COMP5318 Assignment 1: Classification

Group number: A1part2 100 , SID1: 520638064 , SID2: 520026168

In [1]:
```
# Import all libraries
from sklearn.model_selection import StratifiedKFold
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [2]:
```
# Load the specific dataset
dataset = pd.read_csv('breast-cancer-wisconsin.csv')
```

In [3]:
```
# Pre-process the dataset

# Replace the missing value "?" to NA, and replace the classes 'class1' and 'class2' to 0 and 1
dataset.replace(['?', 'class1', 'class2'], [np.nan, 0, 1], inplace=True)

# Using the sklearn.impute.SimpleImputer to replace the missing value to the mean value of the colums
imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
dataset = imp_mean.fit_transform(dataset)

# Normalise the features values between [0,1]
scaler = MinMaxScaler(feature_range=(0, 1), copy=True)
dataset = scaler.fit_transform(dataset)

# Separate the dataset to the features and class, the last column is the class
X_data = dataset[:, :-1]
y_data = dataset[:, -1].astype(int)
```

In [4]:
```
# 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 the first 10 rows of the pre-processed dataset
print_data(X_data, y_data)
```

```
0.4444,0.0000,0.0000,0.0000,0.1111,0.0000,0.2222,0.0000,0.0000,0
0.4444,0.3333,0.3333,0.4444,0.4667,1.0000,0.2222,0.1111,0.0000,0
0.2222,0.0000,0.0000,0.0000,0.1111,0.2222,0.2222,0.0000,0.0000,0
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
0.7778,1.0000,1.0000,0.7778,0.4667,1.0000,0.8889,0.6667,0.0000,1
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 [5]:
```
## 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
```

In [6]:
```
# Logistic Regression
def logregClassifier(X, y):
    """
    Takes a numpy data array and target and return the average accuracy
    Achieve the Logistic Regression Classifier

    :param X: numpy array of shape (n_examples, n_features)
    :param y: numpy array of shape (n_examples)
    :return: the average accuracy of classifier
    """
    # Initialize the Logistic Regression classifier
    logreg = LogisticRegression(random_state=0)
    # Calculate the cross-validation score
    scores = cross_val_score(logreg, X, y, cv=cvKFold)
    return scores.mean()
```

In [7]:
```
#Naive Bayes
def nbClassifier(X, y):
    """
    Takes a numpy data array and target and return the average accuracy
    Achieve the Naive Bayes Classifier

    :param X: numpy array of shape (n_examples, n_features)
    :param y: numpy array of shape (n_examples)
    :return: the average accuracy of classifier
    """
    # Initialize the Naive Bayes classifier
    nb = GaussianNB()
    # Calculate the cross-validation score
    scores = cross_val_score(nb, X, y, cv=cvKFold)
    return scores.mean()
```

In [8]:
```
# Decision Tree
def dtClassifier(X, y):
    """
    Takes a numpy data array and target and return the average accuracy
    Achieve the Decision Tree Classifier

    :param X: numpy array of shape (n_examples, n_features)
    :param y: numpy array of shape (n_examples)
    :return: the average accuracy of classifier
    """
    # Initialise the Decision Tree classifier
    dt = DecisionTreeClassifier(random_state=0, criterion="entropy")
    # Calculate the cross-validation score
    scores = cross_val_score(dt, X, y, cv=cvKFold)
    return scores.mean()
```

In [9]:
```
# Ensembles: Bagging, Ada Boost and Gradient Boosting
def bagDTClassifier(X, y, n_estimators, max_samples, max_depth):
    """
    Takes a numpy data array and target and return the average accuracy
    Achieve the Bagging Classifier

    :param X: numpy array of shape (n_examples, n_features)
    :param y: numpy array of shape (n_examples)
    :param n_estimators: number of base estimators in the ensemble
    :param max_samples: number of samples to draw from X to train each base estimator
    :param max_depth: maximum depth of the tree
    :return: the average accuracy of classifier
    """
    # Initialise the Bagging classifier using the Decision Tree classifier
    dt = DecisionTreeClassifier(random_state=0, criterion="entropy", max_depth=max_depth)
    bag = BaggingClassifier(dt, n_estimators=n_estimators, max_samples=max_samples, random_state=0)
    # Calculate the cross-validation score
    scores = cross_val_score(bag, X, y, cv=cvKFold)
    return scores.mean()

def adaDTClassifier(X, y, n_estimators, learning_rate, max_depth):
    """
    Takes a numpy data array and target and return the average accuracy
    Achieve the Adaboost Classifier

    :param X: numpy array of shape (n_examples, n_features)
    :param y: numpy array of shape (n_examples)
    :param n_estimators: maximum number of estimators at which boosting is terminated
    :param learning_rate: weight applied to each classifier at each boosting iteration
    :param max_depth: maximum depth of the tree
    :return: the average accuracy of classifier
    """
    # Initialise the AdaBoost classifier using the Decision Tree classifier
    dt = DecisionTreeClassifier(random_state=0, criterion="entropy", max_depth=max_depth)
    ada = AdaBoostClassifier(dt, learning_rate=learning_rate, n_estimators=n_estimators, random_state=0)
    # Calculate the cross-validation score
    scores = cross_val_score(ada, X, y, cv=cvKFold)
    return scores.mean()

def gbClassifier(X, y, n_estimators, learning_rate):
    """
    Takes a numpy data array and target and return the average accuracy
    Achieve the Gradient Boosting Classifier

    :param X: numpy array of shape (n_examples, n_features)
    :param y: numpy array of shape (n_examples)
    :param n_estimators: number of boosting stages to perform
    :param learning_rate: weight applied to each classifier at each boosting iteration
    :return: the average accuracy of classifier
    """
    # Initialise the Gradient Boosting classifier
    gb = GradientBoostingClassifier(learning_rate=learning_rate, n_estimators=n_estimators, random_state=0)
    # Calculate the cross-validation score
    scores = cross_val_score(gb, X, y, cv=cvKFold)
    return scores.mean()
```

Part 1 Results

In [10]:
```
# 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

# Print results for each classifier in part 1 to 4 decimal places here:
print("LogR average cross-validation accuracy: %.4f" % logregClassifier(X_data, y_data))
print("NB average cross-validation accuracy: %.4f" % nbClassifier(X_data, y_data))
print("DT average cross-validation accuracy: %.4f" % dtClassifier(X_data, y_data))
print("Bagging average cross-validation accuracy: %.4f" % bagDTClassifier(X_data, y_data, bag_n_estimators, bag_max_samples, bag_max_depth))
print("AdaBoost average cross-validation accuracy: %.4f" % adaDTClassifier(X_data, y_data, ada_n_estimators, ada_learning_rate, ada_bag_max_depth))
print("GB average cross-validation accuracy: %.4f" % gbClassifier(X_data, y_data, gb_n_estimators, gb_learning_rate))
```

```
LogR average cross-validation accuracy: 0.9642
NB average cross-validation accuracy: 0.9585
DT average cross-validation accuracy: 0.9285
Bagging average cross-validation accuracy: 0.9571
AdaBoost average cross-validation accuracy: 0.9599
GB average cross-validation accuracy: 0.9613
```

Part 2: Cross-validation with parameter tuning

In [11]:
```
# Split data into training and test subsets
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, stratify=y_data, random_state=0)
```

In [12]:
```
# KNN
k = [1, 3, 5, 7, 9]
p = [1, 2]

def bestKNNClassifier(X, y):
    """
    Takes a numpy data array and target and return the average accuracy
    Achieve the Grid Search KNN Classifier

    :param X: numpy array of shape (n_examples, n_features)
    :param y: numpy array of shape (n_examples)
    :return: the best parameter, cross-validation, test accuracy of the best model
    """
    # Set the parameter grid for grid search
    param_grid = {'n_neighbors': k, 'p': p}
    # Initialise and train the grid search classifier for KNN
    knn = GridSearchCV(KNeighborsClassifier(), param_grid, cv=cvKFold, return_train_score=True)
    knn.fit(X, y)
    # Obtain the parameters and accuracy for the best model
    best_k = knn.best_params_['n_neighbors']
    best_p = knn.best_params_['p']
    cv_accuracy = knn.best_score_
    test_accuracy = knn.score(X_test, y_test)
    return best_k, best_p, cv_accuracy, test_accuracy
```

In [13]:
```
# 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]

def bestSVMClassifier(X, y):
    """
    Takes a numpy data array and target and return the average accuracy
    Achieve the Grid Search SVM Classifier

    :param X: numpy array of shape (n_examples, n_features)
    :param y: numpy array of shape (n_examples)
    :return: the best parameter, cross-validation, test accuracy of the best model
    """
    # Set the parameter grid for grid search
    param_grid = {'C': C, 'gamma': gamma}
    # Initialise and train the grid search classifier for SVM
    svm = GridSearchCV(SVC(kernel='rbf'), param_grid, cv=cvKFold, return_train_score=True)
    svm.fit(X, y)
    # Obtain the parameters and accuracy for the best model
    best_C = svm.best_params_['C']
    best_gamma = svm.best_params_['gamma']
    cv_accuracy = svm.best_score_
    test_accuracy = svm.score(X_test, y_test)
    return best_C, best_gamma, cv_accuracy, test_accuracy
```

In [14]:
```
# Random Forest
# You should use RandomForestClassifier from sklearn.ensemble with information gain and max_features set to 'sqrt'.
n_estimators = [10, 30, 60, 100, 150]
max_leaf_nodes = [6, 12, 18]

def bestRFClassifier(X, y):
    """
    Takes a numpy data array and target and return the average accuracy
    Achieve the Grid Search Random Forest Classifier

    :param X: numpy array of shape (n_examples, n_features)
    :param y: numpy array of shape (n_examples)
    :return: the best parameter, cross-validation, test accuracy, F1-score of the best model
    """
    # Set the parameter grid for grid search
    param_grid = {'n_estimators': n_estimators, 'max_leaf_nodes': max_leaf_nodes}
    # Initialise and train the grid search classifier for RandomForest
    rf = GridSearchCV(RandomForestClassifier(criterion="entropy", max_features="sqrt", random_state=0), param_grid, cv=cvKFold, return_train_score=True)
    rf.fit(X, y)
    # Predict the results of the test data
    _pred = rf.predict(X_test)
    # Obtain the parameters and accuracy for the best model
    best_n = rf.best_params_['n_estimators']
    best_leaf = rf.best_params_['max_leaf_nodes']
    cv_accuracy = rf.best_score_
    test_accuracy = rf.score(X_test, y_test)
    # Calculate the F1-score by predict_label and target_label
    f1_macro = f1_score(y_test, _pred, average='macro')
    f1_weight = f1_score(y_test, _pred, average='weighted')
    return best_n, best_leaf, cv_accuracy, test_accuracy, f1_macro, f1_weight
```

Part 2: Results

In [15]:
```
# Perform Grid Search with 10-fold stratified cross-validation (GridSearchCV in sklearn).
# The stratified folds from cvKFold should be provided to GridSearchCV

# This should include using train_test_split from sklearn.model_selection with stratification and random_state=0
# 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.
print('KNN best k: {}'.format(bestKNNClassifier(X_train, y_train)[0]))
print('KNN best p: {}'.format(bestKNNClassifier(X_train, y_train)[1]))
print('KNN cross-validation accuracy {:.4f}'.format(bestKNNClassifier(X_train, y_train)[2]))
print('KNN test set accuracy: {:.4f}'.format(bestKNNClassifier(X_train, y_train)[3]))

print()

print('SVM best C: {:.4f}'.format(bestSVMClassifier(X_train, y_train)[0]))
print('SVM best gamma: {:.4f}'.format(bestSVMClassifier(X_train, y_train)[1]))
print('SVM cross-validation accuracy {:.4f}'.format(bestSVMClassifier(X_train, y_train)[2]))
print('SVM test set accuracy: {:.4f}'.format(bestSVMClassifier(X_train, y_train)[3]))

print()

print('RF best n_estimators: {}'.format(bestRFClassifier(X_train, y_train)[0]))
print('RF best max_leaf_nodes: {}'.format(bestRFClassifier(X_train, y_train)[1]))
print('RF cross-validation accuracy: {:.4f}'.format(bestRFClassifier(X_train, y_train)[2]))
print('RF test set accuracy: {:.4f}'.format(bestRFClassifier(X_train, y_train)[3]))
print('RF test set macro average F1: {:.4f}'.format(bestRFClassifier(X_train, y_train)[4]))
print('RF test set weighted average F1: {:.4f}'.format(bestRFClassifier(X_train, y_train)[5]))
```

```
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
```