COMP5318 Assignment 1: Classification

Group number: A1 Group 112, SID1: 520074642 , SID2: 520076163

In [1]:

```
# Import all libraries
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
import numpy as np

from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import  cross_val_score
from sklearn.neighbors import KNeighborsClassifier
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.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```

In [2]:

```
# Load Breast Cancer Wisconsin Dataset
# Make sure the file is in the same root directory as 'breast-cancer-wisconsin.csv'.
db = pd.read_csv('breast-cancer-wisconsin.csv')
```

In [3]:

```
#Because SimpleImputer requires the missing and replacement values to be in the same
#But '? is a 'string', and the average of each column is a 'float'
#So need to replace '?' to np.nan at first
db.replace('?', np.nan, inplace=True)

#Changing the class values
db['class'].replace("class1",0,inplace=True)
db['class'].replace("class2",1,inplace=True)

#Using SimpleImputer to replace missing values
imp = SimpleImputer(missing_values=np.nan,strategy='mean')
db = imp.fit_transform(db)

#Using MinMaxScaler to Normalise the data between (0,1)
scaler = MinMaxScaler(feature_range=(0, 1))
db = scaler.fit_transform(db)

#Setting X as features, y as classes
df = pd.DataFrame(db) #Convert db into dataframe form to facilitate data processing
X = df.drop(df.columns[-1], axis=1).values #Delete the last column to get the data s
y = df[df.columns[-1]].values #Get the last column as a target set
```

In [4]:

```
# Print first ten rows of pre-processed dataset to 4 decimal places
# Iterate through the first 10 items of the array.
# Use the Map and List functions to convert the array to a list.
# And then use the Join function to join the data and output.
for i in range(0,10):
    print(','.join(list(map('{:.4f}'.format,db[i]))))
```

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

Part 1: Cross validation without parameter tuning

In [5]:

```
## Setting the 10 fold stratified cross-validation
# The stratified folds from cvKFold should be provided to the classifiers
cvKFold=StratifiedKFold(n_splits=10, shuffle=True, random_state=0)
```

In [6]:

```
# K-Nearest Neighbour
def kNNClassifier(X, y, k):
    # Create the KNN classifier,and import the parameter k as the number of neighbor
    knn = KNeighborsClassifier(n_neighbors=k)
    # Cross-validation is performed using the cross_val_score function.
    # Set the classifier parameters to KNN.
    # Set the dataset to the processed dataset X and the target set y.
    # Set the number of folds to cvKFold as required.
    scores = cross_val_score(knn, X, y, cv=cvKFold)
    return scores.mean()
```

In [7]:

```
# Logistic Regression
def logregClassifier(X, y):
    # Create the Logistic regression classifier
    logreg = LogisticRegression()
    # Cross-validation is performed using the cross_val_score function.
    # Set the classifier parameters to Logistic Regression.
    # Set the dataset to the processed dataset X and the target set y.
    # Set the number of folds to cvKFold as required.
    scores = cross_val_score(logreg, X, y, cv=cvKFold)
    return scores.mean()
```

In [8]:

```
#Naïve Bayes
def nbClassifier(X, y):
    # Create the Naïve Bayes classifier
    nb = GaussianNB()
    # Cross-validation is performed using the cross_val_score function.
    # Set the classifier parameters to Naïve Bayes.
    # Set the dataset to the processed dataset X and the target set y.
    # Set the number of folds to cvKFold as required.
    scores = cross_val_score(nb, X, y, cv=cvKFold)
    return scores.mean()
```

In [9]:

```
# Decision Tree
def dtClassifier(X, y):
    # Create the Decision Tree classifier
    dt = DecisionTreeClassifier()
    # Cross-validation is performed using the cross_val_score function.
    # Set the classifier parameters to Decision Tree.
    # Set the dataset to the processed dataset X and the target set y.
    # Set the number of folds to cvKFold as required.
    scores = cross_val_score(dt, X, y, cv=cvKFold)
    return scores.mean()
```

In [10]:

```
# Ensembles: Bagging, Ada Boost and Gradient Boosting
def bagDTClassifier(X, y, n_estimators, max_samples, max_depth):
    # Create the Bagging classifier,and import parameters as required.
    bag_clf = BaggingClassifier(
        DecisionTreeClassifier(max_depth=max_depth), n_estimators=n_estimators, max_
    # Cross-validation is performed using the cross_val_score function.
    # Set the classifier parameters to Bagging.
    # Set the dataset to the processed dataset X and the target set y.
    # Set the number of folds to cvKFold as required.
    scores = cross_val_score(bag_clf, X, y, cv=cvKFold)
    return scores.mean()

def adaDTClassifier(X, y, n_estimators, learning_rate, max_depth):
    # Create the Ada Boost classifier,and import parameters as required.
    ada_clf = AdaBoostClassifier(
        DecisionTreeClassifier(max_depth=max_depth), n_estimators=n_estimators, lear
    # Cross-validation is performed using the cross_val_score function.
    # Set the classifier parameters to Ada Boost.
    # Set the dataset to the processed dataset X and the target set y.
    # Set the number of folds to cvKFold as required.
    scores = cross_val_score(ada_clf, X, y, cv=cvKFold)
    return scores.mean()

def gbClassifier(X, y, n_estimators, learning_rate):
    # Create the Gradient Boosting classifier,and import parameters as required.
    gb_clf = GradientBoostingClassifier(n_estimators=n_estimators, learning_rate=lea
    # Cross-validation is performed using the cross_val_score function.
    # Set the classifier parameters to Gradient Boosting.
    # Set the dataset to the processed dataset X and the target set y.
    # Set the number of folds to cvKFold as required.
    scores = cross_val_score(gb_clf, X, y, cv=cvKFold)
    return scores.mean()
```

Part 1 Results

In [11]:

```
# Parameters for Part 1:
#KNN
k=3

#Bagging
bag_n_estimators = 50
bag_max_samples = 100
bag_max_depth = 5

#AdaBoost
ada_n_estimators = 50
ada_learning_rate = 0.5
ada_bag_max_depth = 5

#GB
gb_n_estimators = 50
gb_learning_rate = 0.5

# Print results for each classifier in part 1 to 4 decimal places here:
print("kNN average cross-validation accuracy: {:.4f}".format(kNNClassifier(X, y, k))
print("LR average cross-validation accuracy: {:.4f}".format(logregClassifier(X, y)))
print("NB average cross-validation accuracy: {:.4f}".format(nbClassifier(X, y)))
print("DT average cross-validation accuracy: {:.4f}".format(dtClassifier(X, y)))
print("Bagging average cross-validation accuracy: {:.4f}".format(bagDTClassifier(X,
print("AdaBoost average cross-validation accuracy: {:.4f}".format(adaDTClassifier(X,
print("GB average cross-validation accuracy: {:.4f}".format(gbClassifier(X, y, gb_n_
```

```
kNN average cross-validation accuracy: 0.9642
LR average cross-validation accuracy: 0.9642
NB average cross-validation accuracy: 0.9585
DT average cross-validation accuracy: 0.9356
Bagging average cross-validation accuracy: 0.9642
AdaBoost average cross-validation accuracy: 0.9585
GB average cross-validation accuracy: 0.9614
```

Part 2: Cross validation with parameter tuning

In [12]:

```
# Linear SVM
# You should use SVC from sklearn.svm
C = {0.001, 0.01, 0.1, 1, 10, 100}
gamma = {0.001, 0.01, 0.1, 1, 10, 100}

def bestLinClassifier(X,y):
    # Iterate through the parameters C and gamma, converting them to the form of a l
    C_list=[]
    for value in C:
        C_list.append(value)
    gamma_list=[]
    for value in gamma:
        gamma_list.append(value)

    # Wrap the parameters C and gamma into a new dictionary.
    para_grid = {'C':C_list,'gamma':gamma_list}

    # Use the spilt function to get the training set and test set in cvKFold.
    for train_index, test_index in cvKFold.split(X, y):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

    # Create the SVM classifier, and set the kernel method to linear.
    lin_svm = SVC(kernel="linear")
    # Set estimator for grid search.
    grid_search = GridSearchCV(lin_svm, para_grid, cv=cvKFold, return_train_score=Tr
    # Import the training set for training.
    grid_search.fit(X_train, y_train)

    # Get the desired results.
    best_C = grid_search.best_params_['C']
    best_gamma = grid_search.best_params_['gamma']
    best_val_score = grid_search.best_score_
    best_set_score = grid_search.score(X_test, y_test)
    return best_C, best_gamma, best_val_score, best_set_score #(appropriate values s
```

In [13]:

```
# Random Forest
# You should use RandomForestClassifier from sklearn.ensemble with information gain
n_estimators = {10, 20, 30, 50, 100}
max_leaf_nodes = {4, 10, 16, 20, 30}

def bestRFClassifier(X,y):
    # Iterate through the parameters n_estimators and max_leaf_nodes, converting the
    n_estimators_list=[]
    for value in n_estimators:
        n_estimators_list.append(value)
    max_leaf_nodes_list=[]
    for value in max_leaf_nodes:
        max_leaf_nodes_list.append(value)

    # Wrap the parameters n_estimators and max_leaf_nodes into a new dictionary.
    para_grid = {'n_estimators':n_estimators_list,'max_leaf_nodes':max_leaf_nodes_li

    # Use the spilt function to get the training set and test set in cvKFold.
    for train_index, test_index in cvKFold.split(X, y):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

    # Create the Random Forest classifier, and set the criterion as entropy and max_
    rnd_clf = RandomForestClassifier(criterion='entropy',max_features='sqrt')
    # Set estimator for grid search.
    grid_search = GridSearchCV(rnd_clf, para_grid, cv=cvKFold, return_train_score=Tr
    # Import the training set for training.
    grid_search.fit(X_train, y_train)

    # Get the desired results.
    best_n_estimators = grid_search.best_params_['n_estimators']
    best_max_leaf_nodes = grid_search.best_params_['max_leaf_nodes']
    best_val_score = grid_search.best_score_
    best_set_score = grid_search.score(X_test, y_test)
    return best_n_estimators, best_max_leaf_nodes, best_val_score, best_set_score#(a
```

Part 2 Results

In [14]:

```
# Perform Grid Search with 10-fold Stratified Cross Validation (GridSearchCV in skle
# The stratified folds from cvKFold should be provided to GridSearchV

# This should include using train_test_split from sklearn.model_selection with strat
# Print results for each classifier here. All results should be printed to 4 decimal
# "n_estimators" and "max_leaf_nodes" which should be printed as integers.

best_C, best_gamma, best_val_score, best_set_score = bestLinClassifier(X,y)
print("SVM best C: {:.4f}".format(best_C))
print("SVM best gamma: {:.4f}".format(best_gamma))
print("SVM cross-validation accuracy: {:.4f}".format(best_val_score))
print("SVM test set accuracy: {:.4f}".format(best_set_score))

best_n_estimators, best_max_leaf_nodes, best_val_score, best_set_score = bestRFClass
print("RF best n_estimators: {:.4f}".format(best_n_estimators))
print("RF best max_leaf_nodes: {:.4f}".format(best_max_leaf_nodes))
print("RF cross-validation accuracy: {:.4f}".format(best_val_score))
print("RF test set accuracy: {:.4f}".format(best_set_score))
```

```
SVM best C: 1.0000
SVM best gamma: 0.1000
SVM cross-validation accuracy: 0.9683
SVM test set accuracy: 0.9710
RF best n_estimators: 20.0000
RF best max_leaf_nodes: 16.0000
RF cross-validation accuracy: 0.9714
RF test set accuracy: 0.9565
```