**Introduction to Focus Area Project-1**

**Gitlab:** [**https://git.imp.fu-berlin.de/kunaphak91/ifabi-2019**](https://git.imp.fu-berlin.de/kunaphak91/ifabi-2019)

**Name: 1. Minie Jung**

**2. Kunaphas Kongkitimanon**

**3. Chang Wan-Ju**

# The goal of the project

Based on the available heart disease dataset from UCI machine learning[2], we develop models by training five different classifiers to predict the presence of heart disease in the patient. We also compare and analyze the results of heart disease prediction. Moreover, we conduct further analysis to see the advantages and disadvantages of using various data processing methods and more datasets. In addition, cooperating with preprocessing can aid to improve the procession of classifiers.

# 

# Scientific background

## Machine learning methods

### Multinomial Logistic regression

Multinomial logistic regression is a classification method that generalizes logistic regression for solving multiclass problems (more than two levels)[3]. A linear combination is used along with observed features and model parameters to determine the probability of each distinct value of the dependent variable.

### SVM (Support vector machines)

SVM constructs vectors/hyperplanes to separate data. To improve the accuracy of prediction, this method tries to find a vector/plane that has the maximum distance between the data of classes. A new example is predicted based on the space where it is mapped.

### 

### KNN(K-Nearest Neighbor)

KNN algorithm assumes that similar things are near to each other. The training set is vectors in multidimensional space. KNN algorithm selects k-closest vectors based on calculating distance between two vectors and predicts label to the most frequency label of k-closest training vectors.

### Decision Tree

A decision tree is a tree structure, which branch represents decision rule, an internal node represents features, and a leaf node represents the result. It classifies the data by using the Attribute Selection Measures. After selection measures, each feature gets a score. Each time the data partitioned, the decision tree selects the feature with the best score as a splitting feature.

### Random Forest Algorithms

Based on decision tree algorithms, a random forest algorithm builds a multiple depth decision trees to train data and gets averaging to minimize the loss function. However, it splits features depend on random cut rather than choosing the best cut point.

## 

## Description of the Data

This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them.

Attribute Information:

1. age

2. sex

3. chest pain type (4 values)

4. resting blood pressure

5. serum cholesterol in mg/dl

6. fasting blood sugar > 120 mg/dl

7. resting electrocardiographic results (values 0,1,2)

8. maximum heart rate achieved

9. exercise-induced angina

10. oldpeak = ST depression induced by exercise relative to rest

11. the slope of the peak exercise ST segment

12. number of major vessels (0-3) colored by fluoroscopy

13. thal: 3 = normal; 6 = fixed defect; 7 = reversible defect

14. target: integer valued from 0 (no presence) to 4

## Evaluation methods

### K-Folds Cross Validation

K-Fold is one of the techniques used to test the effectiveness of machine learning models. The method makes a fixed number of folds (or partitions) of the data, run the analysis on each fold, and then average the overall error estimate.

### 

### Accuracy

Accuracy is the ratio of the number of correct predictions to the total number of input samples.

### 

### F1

F1 is simply the harmonic mean (weighted average) of precision and recall, where an F1 score reaches its best value at 1 and worst score at 0.

### 

### Precision

Precision addresses how precise and accurate your model is out of those predicted positive value; in other words, how many of them are positive. Precision is a suitable measurement method to determine whether the costs of false positive is high or not.

### 

### Recall

Recall calculates how a proportion of actual positives were correctly identified against the results classified by an algorithm. Precision is a suitable measurement method to determine false negatives.

# Methodology

1. We use two datasets to feed and compare our model.
   1. processed.cleveland.data
   2. processed.merged.data; This data merged from 4 different data sources, which are processed.cleveland.data, processed.hungarian.data, processed.switzerland.data, and processed.va.data.
2. Data preprocessing
   1. Taking care of missing data
   2. Encode enum-like features
   3. Detect and remove outliers
   4. Normalize features
   5. Split data into feature and label set
3. Construct five machine learning models
4. Evaluate and Compare the result
   1. List each prediction scores
   2. Compare results between processed.merged.data and processed.cleveland.data.

# 

# Implementation

We use Scikit learn[1] library to help our implementation.

## Data preprocessing

Load data into a data frame and assign the column name into its.

col\_names=["age","sex", "cp", "trestbps", "chol", "fbs", "restecg", "thalach", "exang", "oldpeak", "slope", "ca", "thal", "target"]

df = pd.read\_csv("processed.cleveland.data",header=None, names=col\_names, na\_values="?")

df.info()

Identify missing values in a dataset

total = df.isnull().sum().sort\_values(ascending=False)

percent = ((df.isnull().sum()/df.isnull().count())\*100).sort\_values(ascending=False)

missing\_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])

f, ax = plt.subplots(figsize=(15, 6))

plt.xticks(rotation='90')

sns.barplot(x=missing\_data.index, y=missing\_data['Percent'])

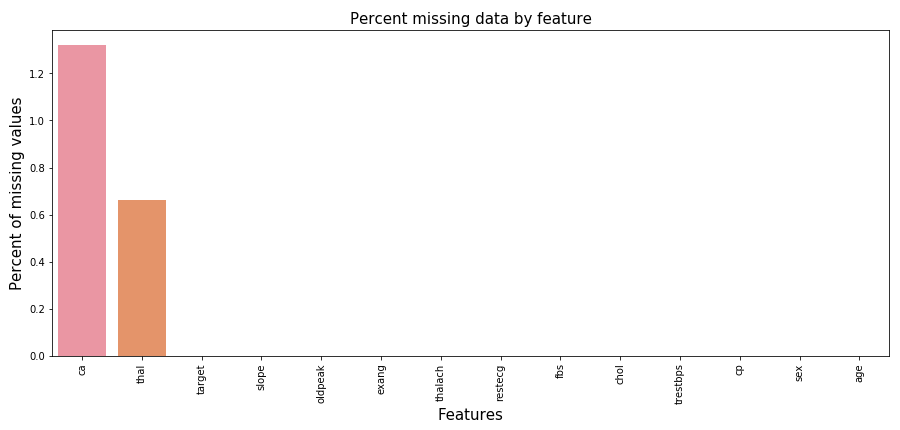
plt.xlabel('Features', fontsize=15)

plt.ylabel('Percent of missing values', fontsize=15)

plt.title('Percent missing data by feature', fontsize=15)

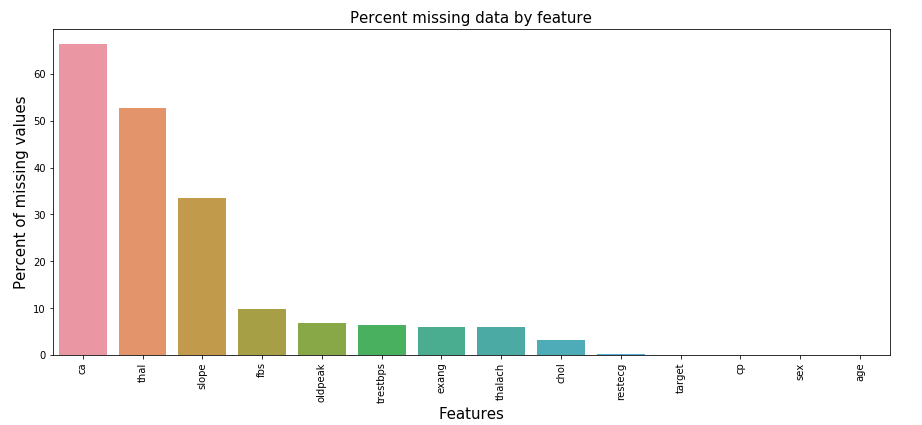
missing\_data

For **processed.cleveland.data**



The cleveland dataset contains a few missing value in “ca” and “thal” columns.

For **processed.merged.data**

****

However, when we did merging data sources, the volume of missing value is significantly raised.

We apply a Univariate feature imputation[4] to handle missing data

from sklearn.experimental import enable\_iterative\_imputer

from sklearn.impute import IterativeImputer , SimpleImputer

# take care oldpeak

df\_oldpeak =pd.DataFrame(data=df["oldpeak"].to\_numpy(),columns=["oldpeak"])

imp = SimpleImputer(strategy ="mean")

imp.fit(df\_oldpeak.to\_numpy())

\_oldpeak = np.around(imp.transform(df\_oldpeak.to\_numpy() ), decimals=1) # round to the given number of decimals.

#\_oldpeak

# take care thal

df\_thal =pd.DataFrame(data=df["thal"].to\_numpy(),columns=["thal"])

imp = SimpleImputer(strategy ="median")

imp.fit(df\_thal.to\_numpy())

\_thal = np.round(imp.transform(df\_thal.to\_numpy() ))

# All remaing columns

imp = SimpleImputer(strategy ="mean")

imp.fit(df.to\_numpy())

np\_imputed = np.round(imp.transform(df.to\_numpy()))

# Comvert NP array to data frame

df = pd.DataFrame(data=np\_imputed[:,:],columns=col\_names)

df["oldpeak"] = \_oldpeak

df["thal"] = \_thal

#df\_test.to\_csv('df\_new.csv')

for col in df:

print(col , df[col].unique())

The machine might be considered the enum-like data as continuous data. So we convert the data like [0, 1] → [[1, 0], [0, 1]]

def onehot(ser, num\_classes=None):

if num\_classes == None:

num\_classes = len(np.unique(ser))

return np.identity(num\_classes)[ser]

new\_col\_names = []

need\_encode\_col = ["restecg", "thal", "slope", "cp"]

no\_encode\_col = [col for col in df.columns if col not in need\_encode\_col]

df\_new = df[need\_encode\_col]

new\_df = df[no\_encode\_col]

for col in need\_encode\_col:

#print(df[col])

if col == "cp":

df\_new[col] = df\_new[col]-1

elif col == "slope":

df\_new[col] = df\_new[col]-1

num\_classes = len(df[col].unique())

new\_col\_names = [f"{col}\_{i}" for i in range(num\_classes)]

encoded = pd.DataFrame(onehot(df\_new[col], num\_classes), columns=new\_col\_names, dtype=int)

new\_df = pd.concat([new\_df, encoded], axis=1)

new\_df.head()

We use the Elliptic Envelope method[5] to detect outliers. This model fits multivariate gaussian distribution to the dataset. It makes ellipsoid shape of data and define outliers which are far from shape.

#detect outliers\_Elliptic envelope

from sklearn.covariance import EllipticEnvelope

clf = EllipticEnvelope(contamination=.1,random\_state=0)

clf.fit(new\_df)

ee\_scores = pd.Series(clf.decision\_function(new\_df))

ee\_predict = clf.predict(new\_df)

ee\_predict #value -1 means it is outlier

Some features have values on large scale and some are not. After normalization, we can make the feature value to be centered to 0.

import copy

norm\_col = ["age", "trestbps", "chol", "thalach", "oldpeak"]

norm\_df = copy.copy(ee\_df)

for col in norm\_col:

norm\_df[col] = (norm\_df[col] - norm\_df[col].mean()) / (norm\_df[col].max() - norm\_df[col].min())

norm\_df

Finally we split data into two parts to prepare an input for building machine learning model and evaluation.

#For model without normalization

X = ee\_df.loc[:, new\_df.columns != 'target'] # Features

y = ee\_df[ ["target"]] # Label

print(X)

print(y)

# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state = 0)

#For model with normalization

X = norm\_df.loc[:, new\_df.columns != 'target'] # Features

y = norm\_df[ ["target"]] # Label

print(X)

print(y)

# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state = 0)

## Classifiers

### Multinomial Logistic regression

from sklearn.linear\_model import LogisticRegression

classifier\_logis = LogisticRegression(random\_state = 0,multi\_class='multinomial', solver='newton-cg')

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None, max\_iter=100,

multi\_class='multinomial', n\_jobs=None, penalty='l2',

random\_state=0, solver='newton-cg', tol=0.0001, verbose=0,

warm\_start=False)

### SVM (Support vector machines)

from sklearn.svm import SVC

classifier\_svm = SVC(kernel = "rbf", random\_state = 0 )

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='auto\_deprecated',

kernel='rbf', max\_iter=-1, probability=False, random\_state=0,

shrinking=True, tol=0.001, verbose=False)

### 

### K-Nearest Neighbor

from sklearn.neighbors import KNeighborsClassifier

clf\_neigh = KNeighborsClassifier(n\_neighbors=3)

KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',

metric\_params=None, n\_jobs=None, n\_neighbors=3, p=2,

weights='uniform')

### Decision Tree

from sklearn import tree

clf\_decision\_tree = tree.DecisionTreeClassifier(criterion = 'entropy', max\_depth = 4, random\_state = 0)

DecisionTreeClassifier(class\_weight=None, criterion='entropy', max\_depth=4,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, presort=False,

random\_state=0, splitter='best')

### 

### Random Forest Algorithms

from sklearn.ensemble import RandomForestClassifier

clf\_random\_forest=RandomForestClassifier(random\_state = 0)

RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

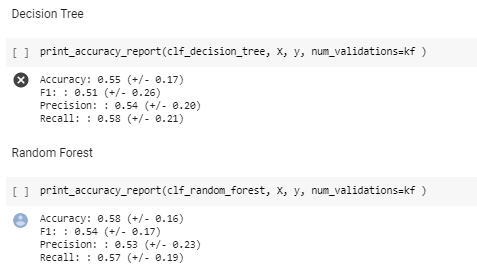
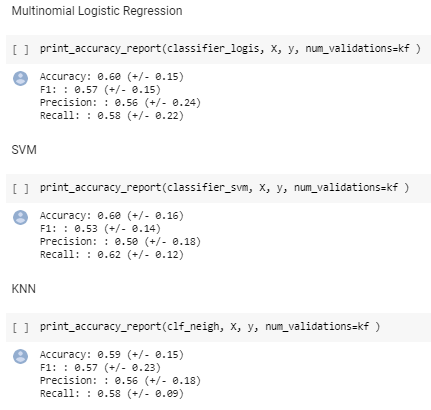
min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators='warn',

n\_jobs=None, oob\_score=False, random\_state=0, verbose=0,

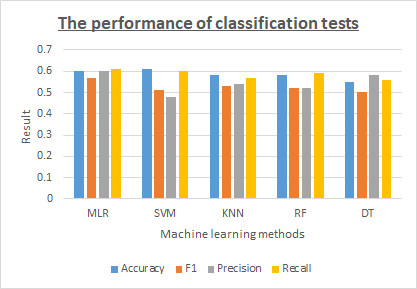
warm\_start=False)

Some screenshots from final result when using processed cleveland data.



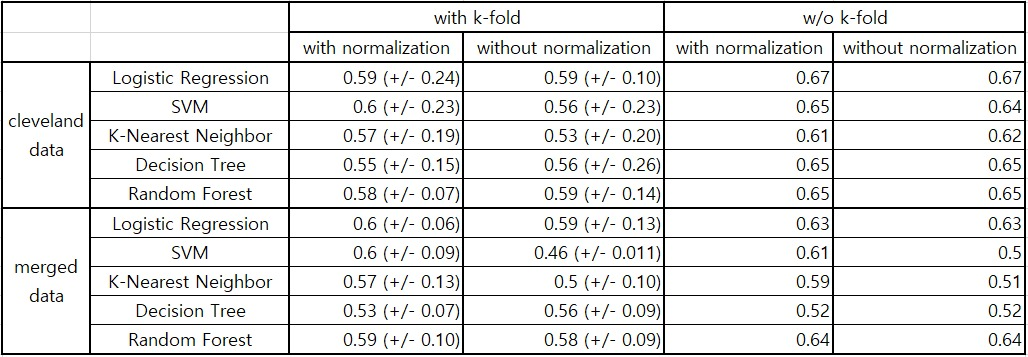
## 

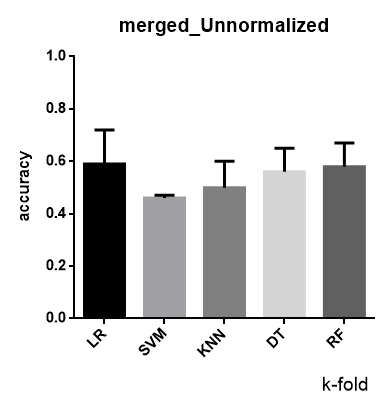
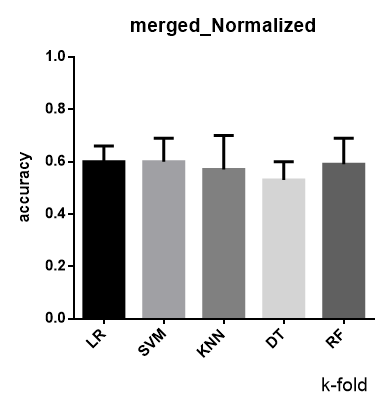
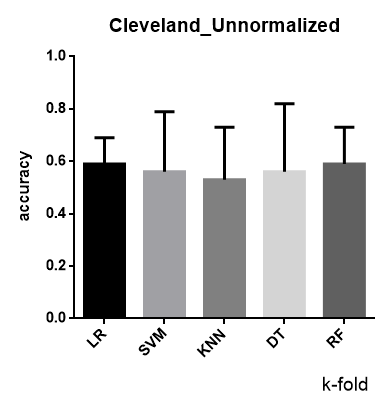
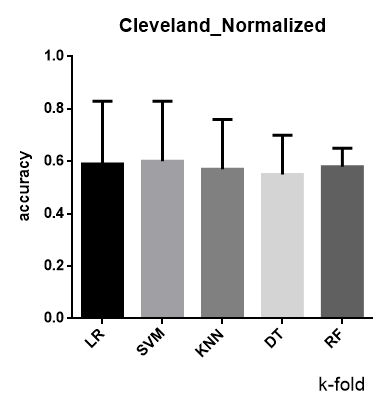
## Evaluation Results

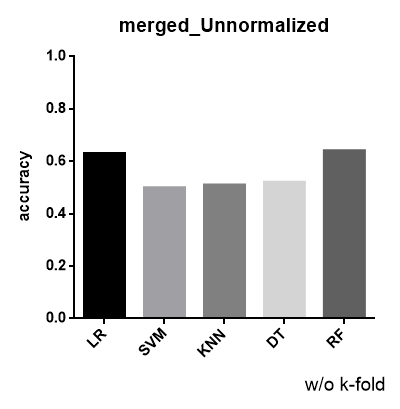
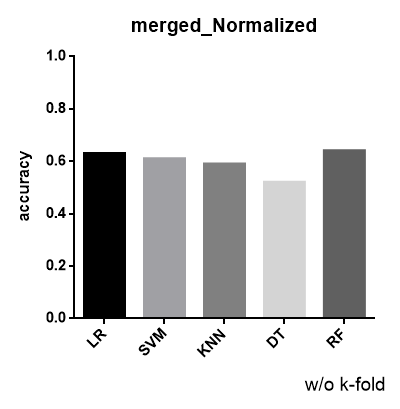
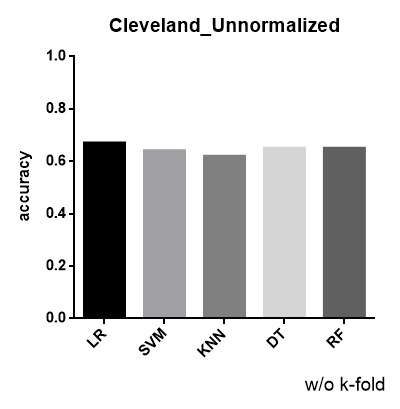
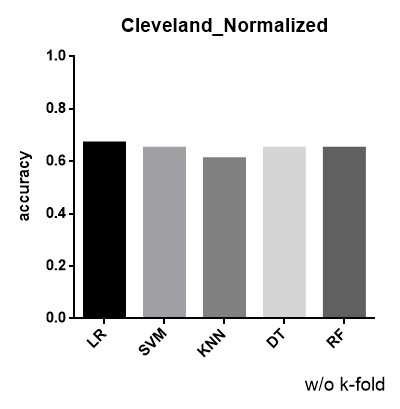


It is better to look at both the precision and recall scores, so we consider the F1 score as a weighted average of precision and recall. Overall, The best performance among tested algorithms is in MLR (Multiple Linear Regression).

Through code testing, We found several parameters that might affect accuracy. Therefore, we want to observe the difference by testing those parameters separately (normalized data, unnormalized data, with k-fold, without k-fold).







In most classifiers, it does not have significant difference except K-NN. We observed that KNN's performance is improved after normalized with k-fold. However, the group of cleveland without k-fold seems not different after normalized.

# Discussion

**Why is this a typical project for a data-scientist?**

In our opinion, data scientist is a group of people who can develop available tools by different methods(mathematics, computers, statistics) to obtain meaningful information from various data.

Through this project, we learn how to utilize machine learning to develop a tool and apply it in the medical field. Besides, learning how algorithms operate in the progress of disease prediction. We also observe which parameters may cause influences in the models. Moreover, we can enhance our models by using other techniques to assist us in achieving better model performance, such as hyper-parameter tuning, which is attempting to adjust kernel or parameters automatically to reach a better score or avoiding dummy variable trap. We may also select other machine learning methods, for example, ordinal logistic regression, to find an algorithm that matches our datasets.

Since it corresponds with the goal in data science, we consider that it belonged to one of the typical projects for data scientists.

# References

1. [scikitlearn] <https://scikit-learn.org/stable/>
2. [UCI dataset]<https://archive.ics.uci.edu/ml/datasets/Heart+Disease>
3. [Multinomial]<https://en.wikipedia.org/wiki/Multinomial_logistic_regression>
4. [Impute]<https://scikit-learn.org/stable/modules/impute.html>
5. [Elliptic Envelope]<https://scikit-learn.org/stable/modules/outlier_detection.html>

# 