Python 실습

앙상블 실습

- 분류 문제

  1. 5-Cross validation을 이용한 파라미터 튜닝 및 테스트데이터 예측
  2. dataset : Personal loan

- 회귀 문제

  1. 3-Cross validation을 이용한 파라미터 튜닝 및 테스트데이터 예측
  2. dataset : mortgage

앙상블 실습

- 분류 문제

1. 5-Cross validation을 이용한 파라미터 튜닝 및 테스트데이터 예측
2. dataset : Personal loan
3. 사용 알고리즘
    - 3-(1) ANN
    - 3-(2) RandomForest
    - 3-(3) Adaboost
    - 3-(4) Bagging ANN
    - 3-(5) Bagging Tree
    - 3-(6) Gradient boosting machine

- Code

1. Classification_CV.ipynb
2. Classification_Test.ipynb

분류 문제 5-fold CV

- 사용할 모듈을 불러옴
- 5-Fold CV를 위해서 FOLD_VALUE에 5, RANDOM_STATE는 실습에서의 동일을 결과를 위해 사용

Ensemble Code 1 - Cross Validation

필요한 모듈을 불러온다

[1]:
```
import numpy as np
import copy
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import collections
print("Module Ready!")
```

```
Module Ready!
```

1. Fold_Value는 CV Fold갯수를 의미함

2. RANDOM_STATE는 실습을 위해 모델의 결과를 같게 하기 위함임

[2]:
```
############################
FOLD_VALUE = 5
RANDOM_STATE = 1026
############################
```

분류 문제 5-fold CV

- **Personal Loan에서 사용할 입력 변수 중 ID와 ZIPCode는 제외**

ID	Customer ID
Age	Customer's Age in completed years
Experience	#years of professional experience
Income	Annual income of the customer ($000)
ZIPCode	Home Address ZIP code.
Family	Family size (dependents) of the customer
CCAvg	Avg. Spending on Credit Cards per month ($000)
Education	Education Level. 1: Undergrad; 2: Graduate; 3: Advanced/Professional
Mortgage	Value of house mortgage if any. ($000)
Personal Loan	Did this customer accept the personal loan offered in the last campaign?
Securities Account	Does the customer have a Securities account with the bank?
CD Account	Does the customer have a Certificate of Deposit (CD) account with the bank?
Online	Does the customer use internet banking facilities?
CreditCard	Does the customer use a credit card issued by UniversalBank?

분류 문제 5-fold CV

- 난수를 고정하여 8:2 = Training data : Test data로 나눔

사용할 Personal Loan 데이터셋을 불러옴

> 1. 난수를 고정하여 8:2 = Training data : Test data로 나눔
> 2. Training dataset의 column별 std와 mean을 이용하여 Train/Test dataset standardization 수행

In [3]:

```
# 사용할 Personal Loan 데이터셋을 불러옵니다.
Rawdata = pd.read_csv('dataset/Personal Loan.csv')
# Print Column names
print("'Personal Loan' data column name : ", list(Rawdata.columns.values))
print("ID와 ZIP Code는 사용하지 않습니다. 또한 Personal Loan을 분류하는 binary classification 문제 입니다.")
# Allocate column index based on Input and Output varaibles
Input_Column_Index = np.concatenate((range(1,4),range(5,9),range(10,14)))
Target_Column_Index = np.array([9])


# 같은 데이터셋을 사용하기 위해서 난수를 고정합니다.
np.random.seed(150)
Train_Index = np.random.choice(np.shape(Rawdata)[0],int(np.shape(Rawdata)[0]*0.8),replace=False)

# Input variable과 Output variable을 Numpy array로 변환합니다.
Rawdata_Input = np.array(Rawdata)[:,Input_Column_Index]
Rawdata_Output = np.array(Rawdata)[:,Target_Column_Index]


# Training data와 Test data를 나누어 줍니다.
Train_Input = Rawdata_Input[Train_Index,:]
Train_Output = Rawdata_Output[Train_Index,:]
Test_Input = Rawdata_Input[np.delete(range(np.shape(Rawdata)[0]),Train_Index),:]
Test_Output = Rawdata_Output[np.delete(range(np.shape(Rawdata)[0]),Train_Index),:]
print('Data partition complete! ₩nTrain_Input shape :',np.shape(Train_Input),'₩nTrain_Output shape :',np.shape(Train_Output))
print('Test_Input shape :',np.shape(Test_Input),'₩nTest_Output shape :',np.shape(Test_Output))
```

분류 문제 5-fold CV

- **Training dataset의 std와 mean을 이용하여 Train/Test dataset standardization 수행**

```
# Input variable standardization based on Training data

def standardization(Data,Data2):
    return ((Data - np.mean(Data2, axis=0)) / np.std(Data2, axis=0))

Train_Input_Normalized = copy.deepcopy(standardization(Train_Input,Train_Input))
Test_Input_Normalized = copy.deepcopy(standardization(Test_Input,Train_Input))
print("standardization complete!")
```

```
'Personal Loan' data column name :  ['ID', 'Age', 'Experience', 'Income', 'ZIP Code', 'Family', 'CCAvg', 'Education', 'Mortgage', 'Personal
 Loan', 'Securities Account', 'CD Account', 'Online', 'CreditCard']
ID와 ZIP Code는 사용하지 않습니다. 또한 Personal Loan을 분류하는 binary classification 문제 입니다.
Data partition complete!
Train_Input shape : (2000, 11)
Train_Output shape : (2000, 1)
Test_Input shape : (500, 11)
Test_Output shape : (500, 1)
standardization complete!
```

분류 문제 5-fold CV

- **N-Fold dataset을 위하여 함수를 생성함**

Best hyperparameter를 찾기위하여 5-Fold 로 데이터를 나누며 CV할 함수를 생성

In [4]:
```
# Best Hyperparameter를 찾기위하여 5-Fold Cross Validation을 한다
def k_Fold_Maker(InputData,OutputData,Partition_Number):
    Index = 0
    Input_List = list()
    Output_List = list()
    Length = int(np.floor(np.shape(InputData)[0]/Partition_Number))
    for i in range(Partition_Number):
        if(i == (Partition_Number-1)):
            Input_List.append(InputData[range(Index+(Length* i), np.shape(InputData)[0]), :])
            Output_List.append(OutputData[range(Index+(Length* i), np.shape(InputData)[0]), :])
        else:
            Input_List.append(InputData[range(Index + (Length * i), Index + (Length * (i + 1))), :])
            Output_List.append(OutputData[range(Index + (Length * i), Index + (Length * (i + 1))), :])
    return(Input_List,Output_List)

# Make 5-Fold dataset for Cross validation
Fold_Input, Fold_Output = k_Fold_Maker(Train_Input_Normalized, Train_Output, FOLD_VALUE)
```

분류 문제 5-fold CV

- **Cross validation의 결과를 생성할 함수와 분류 평가지표를 위한 함수를 생성**

```
# Cross validation의 결과를 위한 함수
def CV_Result_Each_Model(Hyper_Para,Model):
    FULL_Results = list() # 모든 결과를 담을 객체
    for i in range(FOLD_VALUE):
        Tr_Index = np.delete(range(FOLD_VALUE),i) #Training 에 사용할 Fold Index
        Val_Index = i                    #Validation에 사용할 Fold Index

        TRAIN_INPUT  = list()
        TRAIN_OUTPUT = list()
        for j in Tr_Index:
            TRAIN_INPUT.append(Fold_Input[j])
            TRAIN_OUTPUT.append(Fold_Output[j])
        TRAIN_INPUT = np.concatenate(TRAIN_INPUT)
        TRAIN_OUTPUT = np.concatenate(TRAIN_OUTPUT)
        VALID_INPUT = Fold_Input[i]
        VALID_OUTPUT = Fold_Output[i]
        FULL_Results.append(Model(Hyper_Para,TRAIN_INPUT,TRAIN_OUTPUT,VALID_INPUT,VALID_OUTPUT))
    print("CV Complete!")
    FULL_Results=np.concatenate(FULL_Results,axis=0)
    return(FULL_Results)

# 평가지표를 위한 함수를 생성한다.
def Valid_Index(Data,NAME):
    Accuracy = (Data[0,0]+Data[1,1])/np.sum(Data)
    TPR = Data[1,1]/np.sum(Data[1,:])
    TNR = Data[0,0]/np.sum(Data[0,:])
    Precision = Data[1,1]/np.sum(Data[:,1])
    BCR = np.sqrt(TPR*TNR)
    F1 = (2*TPR*Precision)/(TPR+Precision)
    TMP=pd.DataFrame({'Model' : NAME,
                      'Accuracy' : [Accuracy],
                      'TPR': [TPR],
                      'TNR': [TNR],
                      'Precision': [Precision],
                      'BCR': [BCR],
                      'F1': [F1]})
    Results=TMP[['Model','Accuracy','F1','BCR','Precision','TPR','TNR']]
    return(Results)
```

분류 문제 5-fold CV – (1) ANN

- **5-fold cross validation Neural Network**

```
##############################################
# Neural Network Hyperparameter Set
##############################################
ACTIVATION = 'tanh'
SOLVER = 'adam'
BATCH_SIZE = 32
HIDDEN_LAYER= [10,10,10]
TR_INPUT = Train_Input
TR_OUTPUT = Train_Output[:,0]
Iterlation = 3000
L2_Penalty = 0.001
Visualization = False
Validation_Percent = 0.0
Decay_Method = 'invscaling'
Power_Value = 0.5
Tolerence_Value = 1e-04
##############################################
```

sklearn.neural_network.MLPClassifier

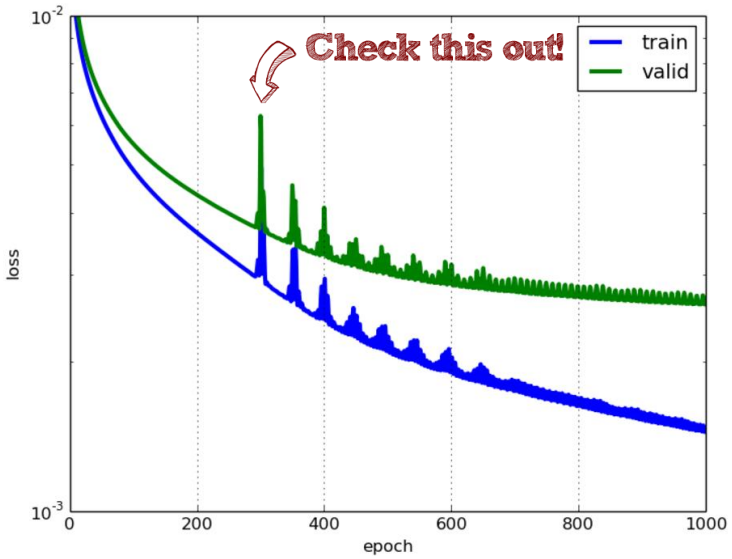
class sklearn.neural_network. MLPClassifier (*hidden_layer_sizes=(100,), activation='relu', solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08*) [source]

Neural network hyperparameter set	
ACTIVATION = 'tanh'	# 어떤 Activation Function을 사용할 것인가? ('identity', 'logisitic', 'tanh', 'relu' 제공)
SOLVER = 'adam'	# 어떤 Optmization Method를 사용할 것인가 ? ('lbfgs', 'sgd', 'adam') (Network가 크면 lbfgs는 피해주세요)
BATCH_SIZE = 32	# Batch-Size는 어떤것을 사용할 것인가? (데이터에 따라서 달리 사용하면 됨)
HIDDEN_LAYER= [10,10,10]	# Hidden Layer와 Node는 어떻게 구성할 것인가? (실험적으로 최적의 Network구조를 찾아야함)
TR_INPUT = Train_Input	# Input Data
TR_OUTPUT = Train_Output[:,0]	# Output Data
Iterlation = 3000	# Epoch 32*200/1600 == 4 (최대 몇 iterlation까지 갈것인가? 결국 사용하는 데이터의 정량적인 갯수는 BATCH_SIZE*Iterlation이 됨)
L2_Penalty = 0.001	#(Cross_Entropy에서의 L2 regularization Lambda를 얼마를 할당할 것인가?)
Visualization = False	# Iterlation 마다의 Loss를 console에서 확인 할 것인가?
Validation_Percent = 0.0 #Float type	# Validation셋을 얼마나 구성하여 확인 할 것인가?
Decay_Method = 'invscaling'	# Weight Decay Method를 어떤것을 사용할 것인가 (constant, invscaling adaptive 제공)
Power_Value = 0.5	# invscaling을 사용할때 exponent for inverse scaling learning rate를 얼만큼 줄것인가?
Tolerence_Value = 1e-04 (Tolerence_Value = 0.5?)	# 해당 sklearn에는 Tolerence_Value보다 2번의 iterlation에서 작게 나오면 early stop을 하게 설계되어있음 # 이는 실제 상황에서 너무 일찍 멈출 확률이 높음 심지어 -로 튀는 경우도 네트워크에서 많기 때문임 # 따라서 Tolerence_Value를 음의 값을 크게 주면 설정한 Iterlation을 모두 돌게 되는 상황을 만들 수 있음 # 이는 CV등으로 Overfitting되는 지점을 모니터링한후 최적의 Network를 만들 수 있음

분류 문제 5-fold CV – (1) ANN

- **Sklearn에서 제공하는 optimization method중 sgd나 adam을 사용할 경우 loss가 tol보다 연속적으로 튀는 경우가 발생할 수도 있음 이 경우는 tol값을 음수로 주고 자신이 지정한 epoch size만큼 학습 하는것이 하나의 방법**

Tolerence_Value = 1e-04 (Tolerence_Value = 0.5?)	# 해당 sklearn에는 Tolerence_Value보다 2번의 iterlation에서 작게 나오면 early stop을 하게 설계되어있음 # 이는 실제 상황에서 너무 일찍 멈출 확률이 높음 심지어 -로 튀는 경우도 네트워크에서 많기 때문임 # 따라서 Tolerence_Value를 음의 값을 크게 주면 설정한 Iterlation을 모두 돌게 되는 상황을 만들 수 있음 # 이는 CV등으로 Overfitting되는 지점을 모니터링한후 최적의 Network를 만들 수 있음



sklearn.neural_network.MLPClassifier

class sklearn.neural_network. MLPClassifier (*hidden_layer_sizes=(100,), activation='relu', solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08*) [source]

tol : float, optional, default 1e-4

Tolerance for the optimization. When the loss or score is not improving by at least tol for two consecutive iterations, unless learning_rate is set to 'adaptive', convergence is considered to be reached and training stops.

http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

분류 문제 5-fold CV – (1) ANN

- from sklearn.neural_network import MLPClassifier 의 모듈을 이용하여 함수를 구축
- 앞서 구축한 함수인 CV_Results_Each_Model을 사용하여 Hidden layer/node별 결과를 저장
- 앞서 구축한 함수인 Valid_Index을 사용하여 5-fold CV 분류 성능을 저장

```
def NeuralNetwork(HIDDEN_LAYER,Train_Input,TR_OUTPUT,Val_Input,Val_Output):
    MLP=MLPClassifier(activation=ACTIVATION,solver=SOLVER,alpha=L2_Penalty,
                      hidden_layer_sizes=HIDDEN_LAYER,
                      batch_size=BATCH_SIZE,max_iter=Iterlation,verbose=Visualization,early_stopping=False,power_t=Power_Value,
                      validation_fraction=Validation_Percent,learning_rate=Decay_Method,tol=Tolerence_Value,
                      random_state =RANDOM_STATE).fit(Train_Input,TR_OUTPUT[:,0])
    Predict_Value = MLP.predict(Val_Input)
    return(np.concatenate((Predict_Value[:,np.newaxis],Val_Output),axis=1))


# 자 이제 NeuralNet을 기준으로 파라미터별 5-Fold CV를 해보도록 한다.
One_Layer_Node_10 = CV_Result_Each_Model([10],NeuralNetwork)
One_Layer_Node_20 = CV_Result_Each_Model([20],NeuralNetwork)
Two_Layer_Node_10 = CV_Result_Each_Model([10,10],NeuralNetwork)
Two_Layer_Node_20 = CV_Result_Each_Model([20,20],NeuralNetwork)
Three_Layer_Node_10 = CV_Result_Each_Model([10,10,10],NeuralNetwork)
Three_Layer_Node_20 = CV_Result_Each_Model([20,20,20],NeuralNetwork)

ANN_5CV_Results=Valid_Index(confusion_matrix(One_Layer_Node_10[:,1],One_Layer_Node_10[:,0]),"One_Layer_Node_10").append([
Valid_Index(confusion_matrix(One_Layer_Node_20[:,1],One_Layer_Node_20[:,0]),"One_Layer_Node_20"),
Valid_Index(confusion_matrix(Two_Layer_Node_10[:,1],Two_Layer_Node_10[:,0]),"Two_Layer_Node_10"),
Valid_Index(confusion_matrix(Two_Layer_Node_20[:,1],Two_Layer_Node_20[:,0]),"Two_Layer_Node_20"),
Valid_Index(confusion_matrix(Three_Layer_Node_10[:,1],Three_Layer_Node_10[:,0]),"Three_Layer_Node_10"),
Valid_Index(confusion_matrix(Three_Layer_Node_20[:,1],Three_Layer_Node_20[:,0]),"Three_Layer_Node_20")])

ANN_5CV_Results = ANN_5CV_Results.sort_values(by=['F1'],ascending=False)
pd.DataFrame(ANN_5CV_Results)
```

```
CV Complete!
CV Complete!
CV Complete!
CV Complete!
CV Complete!
CV Complete!
```

Out [5] :

	Model	Accuracy	F1	BCR	Precision	TPR	TNR
0	One_Layer_Node_20	0.9735	0.860892	0.901252	0.906077	0.820	0.990556
0	Two_Layer_Node_20	0.9705	0.849873	0.907160	0.865285	0.835	0.985556
0	One_Layer_Node_10	0.9715	0.848000	0.887656	0.908571	0.795	0.991111
0	Three_Layer_Node_10	0.9695	0.844784	0.904185	0.860104	0.830	0.985000
0	Three_Layer_Node_20	0.9690	0.837696	0.888944	0.879121	0.800	0.987778
0	Two_Layer_Node_10	0.9680	0.833333	0.888444	0.869565	0.800	0.986667

분류 문제 5-fold CV – (2) RandomForest

- **5-fold cross validation RandomForest**

3.2.4.3.1. sklearn.ensemble.RandomForestClassifier

« *class* sklearn.ensemble. **RandomForestClassifier** (*n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False, class_weight=None*) [source]

Model2. RandomForest

RandomForest 함수 ,5-Fold CV를 위해 함수 실행

In [6]:
```
def RandomForest(NUMBER,Train_Input,TR_OUTPUT,Val_Input,Val_Output):
    RF=RandomForestClassifier(n_estimators=NUMBER,
                              max_features="sqrt",
                              random_state=RANDOM_STATE).fit(Train_Input,TR_OUTPUT[:,0])
    Predict_Value = RF.predict(Val_Input)
    return(np.concatenate((Predict_Value[:,np.newaxis],Val_Output),axis=1))

RF50_CV = CV_Result_Each_Model(50,RandomForest)
RF100_CV = CV_Result_Each_Model(100,RandomForest)
RF150_CV = CV_Result_Each_Model(150,RandomForest)
RF200_CV = CV_Result_Each_Model(200,RandomForest)
```

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

분류 문제 5-fold CV – (2) RandomForest

- from sklearn.ensemble import RandomForestClassifier의 모듈을 이용하여 함수를 구축
- 앞서 구축한 함수인 CV_Results_Each_Model을 사용하여 tree 개수별 결과를 저장
- 앞서 구축한 함수인 Valid_Index을 사용하여 5-fold CV 분류 성능을 저장

```
def RandomForest(NUMBER,Train_Input,TR_OUTPUT,Val_Input,Val_Output):
    RF=RandomForestClassifier(n_estimators=NUMBER,
                              max_features="sqrt",
                              random_state=RANDOM_STATE).fit(Train_Input,TR_OUTPUT[:,0])
    Predict_Value = RF.predict(Val_Input)
    return(np.concatenate((Predict_Value[:,np.newaxis],Val_Output),axis=1))

RF50_CV = CV_Result_Each_Model(50,RandomForest)
RF100_CV = CV_Result_Each_Model(100,RandomForest)
RF150_CV = CV_Result_Each_Model(150,RandomForest)
RF200_CV = CV_Result_Each_Model(200,RandomForest)

ANN_5CV_Results=Valid_Index(confusion_matrix(RF50_CV[:,1],RF50_CV[:,0]),"RF50_CV").append([
Valid_Index(confusion_matrix(RF100_CV[:,1],RF100_CV[:,0]),"RF100_CV"),
Valid_Index(confusion_matrix(RF150_CV[:,1],RF150_CV[:,0]),"RF150_CV"),
Valid_Index(confusion_matrix(RF200_CV[:,1],RF200_CV[:,0]),"RF200_CV")])

ANN_5CV_Results = ANN_5CV_Results.sort_values(by=['F1'],ascending=False)
pd.DataFrame(ANN_5CV_Results)
```

```
CV Complete!
CV Complete!
CV Complete!
CV Complete!
```

	Model	Accuracy	F1	BCR	Precision	TPR	TNR
0	RF150_CV	0.9840	0.914439	0.923891	0.982759	0.855	0.998333
0	RF200_CV	0.9835	0.911528	0.921186	0.982659	0.850	0.998333
0	RF100_CV	0.9830	0.909091	0.920929	0.977011	0.850	0.997778
0	RF50_CV	0.9830	0.908602	0.918472	0.982558	0.845	0.998333

분류 문제 5-fold CV – (3) AdaBoost

- from sklearn.ensemble import AdaBoostClassifier 의 모듈을 이용하여 함수를 구축
- 앞서 구축한 함수인 CV_Results_Each_Model을 사용하여 tree 개수별 결과를 저장
- 앞서 구축한 함수인 Valid_Index을 사용하여 5-fold CV 분류 성능을 저장

sklearn.ensemble.AdaBoostClassifier

class sklearn.ensemble. AdaBoostClassifier (*base_estimator=None*, *n_estimators=50*, *learning_rate=1.0*, *algorithm='SAMME.R'*, *random_state=None*)

Model3. Adaboost

Adaboost함수와 ,5-Fold CV를 위해 함수를 생성 후 이행

In [14]:
```
def AdaBoost(NUMBER,Train_Input,TR_OUTPUT,Val_Input,Val_Output):
    ADA=AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),n_estimators=NUMBER,algorithm="SAMME",
                           random_state=RANDOM_STATE).fit(Train_Input,TR_OUTPUT[:,0])
    Predict_Value = ADA.predict(Val_Input)
    return(np.concatenate((Predict_Value[:,np.newaxis],Val_Output),axis=1))

Adaboost_50 = CV_Result_Each_Model(50,AdaBoost)
Adaboost_100 = CV_Result_Each_Model(100,AdaBoost)
Adaboost_200 = CV_Result_Each_Model(200,AdaBoost)
Adaboost_300 = CV_Result_Each_Model(300,AdaBoost)
Adaboost_1000 = CV_Result_Each_Model(1000,AdaBoost)

ADA_5CV_Results=Valid_Index(confusion_matrix(Adaboost_50[:,1],Adaboost_50[:,0]),"Ada_50").append([
Valid_Index(confusion_matrix(Adaboost_100[:,1],Adaboost_100[:,0]),"Ada_100"),
Valid_Index(confusion_matrix(Adaboost_200[:,1],Adaboost_200[:,0]),"Ada_200"),
Valid_Index(confusion_matrix(Adaboost_300[:,1],Adaboost_300[:,0]),"Ada_300"),
Valid_Index(confusion_matrix(Adaboost_1000[:,1],Adaboost_1000[:,0]),"Ada_1000")])
```

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

http://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_twoclass.html

분류 문제 5-fold CV – (3) AdaBoost

- from sklearn.ensemble import AdaBoostClassifier 의 모듈을 이용하여 함수를 구축
- 앞서 구축한 함수인 CV_Results_Each_Model을 사용하여 tree 개수별 결과를 저장
- 앞서 구축한 함수인 Valid_Index을 사용하여 5-fold CV 분류 성능을 저장

```
def AdaBoost(NUMBER,Train_Input,TR_OUTPUT,Val_Input,Val_Output):
    ADA=AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),n_estimators=NUMBER,algorithm="SAMME",
                           random_state=RANDOM_STATE).fit(Train_Input,TR_OUTPUT[:,0])
    Predict_Value = ADA.predict(Val_Input)
    return(np.concatenate((Predict_Value[:,np.newaxis],Val_Output),axis=1))

Adaboost_50 = CV_Result_Each_Model(50,AdaBoost)
Adaboost_100 = CV_Result_Each_Model(100,AdaBoost)
Adaboost_200 = CV_Result_Each_Model(200,AdaBoost)
Adaboost_300 = CV_Result_Each_Model(300,AdaBoost)
Adaboost_1000 = CV_Result_Each_Model(1000,AdaBoost)

ADA_5CV_Results=Valid_Index(confusion_matrix(Adaboost_50[:,1],Adaboost_50[:,0]),"Ada_50").append([
Valid_Index(confusion_matrix(Adaboost_100[:,1],Adaboost_100[:,0]),"Ada_100"),
Valid_Index(confusion_matrix(Adaboost_200[:,1],Adaboost_200[:,0]),"Ada_200"),
Valid_Index(confusion_matrix(Adaboost_300[:,1],Adaboost_300[:,0]),"Ada_300"),
Valid_Index(confusion_matrix(Adaboost_1000[:,1],Adaboost_1000[:,0]),"Ada_1000")])

ADA_5CV_Results = ADA_5CV_Results.sort_values(by=['F1'],ascending=False)
ADA_5CV_Results
```

```
CV Complete!
CV Complete!
CV Complete!
CV Complete!
CV Complete!
```

	Model	Accuracy	F1	BCR	Precision	TPR	TNR
0	Ada_300	0.9575	0.776903	0.852311	0.817680	0.740	0.981667
0	Ada_50	0.9580	0.774194	0.841903	0.837209	0.720	0.984444
0	Ada_1000	0.9565	0.774026	0.854459	0.805405	0.745	0.980000
0	Ada_200	0.9565	0.770449	0.846532	0.815642	0.730	0.981667
0	Ada_100	0.9570	0.767568	0.836036	0.835294	0.710	0.984444

분류 문제 5-fold CV – (4) Bagging ANN

- from sklearn.neural_network import MLPClassifier
  from sklearn.ensemble import BaggingClassifier 의 모듈을 이용하여 함수를 구축 (30번 추출)
- 앞서 구축한 함수인 CV_Results_Each_Model을 사용하여 hidden node/layer별 결과를 저장
- 앞서 구축한 함수인 Valid_Index을 사용하여 5-fold CV 분류 성능을 저장

Model4. Bagging ANN

Bagging ANN함수와 ,5-Fold CV를 위해 함수를 생성 후 이행

> 30번 반복 연산하며 오래 걸리므로 (2 Layer/20 Node), (1 Layer/20 Node) 2가지만 시행해봄

```
]: def B_NeuralNetwork(HIDDEN_LAYER,Train_Input,TR_OUTPUT,Val_Input,Val_Output):
       MLP=MLPClassifier(activation=ACTIVATION,solver=SOLVER,alpha=L2_Penalty,
                         hidden_layer_sizes=HIDDEN_LAYER,max_iter=3000,
                         batch_size=BATCH_SIZE,verbose=Visualization,early_stopping=False,power_t=Power_Value,
                         validation_fraction=Validation_Percent,learning_rate=Decay_Method,tol=Tolerence_Value,random_state =RANDOM_STATE)
       BMLP = BaggingClassifier(n_estimators=30, base_estimator=MLP, random_state=RANDOM_STATE, n_jobs=-1).fit(Train_Input, TR_OUTPUT[:, 0])
       Predict_Value = BMLP.predict(Val_Input)
       return(np.concatenate((Predict_Value[:,np.newaxis],Val_Output),axis=1))

   B_ANN_Two_20 = CV_Result_Each_Model([20,20],B_NeuralNetwork)
   B_ANN_One_20 = CV_Result_Each_Model([20],B_NeuralNetwork)

   CV Complete!
   CV Complete!
```

분류 문제 5-fold CV – (4) Bagging ANN

- from sklearn.neural_network import MLPClassifier
  from sklearn.ensemble import BaggingClassifier 의 모듈을 이용하여 함수를 구축 (30번 추출)
- 앞서 구축한 함수인 CV_Results_Each_Model을 사용하여 hidden node/layer별 결과를 저장
- 앞서 구축한 함수인 Valid_Index을 사용하여 5-fold CV 분류 성능을 저장

```
def B_NeuralNetwork(HIDDEN_LAYER,Train_Input,TR_OUTPUT,Val_Input,Val_Output):
    MLP=MLPClassifier(activation=ACTIVATION,solver=SOLVER,alpha=L2_Penalty,
                      hidden_layer_sizes=HIDDEN_LAYER,max_iter=3000,
                      batch_size=BATCH_SIZE,verbose=Visualization,early_stopping=False,power_t=Power_Value,
                      validation_fraction=Validation_Percent,learning_rate=Decay_Method,tol=Tolerence_Value,random_state =RANDOM_STATE)
    BMLP = BaggingClassifier(n_estimators=30, base_estimator=MLP, random_state=RANDOM_STATE, n_jobs=-1).fit(Train_Input, TR_OUTPUT[:, 0])
    Predict_Value = BMLP.predict(Val_Input)
    return(np.concatenate((Predict_Value[:,np.newaxis],Val_Output),axis=1))

B_ANN_Two_20 = CV_Result_Each_Model([20,20],B_NeuralNetwork)
B_ANN_One_20 = CV_Result_Each_Model([20],B_NeuralNetwork)
```

```
CV Complete!
CV Complete!
```

```
B_ANN_Results=pd.concat((Valid_Index(confusion_matrix(B_ANN_Two_20[:,1],B_ANN_Two_20[:,0]),"B_ANN_Two_20"),
          Valid_Index(confusion_matrix(B_ANN_One_20[:,1],B_ANN_One_20[:,0]),"B_ANN_One_20")))

B_ANN_Results = B_ANN_Results.sort_values(by=['F1'],ascending=False)
B_ANN_Results
```

	Model	Accuracy	F1	BCR	Precision	TPR	TNR
0	B_ANN_Two_20	0.9735	0.859416	0.896242	0.915254	0.810	0.991667
0	B_ANN_One_20	0.9730	0.854839	0.888402	0.924419	0.795	0.992778

분류 문제 5-fold CV – (5) Bagging Tree

- from sklearn.tree import DecisionTreeClassifier
  from sklearn.ensemble import BaggingClassifier 의 모듈을 이용하여 함수를 구축 (30번 추출)
- 앞서 구축한 함수인 CV_Results_Each_Model을 사용하여 tree 개수별 결과를 저장
- 앞서 구축한 함수인 Valid_Index을 사용하여 5-fold CV 분류 성능을 저장

Model5. Bagging Decision Tree

Bagging Decision Tree함수와 ,5-Fold CV를 위해 함수를 생성 후 이행

> depth=5, depth=6 2가지만 시행해봄

```
def B_Tree(NUMBER,Train_Input,TR_OUTPUT,Val_Input,Val_Output):
    Tree = DecisionTreeClassifier(max_depth=NUMBER, random_state=RANDOM_STATE).fit(Train_Input, TR_OUTPUT[:, 0])
    BMLP = BaggingClassifier(n_estimators=30, base_estimator=Tree, random_state=RANDOM_STATE, n_jobs=-1).fit(Tra
    Predict_Value = BMLP.predict(Val_Input)
    return(np.concatenate((Predict_Value[:,np.newaxis],Val_Output),axis=1))

B_Tree_D6=CV_Result_Each_Model(6,B_Tree)
B_Tree_D5=CV_Result_Each_Model(5,B_Tree)
```

```
CV Complete!
CV Complete!
```

분류 문제 5-fold CV – (5) Bagging Tree

- from sklearn.tree import DecisionTreeClassifier
  from sklearn.ensemble import BaggingClassifier 의 모듈을 이용하여 함수를 구축 (30번 추출)
- 앞서 구축한 함수인 CV_Results_Each_Model을 사용하여 tree 개수별 결과를 저장
- 앞서 구축한 함수인 Valid_Index을 사용하여 5-fold CV 분류 성능을 저장

Model5. Bagging Decision Tree

Bagging Decision Tree함수와 ,5-Fold CV를 위해 함수를 생성 후 이행

> depth=5, depth=6 2가지만 시행해봄

```
def B_Tree(NUMBER,Train_Input,TR_OUTPUT,Val_Input,Val_Output):
    Tree = DecisionTreeClassifier(max_depth=NUMBER, random_state=RANDOM_STATE).fit(Train_Input, TR_OUTPUT[:, 0])
    BMLP = BaggingClassifier(n_estimators=30, base_estimator=Tree, random_state=RANDOM_STATE, n_jobs=-1).fit(Tra
    Predict_Value = BMLP.predict(Val_Input)
    return(np.concatenate((Predict_Value[:,np.newaxis],Val_Output),axis=1))

B_Tree_D6=CV_Result_Each_Model(6,B_Tree)
B_Tree_D5=CV_Result_Each_Model(5,B_Tree)
```

```
CV Complete!
CV Complete!
```

```
B_Tree_Results=pd.concat((Valid_Index(confusion_matrix(B_Tree_D6[:,1],B_Tree_D6[:,0]),"B_Tree_D6"),
          Valid_Index(confusion_matrix(B_Tree_D5[:,1],B_Tree_D5[:,0]),"B_Tree_D5")))

B_Tree_Results = B_Tree_Results.sort_values(by=['F1'],ascending=False)
B_Tree_Results
```

	Model	Accuracy	F1	BCR	Precision	TPR	TNR
0	B_Tree_D6	0.982	0.906250	0.930143	0.945652	0.87	0.994444
0	B_Tree_D5	0.982	0.905263	0.925299	0.955556	0.86	0.995556

분류 문제 5-fold CV – (6) GBM

- from sklearn.ensemble import GradientBoostingClassifier의 모듈을 이용하여 함수를 구축
- 앞서 구축한 함수인 CV_Results_Each_Model을 사용하여 tree 개수별 결과를 저장
- 앞서 구축한 함수인 Valid_Index을 사용하여 5-fold CV 분류 성능을 저장

3.2.4.3.5. sklearn.ensemble.GradientBoostingClassifier

class sklearn.ensemble. GradientBoostingClassifier (*loss='deviance', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, min_impurity_split=None, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto'*)

Model6. Gradient Boosting Machine

Bagging Decision Tree함수와 ,5-Fold CV를 위해 함수를 생성 후 이행

```
def GBM(NUMBER,Train_Input,TR_OUTPUT,Val_Input,Val_Output):
    gbm=GradientBoostingClassifier(n_estimators=NUMBER,random_state=RANDOM_STATE).fit(Train_Input,TR_OUTPUT[:,0]
    Predict_Value = gbm.predict(Val_Input)
    return(np.concatenate((Predict_Value[:,np.newaxis],Val_Output),axis=1))

GBM_100 = CV_Result_Each_Model(100,GBM)
GBM_150 = CV_Result_Each_Model(150,GBM)
GBM_200 = CV_Result_Each_Model(200,GBM)
GBM_250 = CV_Result_Each_Model(250,GBM)
GBM_300 = CV_Result_Each_Model(300,GBM)
```

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

분류 문제 5-fold CV – (6) GBM

- from sklearn.ensemble import GradientBoostingClassifier의 모듈을 이용하여 함수를 구축
- 앞서 구축한 함수인 CV_Results_Each_Model을 사용하여 tree 개수별 결과를 저장
- 앞서 구축한 함수인 Valid_Index을 사용하여 5-fold CV 분류 성능을 저장

Model6. Gradient Boosting Machine

Bagging Decision Tree함수와 ,5-Fold CV를 위해 함수를 생성 후 이행

```
def GBM(NUMBER,Train_Input,TR_OUTPUT,Val_Input,Val_Output):
    gbm=GradientBoostingClassifier(n_estimators=NUMBER,random_state=RANDOM_STATE).fit(Train_Input,TR_OUTPUT[:,0]
    Predict_Value = gbm.predict(Val_Input)
    return(np.concatenate((Predict_Value[:,np.newaxis],Val_Output),axis=1))

GBM_100 = CV_Result_Each_Model(100,GBM)
GBM_150 = CV_Result_Each_Model(150,GBM)
GBM_200 = CV_Result_Each_Model(200,GBM)
GBM_250 = CV_Result_Each_Model(250,GBM)
GBM_300 = CV_Result_Each_Model(300,GBM)

GBM_5CV_Results=Valid_Index(confusion_matrix(GBM_100[:,1],GBM_100[:,0]),"GBM_100").append([
Valid_Index(confusion_matrix(GBM_150[:,1],GBM_150[:,0]),"GBM_150"),
Valid_Index(confusion_matrix(GBM_200[:,1],GBM_200[:,0]),"GBM_200"),
Valid_Index(confusion_matrix(GBM_250[:,1],GBM_250[:,0]),"GBM_250"),
Valid_Index(confusion_matrix(GBM_300[:,1],GBM_300[:,0]),"GBM_300")])

GBM_5CV_Results = GBM_5CV_Results.sort_values(by=['F1'],ascending=False)
print(GBM_5CV_Results)
```

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

분류 문제 Test

- 5-Fold Cross validation으로 선택된 6가지 모델의 최적의 하이퍼파라미터 모델의 테스트 데이터의 결과임
- Code : Classification_Test.ipynb

Model	Accuracy	F1	BCR	Precision	TPR	TNR
GBM	0.984	0.928571	0.959274	0.928571	0.928571	0.990991
RandomForset	0.984	0.927273	0.951084	0.944444	0.910714	0.993243
B_Tree	0.982	0.920354	0.958183	0.912281	0.928571	0.988739
Neuralnetwork	0.982	0.917431	0.941714	0.943396	0.892857	0.993243
B_Neuralnetwork	0.978	0.900901	0.939576	0.909091	0.892857	0.988739
Adaboost	0.978	0.895238	0.914060	0.959184	0.839286	0.995495

분류 문제 Test

- RandomForest와 Gradient Boosting Machine에서 산출된 변수 중요도는 아래와 같음

Feature Importance

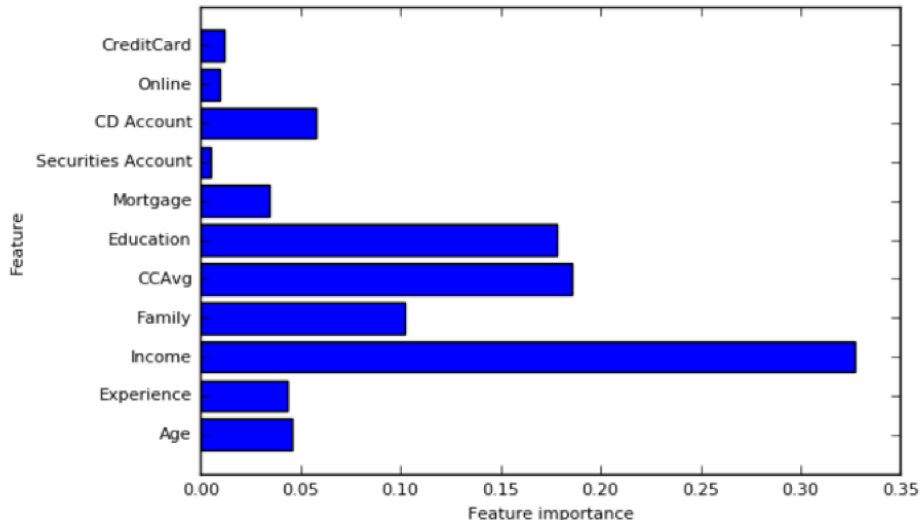
```
Use_Feature_Name=list(np.array(list(Rawdata.columns.values))[np.concatenate((range(1,4),range(5,9),range(10,14)))])
OP_RF=RandomForestClassifier(n_estimators=150,max_features="sqrt",
                             random_state=RANDOM_STATE).fit(Train_Input_Normalized,Train_Output[:,0])
OP_GBM=GradientBoostingClassifier(n_estimators=50,
                             random_state=RANDOM_STATE).fit(Train_Input_Normalized,Train_Output[:,0])

def plot_feature_importances(model):
    n_features = np.shape(Train_Input_Normalized)[1]
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), Use_Feature_Name)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature")
    plt.ylim(-1, n_features)
    plt.rcParams.update({'font.size': 8})
```

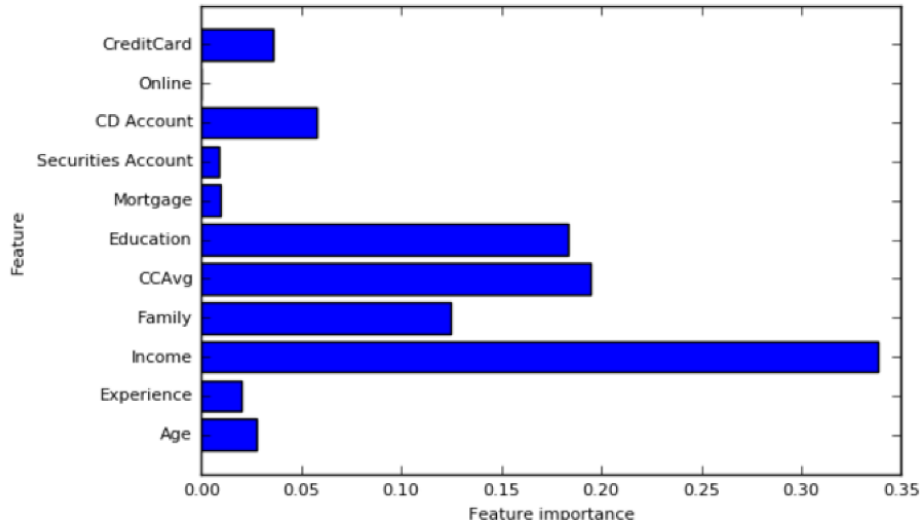
분류 문제 Test

- RandomForest와 Gradient Boosting Machine에서 산출된 변수 중요도는 아래와 같음

```
plot_feature_importances(OP_RF)
```




```
plot_feature_importances(OP_GBM)
```



앙상블 실습

- 분류 문제

1. 3-Cross validation을 이용한 파라미터 튜닝 및 테스트데이터 예측
2. dataset : mortgage http://sci2s.ugr.es/keel/dataset.php?cod=43
3. 사용 알고리즘
    - 3-(1) ANN
    - 3-(2) Bagging ANN
    - 3-(3) RandomForest
    - 3-(4) Bagging Tree
    - 3-(5) Gradient boosting machine

- Code

1. Regression_CV.ipynb
2. Regression_Test.ipynb

회귀 문제 3-fold CV

- 사용할 모듈을 불러옴

필요한 모듈을 불러온다

In [1]:
```
import numpy as np
import copy
import pandas as pd
import matplotlib.pyplot as plt
import collections
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
print("Module Ready!")
```

```
Module Ready!
```

회귀 문제 3-fold CV

- **분류와 같이 8:2로 데이터를 나누고 3 fold CV를 하며 standardization해줌**

사용할 데이터를 불러오며 데이터 전처리함

> *1. Fold_Value는 CV Fold갯수를 의미하며 3으로 입력*
>
> *2. 사용할 데이터 mortgage를 불러오며 standardization함*
>
> *3. 난수 고정하여 training:test = 8:2 사용*

```
#############################
FOLD_VALUE = 3
RANDOM_STATE = 1026
#############################

def mean_absolute_percentage_error(Data):
    return(np.mean(np.abs(Data[:, 1]-Data[:, 0])/Data[:, 1]))

def Valid_Index(Data,Name):
    return(pd.DataFrame(pd.Series({'MAE':mean_absolute_error(Data[:, 0], Data[:, 1]),
                'MSE': mean_squared_error(Data[:, 0], Data[:, 1]),
                'MAPE': mean_absolute_percentage_error(Data),
                'Model': Name})).transpose())


# 사용할 Personal Loan 데이터셋을 불러옵니다.
Rawdata = pd.read_csv('dataset/mortgage.csv')
# Print Column names
print("'mortgage' data column name : ", list(Rawdata.columns.values))
# Allocate column index based on Input and Output varaibles

Input_Column_Index = range(0,15)
Target_Column_Index = np.array([15])

# 같은 데이터셋을 사용하기 위해서 난수를 고정합니다.
np.random.seed(100)
Train_Index = np.random.choice(np.shape(Rawdata)[0],int(np.shape(Rawdata)[0]*0.8),replace=False)

# Input variable과 Output variable을 Numpy array로 변환합니다.
Rawdata_Input = np.array(Rawdata)[:,Input_Column_Index]
Rawdata_Output = np.array(Rawdata)[:,Target_Column_Index]

# Training data와 Test data를 나누어 줍니다.
Train_Input = Rawdata_Input[Train_Index,:]
Train_Output = Rawdata_Output[Train_Index,:]
Test_Input = Rawdata_Input[np.delete(range(np.shape(Rawdata)[0]),Train_Index),:]
Test_Output = Rawdata_Output[np.delete(range(np.shape(Rawdata)[0]),Train_Index),:]
print('Data partition complete! ₩nTrain_Input shape :',np.shape(Train_Input),'₩nTrain_Output shape :',np.shape(Train_Output))
print('Test_Input shape :',np.shape(Test_Input),'₩nTest_Output shape :',np.shape(Test_Output))

def standardization(Data):
```

```
def standardization(Data):
    return ((Data - np.mean(Data, axis=0)) / np.std(Data, axis=0))

Train_Input_Normalized = copy.deepcopy(standardization(Train_Input))
Test_Input_Normalized = copy.deepcopy(standardization(Train_Input))

print("Data standardization complete!")


# Best Hyperparameter를 찾기위하여 3-Fold Cross Validation을 한다
def k_Fold_Maker(InputData,OutputData,Partition_Number):
    Index = 0
    Input_List = list()
    Output_List = list()
    Length = int(np.floor(np.shape(InputData)[0]/Partition_Number))
    for i in range(Partition_Number):
        if(i == (Partition_Number-1)):
            Input_List.append(InputData[range(Index+(Length* i), np.shape(InputData)[0]), :])
            Output_List.append(OutputData[range(Index+(Length* i), np.shape(InputData)[0]), :])
        else:
            Input_List.append(InputData[range(Index + (Length * i), Index + (Length * (i + 1))), :])
            Output_List.append(OutputData[range(Index + (Length * i), Index + (Length * (i + 1))), :])
    return(Input_List,Output_List)


# Make 3-Fold dataset for Cross validation
Fold_Input, Fold_Output = k_Fold_Maker(Train_Input_Normalized, Train_Output, FOLD_VALUE)

# Cross validation을 위한 함수
def CV_Result_Each_Model(Hyper_Para,Model):
# 모든 결과를 담을 객체
    FULL_Results = list()
    for i in range(FOLD_VALUE):
        Tr_Index = np.delete(range(FOLD_VALUE),i) #Training 에 사용할 Fold Index
        Val_Index = i                             #Validation에 사용할 Fold Index

        TRAIN_INPUT  = list()
        TRAIN_OUTPUT = list()
        for j in Tr_Index:
            TRAIN_INPUT.append(Fold_Input[j])
            TRAIN_OUTPUT.append(Fold_Output[j])
        TRAIN_INPUT = np.concatenate(TRAIN_INPUT)
        TRAIN_OUTPUT = np.concatenate(TRAIN_OUTPUT)
        VALID_INPUT = Fold_Input[i]
        VALID_OUTPUT = Fold_Output[i]
        FULL_Results.append(Model(Hyper_Para,TRAIN_INPUT,TRAIN_OUTPUT,VALID_INPUT,VALID_OUTPUT))
    print("CV Complete!")
    FULL_Results=np.concatenate(FULL_Results,axis=0)
    return(FULL_Results)
```

```
'mortgage' data column name :  ['OneMonthCDRate', 'OneY.CMaturityRate', 'ThreeM.Rate.AuctionAverage', 'ThreeM.Rate.SecondaryMarket', 'ThreeY.CMaturityRate', 'FiveY.CMaturityRate', 'BankCredit', 'Currency', 'DemandDeposits', 'FederalFunds', 'MoneyStock', 'CheckableDeposits', 'LoansLeases', 'SavingsDeposits', 'TradeCurrencies', 'ThirtyY.CMortgageRate']
Data partition complete!
Train_Input shape : (839, 15)
Train_Output shape : (839, 1)
Test_Input shape : (210, 15)
Test_Output shape : (210, 1)
Data standardization complete!
```

회귀 문제 3-fold CV – (1) ANN

- from sklearn.neural_network import MLPRegressor 의 모듈을 이용하여 함수를 구축
- 앞서 구축한 함수인 CV_Results_Each_Model을 사용하여 Hidden layer/node별 결과를 저장
- 앞서 구축한 함수인 Valid_Index을 사용하여 3-fold CV 분류 성능을 저장

sklearn.neural_network.MLPRegressor

class sklearn.neural_network. **MLPRegressor** (*hidden_layer_sizes=(100,), activation='relu', solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08*) [source]

Model1. Neural network

NeuralNetwork 함수와 3Fold CV를 위해 함수를 생성 후 이행

```
def NeuralNetwork(HIDDEN_LAYER,Train_Input,TR_OUTPUT,Val_Input,Val_Output):
    MLP=MLPRegressor(hidden_layer_sizes=HIDDEN_LAYER,max_iter=2000 ,random_state =RANDOM_STATE).fit(Train_Input,TR_OUTPUT[:,0])
    Predict_Value = MLP.predict(Val_Input)
    return(np.concatenate((Predict_Value[:,np.newaxis],Val_Output),axis=1))
```

http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html

회귀 문제 3-fold CV – (2) Bagging ANN

- from sklearn.neural_network import MLPRegressor
  from sklearn.ensemble import BaggingRegressor 의 모듈을 이용하여 함수를 구축 (30번 추출)
- 앞서 구축한 함수인 CV_Results_Each_Model을 사용하여 hidden node/layer별 결과를 저장
- 앞서 구축한 함수인 Valid_Index을 사용하여 3-fold CV 분류 성능을 저장

sklearn.ensemble.BaggingRegressor

class sklearn.ensemble. BaggingRegressor (*base_estimator=None*, *n_estimators=10*, *max_samples=1.0*, *max_features=1.0*, *bootstrap=True*, *bootstrap_features=False*, *oob_score=False*, *warm_start=False*, *n_jobs=1*, *random_state=None*, *verbose=0*) [source]

Model2. Bagging Neural network

NeuralNetwork 함수와 3Fold CV를 위해 함수를 생성 후 이행

```
def B_NeuralNetwork(HIDDEN_LAYER,Train_Input,TR_OUTPUT,Val_Input,Val_Output):
    MLP=MLPRegressor(hidden_layer_sizes=HIDDEN_LAYER,max_iter=2000 ,random_state =RANDOM_STATE)
    BMLP = BaggingRegressor(n_estimators=30, base_estimator=MLP,
                            random_state=RANDOM_STATE, n_jobs=-1).fit(Train_Input, TR_OUTPUT[:, 0])
    Predict_Value = BMLP.predict(Val_Input)
    return(np.concatenate((Predict_Value[:,np.newaxis],Val_Output),axis=1))
```

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html

회귀 문제 3-fold CV – (3) RandomForest

- from sklearn.ensemble import RandomForestRegressor 의 모듈을 이용하여 함수를 구축
- 앞서 구축한 함수인 CV_Results_Each_Model을 사용하여 tree 개수별 결과를 저장
- 앞서 구축한 함수인 Valid_Index을 사용하여 3-fold CV 분류 성능을 저장

3.2.4.3.2. sklearn.ensemble.RandomForestRegressor

class sklearn.ensemble. **RandomForestRegressor** (*n_estimators=10, criterion='mse', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False*) [source]

Model3. RandomForest

RandomForest 함수와 3Fold CV를 위해 함수를 생성 후 이행

```
def RandomForest(NUMBER,Train_Input,TR_OUTPUT,Val_Input,Val_Output):
    RF=RandomForestRegressor(n_estimators=NUMBER,
                             max_features="sqrt",
                             random_state=RANDOM_STATE).fit(Train_Input,TR_OUTPUT[:,0])
    Predict_Value = RF.predict(Val_Input)
    return(np.concatenate((Predict_Value[:,np.newaxis],Val_Output),axis=1))

RF50_CV = CV_Result_Each_Model(50,RandomForest)
RF100_CV = CV_Result_Each_Model(100,RandomForest)
RF150_CV = CV_Result_Each_Model(150,RandomForest)
RF200_CV = CV_Result_Each_Model(200,RandomForest)
```

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

회귀 문제 3-fold CV – (4) Bagging Tree

- from sklearn.tree import DecisionTreeRegressor
  from sklearn.ensemble import BaggingRegressor 의 모듈을 이용하여 함수를 구축 (30번 추출)
- 앞서 구축한 함수인 CV_Results_Each_Model을 사용하여 tree 개수별 결과를 저장
- 앞서 구축한 함수인 Valid_Index을 사용하여 3-fold CV 분류 성능을 저장

sklearn.tree.DecisionTreeRegressor

class sklearn.tree. DecisionTreeRegressor (*criterion='mse', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, presort=False*) [source]

Model4. Bagging Tree

Bagging Tree 함수와 3Fold CV를 위해 함수를 생성 후 이행

```
def B_Tree(NUMBER,Train_Input,TR_OUTPUT,Val_Input,Val_Output):
    Tree = DecisionTreeRegressor(max_depth=NUMBER, random_state=RANDOM_STATE).fit(Train_Input, TR_OUTPUT[:, 0])
    BMLP = BaggingRegressor(n_estimators=30, base_estimator=Tree, random_state=RANDOM_STATE, n_jobs=-1).fit(Train_Input, TR_OUTPUT[:, 0])
    Predict_Value = BMLP.predict(Val_Input)
    return(np.concatenate((Predict_Value[:,np.newaxis],Val_Output),axis=1))

# 자 이제 NeuralNet을 기준으로 여러가지를 3-Fold CV를 해보도록 한다.
B_Tree_D6 = CV_Result_Each_Model(6,B_Tree)
B_Tree_D5 = CV_Result_Each_Model(5,B_Tree)
```

http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

회귀 문제 3-fold CV – (5) GBM

- from sklearn.ensemble import GradientBoostingRegressor의 모듈을 이용하여 함수를 구축
- 앞서 구축한 함수인 CV_Results_Each_Model을 사용하여 tree 개수별 결과를 저장
- 앞서 구축한 함수인 Valid_Index을 사용하여 3-fold CV 분류 성능을 저장

3.2.4.3.6. sklearn.ensemble.GradientBoostingRegressor

class sklearn.ensemble. GradientBoostingRegressor (*loss='ls', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, min_impurity_split=None, init=None, random_state=None, max_features=None, alpha=0.9, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto'*) [source]

Model5. Gradient Boosting Machine

Gradient Boosting Machine 함수와 3Fold CV를 위해 함수를 생성 후 이행

```
def GBM(NUMBER,Train_Input,TR_OUTPUT,Val_Input,Val_Output):
    gbm=GradientBoostingRegressor(n_estimators=NUMBER,random_state=RANDOM_STATE).fit(Train_Input,TR_OUTPUT[:,0])
    Predict_Value = gbm.predict(Val_Input)
    return(np.concatenate((Predict_Value[:,np.newaxis],Val_Output),axis=1))
```

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

회귀 문제 Test

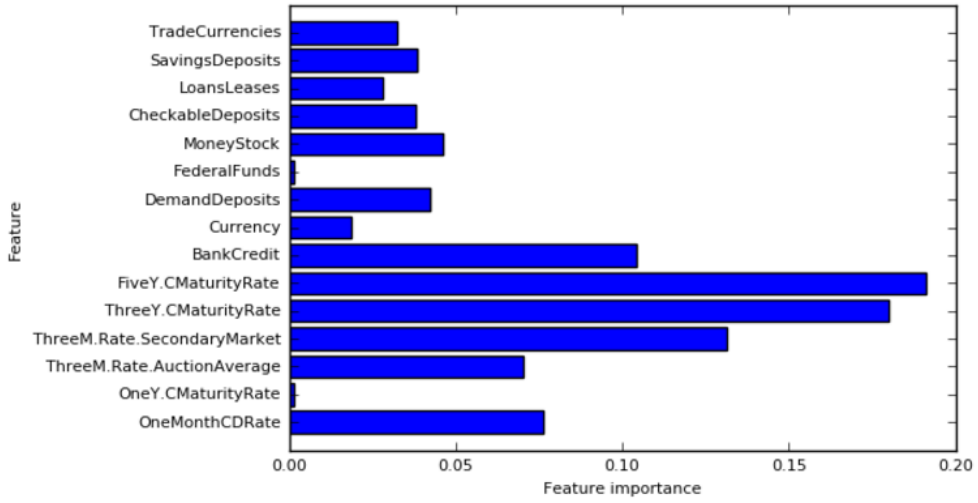
- 3-Fold Cross validation으로 선택된 5가지 모델의 최적의 하이퍼파라미터 모델의 테스트 데이터의 결과임
- Code : Regression_Test.ipynb

	MAE	MAPE	MSE	Model
0	0.0736954	0.00962527	0.0122758	GBM
0	0.0752348	0.00964126	0.0158675	RF
0	0.110823	0.0157357	0.0248484	B_Tree
0	0.137146	0.021833	0.0315289	ANN
0	0.13331	0.0194384	0.0329916	B_ANN

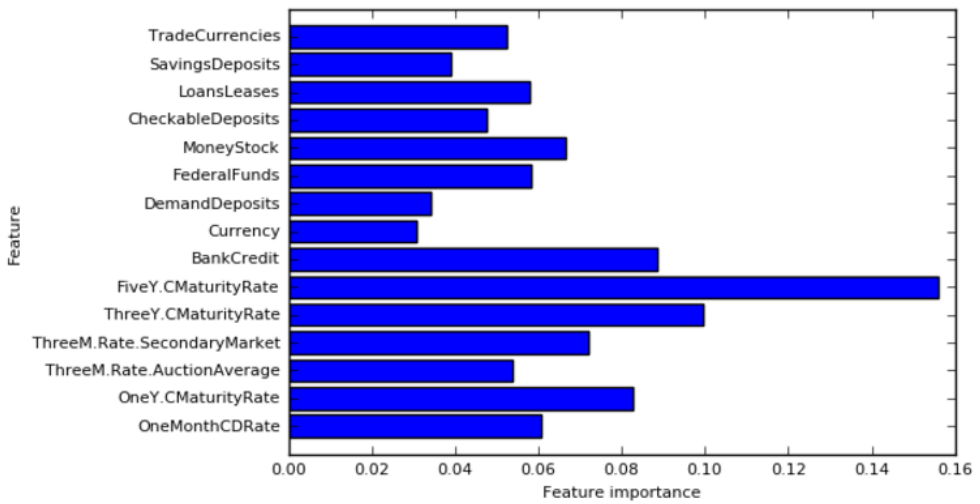
회귀 문제 Test

- RandomForest와 Gradient Boosting Machine에서 산출된 변수 중요도는 아래와 같음

```
plot_feature_importances(OP_RF)
```




```
plot_feature_importances(OP_GBM)
```



Q & A