicted label counts:
1     998
0     981
dtype: int64
```

[From CCCA Proj.]Oversampling

SMOTE(Synthetic Minority Oversampling Technique)

: 불균형 데이터 세트를 해결하는 방법으로 낮은 비율 클래스 데이터들의 최근접을 이용하여 새로운 데이터 생성

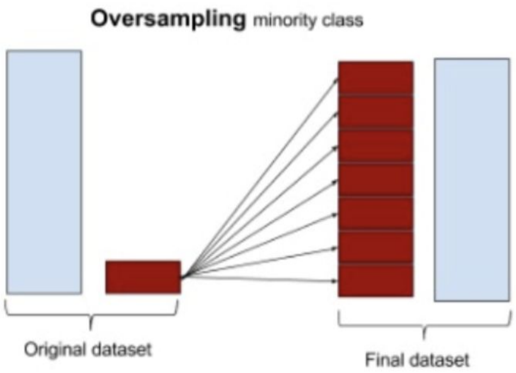
Oversampling

목적: 이상 데이터와 같이 적은 데이터를 증식하여 학습을 위한 충분한 데이터 확보하는 방법으로, 원본 데이터의 피처값들을 약간 변형하여 증식

1. 무작위추출: 무작위로 소수 데이터 복제

2. 유의정보: 사전에 기준을 정해서 minority data 복제
   - 정보가 손실되지 않는 장점이 있으나, 복제된 관측치를 원래 데이터 세트에 추가하기만 하면 여러 유형의 관측치를 다수 추가하여 overfitting을 초래함

3. 합성데이터 생성: 소수데이터를 단순 복제하는 것이 아니라 새로운 복제본을 만들어 냄



Results & Visualization

Support Vector Classification

```
model1=svm.SVC()

# Fitting the model
model1.fit (X_train, y_train)

# Predicting the test variables
y_pred = model1.predict(X_test)

# Getting the score
model1.score (X_test, y_test)
```

0.7170288024254674

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.72	0.67	0.70	957
1	0.71	0.76	0.73	1022
accuracy			0.72	1979
macro avg	0.72	0.72	0.72	1979
weighted avg	0.72	0.72	0.72	1979

Logistic Regression

```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
logreg.score(X_test, y_test)
```

0.6205154118241536

```
metrics.precision_score(y_test, y_pred, average='weighted', labels=np.unique(y_pred))
```

0.6216340000579095

```
metrics.f1_score(y_test, y_pred, average='weighted', labels=np.unique(y_pred))
```

0.6205595967869924

Naïve Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB
```

```
NB_classifier = GaussianNB()
NB_classifier.fit(X_train, y_train)
y_pred = NB_classifier.predict(X_test)
NB_classifier.score(X_test, y_test)
```

0.5538150581101566

```
metrics.precision_score(y_test, y_pred, average='weighted', l
```

0.586269372744594

```
metrics.f1_score(y_test, y_pred, average='weighted', labels=n
```

0.5229371430576902

Conclusion: 반성문

- ❑ Data Data **DATA**
- ❑ Spend more time on researching Precedent Study
- ❑ Secure “ready-to-run” model

Conclusion: To-Do

- ❑ Append additional dataset to enhance the model
- ❑ Careful statistical analysis on each variable
- ❑ Get it run & Post the update on FB

Thank you

Thank you

And I thank you for sharing your invigorating passion and your dearest integrity with us for the past semester. It has been a true pleasure ;D

Methods

- ❖ Test & Train data split
- ❖ Support Vector Classification

Test & Train data split

```
X_train, X_test, y_train,y_test = train_test_split(X_scaled,y,test_size=0.3,random_state=40)
```

Support Vector Classification

```
model1=svm.SVC()

# Fitting the model
model1.fit (X_train, y_train)

# Predicting the test variables
y_pred = model1.predict(X_test)

# Getting the score
model1.score (X_test, y_test)
```

```
0.7170288024254674
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.72      0.67      0.70       957
           1       0.71      0.76      0.73      1022

    accuracy                           0.72      1979
   macro avg       0.72      0.72      0.72      1979
weighted avg       0.72      0.72      0.72      1979
```

```
metrics.precision_score(y_test, y_pred, average='weighted', labels=np.unique(y_pred))
```

```
0.7173385265987279
```

```
metrics.f1_score(y_test, y_pred, average='weighted', labels=np.unique(y_pred))
```

```
0.7164297434462244
```

Methods

❖ Naive Bayes Classifier

Naïve Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB
```

```
NB_classifier = GaussianNB()
NB_classifier.fit(X_train, y_train)
y_pred = NB_classifier.predict(X_test)
NB_classifier.score(X_test, y_test)
```

```
0.5538150581101566
```

```
metrics.precision_score(y_test, y_pred, average='weighted', labels=np.unique(y_pred))
```

```
0.586269372744594
```

```
metrics.f1_score(y_test, y_pred, average='weighted', labels=np.unique(y_pred))
```

```
0.5229371430576902
```

Methods

- ❖ Logistic Regression

Logistic Regression

```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
logreg.score(X_test, y_test)
```

```
0.6205154118241536
```

```
metrics.precision_score(y_test, y_pred, average='weighted', labels=np.unique(y_pred))
```

```
0.6216340000579095
```

```
metrics.f1_score(y_test, y_pred, average='weighted', labels=np.unique(y_pred))
```

```
0.6205595967869924
```

```
print('True label counts:')
print(y_test.value_counts())
print('₩nPredicted label counts:')
print(pd.Series(y_pred).value_counts())
```

```
True label counts:
1    1022
0     957
Name: MCQ160L, dtype: int64

Predicted label counts:
0    1014
1     965
dtype: int64
```

Results_Liver

	SVM	Random Forest	Naive Bayes	Log Regression
TN				
TP				
FN				
FP				
Recall				
Specificity				
Precision				
Accuracy				
F1 score				

Results_Cardiovascular Disease

	SVM	Random Forest	Naive Bayes	Log Regression
TN				
TP				
FN				
FP				
Recall				
Specificity				
Precision				
Accuracy				
F1 score				

Results_Diabetes

	SVM	Random Forest	Naive Bayes	Log Regression
TN				
TP				
FN				
FP				
Recall				
Specificity				
Precision				
Accuracy				
F1 score				