COMP5318 Assignment 1: Classification

Group number: 23 , SID1: 490017845, SID2: 470341997

NOTE: The version of the sklearn is 1.0.2. Hence, the Adaboost result is different.

In [17]:
```
# Import all libraries for the data preprocessing
from sklearn.model_selection import StratifiedKFold
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
import warnings
warnings.filterwarnings('ignore')
```

In [18]:
```
# Load dataset
raw_dataset = pd.read_csv("breast-cancer-wisconsin.csv")
# First, replace the ? with np.nan value
raw_dataset = raw_dataset.replace("?", np.nan)
"""
Split the dataset into data(columns without class) and target(class column)
1. Store the class column
2. Remove the class column from the dataset
3. Store the new dataset to a new address
"""
target = raw_dataset['class']
del raw_dataset['class']
data = raw_dataset
```

In [19]:
```
# Pre-process dataset
#FILL THE MISSING VALUE
# https://towardsdatascience.com/imputing-missing-values-using-the-simpleimputer-class-in-sklearn-99706afaff46
imputer = SimpleImputer(strategy='mean', missing_values=np.nan)
imputer = imputer.fit(data)
data = imputer.transform(data)
```

In [20]:
```
#NORMALISING THE DATA
scaler = MinMaxScaler()
scaler.fit(data)
data = scaler.transform(data)
```

In [21]:
```
#CHANGING CLASS VALUE
target = target.replace("class1", 0)
target = target.replace("class2", 1)
target = target.values
```

In [22]:
```
# Print first ten rows of pre-processed dataset to 4 decimal places as per assignment spec
# A function is provided to assist

def print_data(X, y, n_rows=10):
    """Takes a numpy data array and target and prints the first ten rows.

    Arguments:
        X: numpy array of shape (n_examples, n_features)
        y: numpy array of shape (n_examples)
        n_rows: numpy of rows to print
    """
    for example_num in range(n_rows):
        for feature in X[example_num]:
            print("{:.4f}".format(feature), end=",")

        if example_num == len(X)-1:
            print(y[example_num],end="")
        else:
            print(y[example_num])


print_data(data, target, 10)
```

```
0.4444,0.0000,0.0000,0.0000,0.1111,0.0000,0.2222,0.0000,0.0000,0
0.4444,0.3333,0.3333,0.4444,0.6667,1.0000,0.2222,0.1111,0.0000,0
0.2222,0.0000,0.0000,0.0000,0.1111,0.1111,0.2222,0.0000,0.0000,0
0.5556,0.7778,0.7778,0.0000,0.2222,0.3333,0.2222,0.6667,0.0000,0
0.3333,0.0000,0.0000,0.2222,0.1111,0.0000,0.2222,0.0000,0.0000,0
0.7778,1.0000,1.0000,0.7778,0.6667,1.0000,0.8889,0.6667,0.0000,1
0.0000,0.0000,0.0000,0.0000,0.1111,1.0000,0.2222,0.0000,0.0000,0
0.1111,0.0000,0.1111,0.0000,0.1111,0.0000,0.2222,0.0000,0.0000,0
0.1111,0.0000,0.0000,0.0000,0.1111,0.0000,0.0000,0.0000,0.4444,0
0.3333,0.1111,0.0000,0.0000,0.1111,0.0000,0.1111,0.0000,0.0000,0
```

Part 1: Cross-validation without parameter tuning

In [23]:
```
## Setting the 10 fold stratified cross-validation
cvKFold=StratifiedKFold(n_splits=10, shuffle=True, random_state=0)
# The stratified folds from cvKFold should be provided to the classifiers
# Pass the cvKFold object to cv parameter via ED forum
```

In [24]:
```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
def logregClassifier(X, y):
    logreg = LogisticRegression()
    scores = cross_val_score(logreg, X, y, cv=cvKFold)

    return scores.mean()
```

In [25]:
```
#Naïve Bayes
from sklearn.naive_bayes import GaussianNB
def nbClassifier(X, y):
    nb = GaussianNB()
    scores = cross_val_score(nb, X, y, cv=cvKFold)

    return scores.mean()
```

In [26]:
```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
def dtClassifier(X, y):
    tree = DecisionTreeClassifier(criterion='entropy', random_state=0)
    scores = cross_val_score(tree, X, y, cv=cvKFold)

    return scores.mean()
```

In [27]:
```
# Ensembles: Bagging, Ada Boost and Gradient Boosting
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier

def bagDTClassifier(X, y, n_estimators, max_samples, max_depth):
    tree = DecisionTreeClassifier(criterion='entropy',
                                  max_depth=max_depth, random_state=0)
    bag_clf = BaggingClassifier(tree, n_estimators=n_estimators,
                                max_samples=max_samples, bootstrap=True, random_state=0)
    scores = cross_val_score(bag_clf, X, y, cv=cvKFold)

    return scores.mean()

def adaDTClassifier(X, y, n_estimators, learning_rate, max_depth):
    tree = DecisionTreeClassifier(criterion='entropy',
                                  max_depth=max_depth, random_state=0)
    ada_clf = AdaBoostClassifier(tree, n_estimators=n_estimators,
                                 learning_rate=learning_rate, random_state=0)
    scores = cross_val_score(ada_clf, X, y, cv=cvKFold)

    return scores.mean()

def gbClassifier(X, y, n_estimators, learning_rate):
    gb_clf = GradientBoostingClassifier(n_estimators=n_estimators,
                                        learning_rate=learning_rate, random_state=0)
    scores = cross_val_score(gb_clf, X, y, cv=cvKFold)

    return scores.mean()
```

Part 1 Results

In [28]:
```
# Parameters for Part 1:

#Bagging
bag_n_estimators = 60
bag_max_samples = 100
bag_max_depth = 6

#AdaBoost
ada_n_estimators = 60
ada_learning_rate = 0.5
ada_bag_max_depth = 6

#GB
gb_n_estimators = 60
gb_learning_rate = 0.5

logR_acc = logregClassifier(data, target)
NB_acc = nbClassifier(data, target)
DT_acc = dtClassifier(data, target)
bagging_acc = bagDTClassifier(data, target, bag_n_estimators,
                              bag_max_samples, bag_max_depth)
```

```
ada_acc = adaDTClassifier(data, target, ada_n_estimators,
                          ada_learning_rate, ada_bag_max_depth)
gb_acc = gbClassifier(data, target, gb_n_estimators, gb_learning_rate)

# Print results for each classifier in part 1 to 4 decimal places here:
print("LogR average cross-validation accuracy: {:.4f}".format(logR_acc))
print("NB average cross-validation accuracy:{:.4f}".format(NB_acc))
print("DT average cross-validation accuracy:{:.4f}".format(DT_acc))
print("Bagging average cross-validation accuracy:{:.4f}".format(bagging_acc))
print("AdaBoost average cross-validation accuracy:{:.4f}".format(ada_acc))
print("GB average cross-validation accuracy:{:.4f}".format(gb_acc))
```

```
LogR average cross-validation accuracy: 0.9642
NB average cross-validation accuracy:0.9585
DT average cross-validation accuracy:0.9385
Bagging average cross-validation accuracy:0.9571
AdaBoost average cross-validation accuracy:0.9542
GB average cross-validation accuracy:0.9613
```

Part 2: Cross-validation with parameter tuning

In [29]:
```
# KNN
k = [1, 3, 5, 7, 9]
p = [1, 2]
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
def bestKNNClassifier(X, y):
    #split the data
    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=target, random_state=0)

    param_grid = {'n_neighbors': k,
              'p': p}
    grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=cvKFold,
                           return_train_score=True)


    grid_search.fit(X_train, y_train)
    return grid_search
```

In [30]:
```
# SVM
# You should use SVC from sklearn.svm with kernel set to 'rbf'
C = [0.01, 0.1, 1, 5, 15]
gamma = [0.01, 0.1, 1, 10, 50]

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
def bestSVMClassifier(X, y):
    #split the data
    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=target, random_state=0)

    param_grid = {'C': C,
              'gamma': gamma}
    grid_search = GridSearchCV(SVC(kernel="rbf"), param_grid, cv=cvKFold,
                           return_train_score=True)

    grid_search.fit(X_train, y_train)
    return grid_search
```

In [31]:
```
# Random Forest
# You should use RandomForestClassifier from sklearn.ensemble with information gain and max_features set to ‘sq
n_estimators = [10, 30, 60, 100, 150]
max_leaf_nodes = [6, 12, 18]

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

def bestRFClassifier(X, y):
    #split the data
    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=target, random_state=0)

    param_grid = {'n_estimators': n_estimators,
              'max_leaf_nodes': max_leaf_nodes}

    grid_search = GridSearchCV(
        RandomForestClassifier(criterion='entropy', max_features='sqrt', random_state=0),
                               param_grid, cv=cvKFold, return_train_score=True)

    grid_search.fit(X_train, y_train)
    return grid_search
```

Part 2: Results

In [32]:
```
# Perform Grid Search with 10-fold stratified cross-validation (GridSearchCV in sklearn).
```

```
# The stratified folds from cvKFold should be provided to GridSearchV

# This should include using train_test_split from sklearn.model_selection with stratification and random_state=
# Print results for each classifier here. All results should be printed to 4 decimal places except for
# "k", "p", n_estimators" and "max_leaf_nodes" which should be printed as integers.
from sklearn.model_selection import train_test_split

#split the data
X_train, X_test, y_train, y_test = train_test_split(data, target, stratify=target, random_state=0)

#retrieve the result
knn_result = bestKNNClassifier(data, target)
svm_result = bestSVMClassifier(data, target)
rf_result = bestRFClassifier(data, target)

print("KNN best k:", knn_result.best_params_['n_neighbors'])
print("KNN best p:", knn_result.best_params_['p'])
print("KNN cross-validation accuracy: {:.4f}".format(knn_result.best_score_))
print("KNN test set accuracy: {:.4f}".format(knn_result.score(X_test, y_test)))

print()

print("SVM best C: {:.4f}".format(svm_result.best_params_['C']))
print("SVM best gamma: {:.4f}".format(svm_result.best_params_['gamma']))
print("SVM cross-validation accuracy: {:.4f}".format(svm_result.best_score_))
print("SVM test set accuracy: {:.4f}".format(svm_result.score(X_test, y_test)))

print()

from sklearn.metrics import f1_score

y_pred = rf_result.predict(X_test)
print("RF best n_estimators:", rf_result.best_params_['n_estimators'])
print("RF best max_leaf_nodes:", rf_result.best_params_['max_leaf_nodes'])
print("RF cross-validation accuracy: {:.4f}".format(rf_result.best_score_))
print("RF test set accuracy: {:.4f}".format(rf_result.score(X_test, y_test)))
print("RF test set macro average F1: {:.4f}"
      .format(f1_score(y_test, y_pred, average='macro')))
print("RF test set weighted average F1: {:.4f}"
      .format(f1_score(y_test, y_pred, average='weighted')))
```

```
KNN best k: 3
KNN best p: 1
KNN cross-validation accuracy: 0.9695
KNN test set accuracy: 0.9543

SVM best C: 5.0000
SVM best gamma: 0.1000
SVM cross-validation accuracy: 0.9676
SVM test set accuracy: 0.9714

RF best n_estimators: 150
RF best max_leaf_nodes: 6
RF cross-validation accuracy: 0.9675
RF test set accuracy: 0.9657
RF test set macro average F1: 0.9628
RF test set weighted average F1: 0.9661
```