COMP5318 Assignment 1: Classification

Group number: A1part2 6 , SID1: 520080414

In [35]:
```
import data as data
# Import all libraries
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier, GradientBoos
from sklearn.model_selection import StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from sklearn.metrics import f1_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.datasets import data
from pandas.core import frame
from sklearn.svm import SVC
from sklearn import datasets
from sklearn import preprocessing
from sklearn.datasets import load_iris
import numpy as np
import pandas as pd
```

In [36]:
```
# Load dataset
df = pd.read_csv('breast-cancer-wisconsin.csv')
```

In [37]:
```
# Pre-process dataset

df = df.replace('?', np.nan).replace('class1', 0).replace('class2', 1)
feature = df.iloc[:, 0:-1]
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

features = imputer.fit_transform(feature)
scaler = MinMaxScaler()

X = scaler.fit_transform(features)

classes = df.iloc[:, -1].tolist()
labels = np.unique(classes)
lEnc = LabelEncoder()
lEnc.fit(labels)
label_encoder = lEnc.transform(classes)
numClass = len(labels)

y = label_encoder.astype(np.float64)
```

In [38]:
```
# Print first ten rows of pre-processed dataset to 4 decimal places as per assig
# A function is provided to assist
```

```
dataframe = df.replace('?', np.nan).replace('class1', 0).replace('class2', 1)
simputer = SimpleImputer(missing_values=np.nan, strategy='mean')
minmaxscaler = preprocessing.MinMaxScaler()
dataset = imputer.fit_transform(dataframe)
dataset = minmaxscaler.fit_transform(dataset)


def print_data(X, 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][0:-1:1]:
            print("{:.4f}".format(feature), end=",")
        print(int(X[example_num][-1]))


print_data(dataset)
```

```
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 [39]:
```
## 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 [40]:
```
# Logistic Regression
def logregClassifier(X, y):
    logreg = LogisticRegression(random_state=0)
    scores = cross_val_score(logreg, np.asarray(X, dtype='float64'), y, cv=cvKFo
    return scores.mean()
```

In [41]:
```
#Naïve Bayes
def nbClassifier(X, y):
    nb = GaussianNB()
    scores = cross_val_score(nb, np.asarray(X, dtype='float64'), y, cv=cvKFold)
    return scores.mean()
```

In [42]:
```
# Decision Tree
def dtClassifier(X, y):
    classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
    scores = cross_val_score(classifier, np.asarray(X, dtype='float64'), y, cv=c
    return scores.mean()
```

In [43]:
```
# Ensembles: Bagging, Ada Boost and Gradient Boosting
def bagDTClassifier(X, y, n_estimators, max_samples, max_depth):
    classifier = BaggingClassifier(DecisionTreeClassifier(max_depth=max_depth, c
                                   , n_estimators=n_estimators, max_samples=max_
    scores = cross_val_score(classifier, np.asarray(X, dtype='float64'), y, cv=c
    return scores.mean()


def adaDTClassifier(X, y, n_estimators, learning_rate, max_depth):
    classifier = AdaBoostClassifier(DecisionTreeClassifier(max_depth=max_depth,
                                    , n_estimators=n_estimators, learning_rate=l
    scores = cross_val_score(classifier, np.asarray(X, dtype='float64'), y, cv=c
    return scores.mean()


def gbClassifier(X, y, n_estimators, learning_rate):
    classifier = GradientBoostingClassifier(n_estimators=n_estimators, learning_
    scores = cross_val_score(classifier, np.asarray(X, dtype='float64'), y, cv=c
    return scores.mean()
```

Part 1 Results

In [44]:
```
from sklearn.preprocessing import LabelEncoder

# Parameters for Part 1:
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_sta

#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}".format(logregClassifier(X
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, y, bag_n_estimators, bag_max_samples, bag_max_depth)))
print("AdaBoost average cross-validation accuracy: {:.4f}".format(
    adaDTClassifier(X, y, ada_n_estimators, ada_learning_rate, ada_bag_max_depth
print("GB average cross-validation accuracy: {:.4f}".format(gbClassifier(X, y, g
```

```
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: 0.9585
GB average cross-validation accuracy: 0.9613
```

Part 2: Cross-validation with parameter tuning

In [45]:
```
# KNN
k = [1, 3, 5, 7, 9]
p = [1, 2]


def bestKNNClassifier(X, y):
    param_grid = {'n_neighbors': k, 'p': p}
    classifier = KNeighborsClassifier()
    grid_search= GridSearchCV(classifier, param_grid, cv=cvKFold, return_train_s
    grid_search.fit(X, y)
    return grid_search
```

In [46]:
```
# 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):
    param_grid = {'C': C, 'gamma': gamma}
    grid_search = GridSearchCV(SVC(kernel='rbf', random_state=0), param_grid, cv
    grid_search.fit(X, y)
    return grid_search
```

In [47]:
```
# Random Forest
# You should use RandomForestClassifier from sklearn.ensemble with information g
n_estimators = [10, 30, 60, 100, 150]
max_leaf_nodes = [6, 12, 18]


def bestRFClassifier(X, y):
    param_grid = {'n_estimators': n_estimators, 'max_leaf_nodes': max_leaf_nodes
    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random
    grid_search = GridSearchCV(RandomForestClassifier(random_state=0, criterion=
                               return_train_score=True)
    grid_search.fit(X_train, y_train)
    return grid_search
```

Part 2: Results

In [48]:
```
# Perform Grid Search with 10-fold stratified cross-validation (GridSearchCV in
# The stratified folds from cvKFold should be provided to GridSearchV

# This should include using train_test_split from sklearn.model_selection with s
# Print results for each classifier here. All results should be printed to 4 dec
# "k", "p", n_estimators" and "max_leaf_nodes" which should be printed as intege
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_sta

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

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

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

```
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
```

In [ ]: