COMP5318 Assignment 1: Classification

Group 10: 520302040, 520356283

Importing libraries

In [1]:
```
import pandas as pd
import seaborn as sea
import matplotlib.pyplot as plt
import numpy as np
import warnings

warnings.filterwarnings("ignore")
```

Calling packages from sklearn

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.linear_model import LogisticRegression,LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.impute import SimpleImputer
from sklearn.metrics import f1_score, make_scorer
from sklearn.model_selection import cross_val_score
```

Importing the Dataset

In [3]:
```
data = pd.read_csv("breast-cancer-wisconsin.csv")
data.head()
```

Out[3]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	cl
0	5	1	1	1	2	1	3	1	1	cla
1	5	4	4	5	7	10	3	2	1	cla
2	3	1	1	1	2	2	3	1	1	cla
3	6	8	8	1	3	4	3	7	1	cla
4	4	1	1	3	2	1	3	1	1	cla

Exploring the Dataset

In [4]:
```
data.describe()
```

Out[4]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bland Chromatin	Normal Nucleoli	Mitose
count	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000
mean	4.417740	3.134478	3.207439	2.806867	3.216023	3.437768	2.866953	1.589413
std	2.815741	3.051459	2.971913	2.855379	2.214300	2.438364	3.053634	1.715078
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	2.000000	1.000000	1.000000	1.000000	2.000000	2.000000	1.000000	1.000000
50%	4.000000	1.000000	1.000000	1.000000	2.000000	3.000000	1.000000	1.000000
75%	6.000000	5.000000	5.000000	4.000000	4.000000	5.000000	4.000000	1.000000
max	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 10 columns):
 #   Column                       Non-Null Count  Dtype 
---  ------                       --------------  ----- 
 0   Clump Thickness              699 non-null    int64 
 1   Uniformity of Cell Size      699 non-null    int64 
 2   Uniformity of Cell Shape     699 non-null    int64 
 3   Marginal Adhesion            699 non-null    int64 
 4    Single Epithelial Cell Size 699 non-null    int64 
 5   Bare Nuclei                  699 non-null    object
 6   Bland Chromatin              699 non-null    int64 
 7   Normal Nucleoli              699 non-null    int64 
 8   Mitoses                      699 non-null    int64 
 9   class                        699 non-null    object
dtypes: int64(8), object(2)
memory usage: 54.7+ KB
```

Notes:

1. convert data types - bare nuclei to int.

Data Preprocessing

In [6]:
```
# Checking for NULL values
Null_values=(data=='?').sum()
print(Null_values)
```

```
Clump Thickness                0
Uniformity of Cell Size        0
Uniformity of Cell Shape       0
Marginal Adhesion              0
 Single Epithelial Cell Size   0
Bare Nuclei                   16
Bland Chromatin                0
Normal Nucleoli                0
Mitoses                        0
class                          0
dtype: int64
```

Notes:

1. Bare Nuclei column has 16 missing values.

In [7]:
```
#Assigning the data to a new variable "data_2" without the class column
data_2 = data.drop('class', axis=1)

#Convert all '?' to NaN to allow simple imputer to work
data_2 = data_2.replace('?', np.NaN)
display(data_2)
```

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
0	5	1	1	1	2	1	3	1	1
1	5	4	4	5	7	10	3	2	1
2	3	1	1	1	2	2	3	1	1
3	6	8	8	1	3	4	3	7	1
4	4	1	1	3	2	1	3	1	1
...	...	...	...	...	...	...	...	...	...
694	3	1	1	1	3	2	1	1	1
695	2	1	1	1	2	1	1	1	1
696	5	10	10	3	7	3	8	10	2
697	4	8	6	4	3	4	10	6	1
698	4	8	8	5	4	5	10	4	1

699 rows × 9 columns

In [8]:
```
#Calling in simple imputer to impute missing values with mean
imputer = SimpleImputer(missing_values = np.NaN, strategy ='mean')
imputed_data = pd.DataFrame(imputer.fit_transform(data_2))
imputed_data.columns=data_2.columns

display(imputed_data)

#Checking for null values in the columns
imputed_data.isnull().sum()
```

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
0	5.0	1.0	1.0	1.0	2.0	1.0	3.0	1.0	1.0
1	5.0	4.0	4.0	5.0	7.0	10.0	3.0	2.0	1.0
2	3.0	1.0	1.0	1.0	2.0	2.0	3.0	1.0	1.0
3	6.0	8.0	8.0	1.0	3.0	4.0	3.0	7.0	1.0
4	4.0	1.0	1.0	3.0	2.0	1.0	3.0	1.0	1.0
...	...	...	...	...	...	...	...	...	...
694	3.0	1.0	1.0	1.0	3.0	2.0	1.0	1.0	1.0
695	2.0	1.0	1.0	1.0	2.0	1.0	1.0	1.0	1.0
696	5.0	10.0	10.0	3.0	7.0	3.0	8.0	10.0	2.0
697	4.0	8.0	6.0	4.0	3.0	4.0	10.0	6.0	1.0
698	4.0	8.0	8.0	5.0	4.0	5.0	10.0	4.0	1.0

699 rows × 9 columns

Out[8]:
```
Clump Thickness                0
Uniformity of Cell Size        0
Uniformity of Cell Shape       0
Marginal Adhesion              0
 Single Epithelial Cell Size   0
Bare Nuclei                    0
Bland Chromatin                0
Normal Nucleoli                0
Mitoses                        0
dtype: int64
```

In [9]:
```
#Normalisation of the data
processed_data = imputed_data
scaler = MinMaxScaler()
processed_data = pd.DataFrame(scaler.fit_transform(processed_data))
processed_data.columns = data_2.columns
processed_data
```

Out[9]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
0	0.444444	0.000000	0.000000	0.000000	0.111111	0.000000	0.222222	0.000000	0.000000
1	0.444444	0.333333	0.333333	0.444444	0.666667	1.000000	0.222222	0.111111	0.000000
2	0.222222	0.000000	0.000000	0.000000	0.111111	0.111111	0.222222	0.000000	0.000000
3	0.555556	0.777778	0.777778	0.000000	0.222222	0.333333	0.222222	0.666667	0.000000
4	0.333333	0.000000	0.000000	0.222222	0.111111	0.000000	0.222222	0.000000	0.000000
...	...	...	...	...	...	...	...	...	...
694	0.222222	0.000000	0.000000	0.000000	0.222222	0.111111	0.000000	0.000000	0.000000
695	0.111111	0.000000	0.000000	0.000000	0.111111	0.000000	0.000000	0.000000	0.000000
696	0.444444	1.000000	1.000000	0.222222	0.666667	0.222222	0.777778	1.000000	0.111111
697	0.333333	0.777778	0.555556	0.333333	0.222222	0.333333	1.000000	0.555556	0.000000
698	0.333333	0.777778	0.777778	0.444444	0.333333	0.444444	1.000000	0.333333	0.000000

699 rows × 9 columns

In [10]:
```
#Changing values of classes Class 1 to 0 and Class 2 to 1
data['class'].replace({'class1':0,'class2':1},inplace=True)
data[['class']]
```

Out[10]:

	class
0	0
1	0
2	0
3	0
4	0
...	...
694	0
695	0
696	1
697	1
698	1

699 rows × 1 columns

In [11]:
```
#Checking the shape of the processed data
processed_data.shape
```

Out[11]: (699, 9)

In [12]:
```
#Printing the first 10 rows of the pre-processed dataset with 4 decimal places.
def print_data(X, y, n_rows=10):
    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])

a = np.array(processed_data.values)
b = np.array(data['class'])
print_data(a, b, n_rows = 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
```

Cross Validation without parameter tuning

In [13]:
```
#Setting the 10 fold stratified cross validation
Cross_Validation_Fold = StratifiedKFold(n_splits=10, shuffle=True, random_state=0)
```

K-Nearest Neighbour

In [14]:
```
def KNN_Model(X, y, k):
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv=Cross_Validation_Fold)
    return scores.mean()
```

Logistic Regression

In [15]:
```
def Logistic_Regression_Model(X, y):
    lor = LogisticRegression(random_state=0)
    scores = cross_val_score(lor, X, y, cv=Cross_Validation_Fold)
    return scores.mean()
```

Naive Bayes

In [16]:
```
def Naive_Bayes_Classifier(X, y):
    nbc = GaussianNB()
    scores = cross_val_score(nbc, X, y, cv=Cross_Validation_Fold)
    return scores.mean()
```

Decision Tree

In [17]:
```
def Decision_Tree_Classifier(X, y):
    dtc = DecisionTreeClassifier(random_state=0,criterion="entropy")
    scores = cross_val_score(dtc, X, y, cv=Cross_Validation_Fold)
    return scores.mean()
```

Bagging

In [18]:
```
def Bagging_Classifier(X, y, n_estimators, max_samples, max_depth):
    bagger = BaggingClassifier(base_estimator = DecisionTreeClassifier(criterion="entr
    scores = cross_val_score(bagger, X, y, cv=Cross_Validation_Fold)
    return scores.mean()
```

Ada Boost

In [19]:
```
def Ada_Boost_Classifier(X, y, n_estimators, learning_rate, max_depth):
    ada_boost = AdaBoostClassifier(base_estimator = DecisionTreeClassifier(criterion='
    scores = cross_val_score(ada_boost, X, y, cv=Cross_Validation_Fold)
    return scores.mean()
```

Gradient Boost

In [20]:
```
def Gradient_Boost_Classifier(X, y, n_estimators, learning_rate):
    gradient_boost = GradientBoostingClassifier(n_estimators=n_estimators, learning_ra
    scores = cross_val_score(gradient_boost, X, y, cv=Cross_Validation_Fold)
    return scores.mean()
```

Results for Cross Validation without Parameter Tuning

In [21]:
```
#Split Data
X = processed_data
y = data['class']

#Defining the KNN k value
k = 3

#Defining Bagging parameters
bag_n_estimators = 60
bag_max_samples = 100
bag_max_depth = 6

#Defining Ada_Boost parameters
ada_n_estimators = 60
ada_learning_rate = 0.5
ada_bag_max_depth = 6

#Defining Gradient_Boost parameters
gb_n_estimators = 60
gb_learning_rate = 0.5
```

In [22]:
```
print("kNN average cross-validation accuracy:{:.4f}".format(KNN_Model(X,y,k)))
print("LogR average cross-validation accuracy:{:.4f}".format(Logistic_Regression_Model
print("NB average cross-validation accuracy:{:.4f}".format(Naive_Bayes_Classifier(X,y)
print("DT average cross-validation accuracy:{:.4f}".format(Decision_Tree_Classifier(X,
print("Bagging average cross-validation accuracy:{:.4f}".format(Bagging_Classifier(X,y
```

```
print("AdaBoost average cross-validation accuracy:{:.4f}".format(Ada_Boost_Classifier(
print("GB average cross-validation accuracy:{:.4f}".format(Gradient_Boost_Classifier()
```

```
kNN average cross-validation accuracy:0.9642
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:0.9570
GB average cross-validation accuracy:0.9613
```

Cross Validation with Parameter Tuning

K-Nearest Neighbour

In [23]:
```
k = [1, 3, 5, 7, 9]
p = [1, 2]

def bestKNNClassifier(X, y):
    param_grid = {'n_neighbors': [1, 3, 5, 7, 9], 'p': [1, 2]}
    knn = KNeighborsClassifier()
    grid_search = GridSearchCV(knn, param_grid, cv=Cross_Validation_Fold, return_trair
    grid_search.fit(X, y)
    KNN_Best_Params = grid_search.best_params_
    KNN_Best_Score = grid_search.best_score_
    KNN_Test_Accuracy = grid_search.score(X_test,y_test)
    return KNN_Best_Params, KNN_Best_Score, KNN_Test_Accuracy
```

RBF Support Vector Machine

In [24]:
```
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(random_state=0, kernel='rbf'), param_grid, cv=Cross_\
    grid_search.fit(X,y)
    svm=SVC(C=grid_search.best_params_['C'],gamma=grid_search.best_params_['gamma'],ra
    svm.fit(X_train,y_train)
    y_pred=svm.predict(X_test)
    accuracy=accuracy_score(y_test,y_pred)
    return grid_search.best_params_, grid_search.best_score_, accuracy
```

Random Forest

In [25]:
```
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}
    grid_search=GridSearchCV( RandomForestClassifier(criterion="entropy",max_features=
    grid_search.fit(X,y)
    rf=RandomForestClassifier(criterion="entropy",max_features="sqrt",n_estimators=gri
    rf.fit(X_train,y_train)
    y_pred=rf.predict(X_test)
    accuracy=accuracy_score(y_test,y_pred)
```

```
    f1_scorer = f1_score(y_test, y_pred, average='macro')
    weighted_f1_scorer = f1_score(y_test, y_pred, average='weighted')
    return grid_search.best_params_, grid_search.best_score_, accuracy, f1_scorer, wei
```

Results for Cross Validation with Parameter Tuning

In [26]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=0)
KNN_Best_Params, KNN_Best_Score, KNN_Test_Accuracy = bestKNNClassifier(X_train, y_trai
SVM_Best_Params, SVM_Best_Score, SVM_Test_Accuracy = bestSVMClassifier(X_train, y_trai
RF_Best_Params, RF_Best_Score, RF_Test_Accuracy, RF_F1_score, RF_weighted_F1_score = b

print("KNN best k: ", KNN_Best_Params['n_neighbors'])
print("KNN best p: ", KNN_Best_Params['p'])
print("KNN cross-validation accuracy: {:.4f}".format(KNN_Best_Score))
print("KNN test set accuracy: {:.4f}".format(KNN_Test_Accuracy))

print()

print("SVM best C: {:.4f}".format(SVM_Best_Params['C']))
print("SVM best gamma: {:.4f}".format(SVM_Best_Params['gamma']))
print("SVM cross-validation accuracy: {:.4f}".format(SVM_Best_Score))
print("SVM test set accuracy: {:.4f}".format(SVM_Test_Accuracy))

print()

print("RF best n_estimators: ", RF_Best_Params['n_estimators'])
print("RF best max_leaf_nodes: ", RF_Best_Params['max_leaf_nodes'])
print("RF cross-validation accuracy: {:.4f}".format(RF_Best_Score))
print("RF test set accuracy: {:.4f}".format(RF_Test_Accuracy))
print("RF test set macro average F1: {:.4f}".format(RF_F1_score.mean()))
print("RF test set weighted average F1: {:.4f}".format(RF_weighted_F1_score.mean()))
```

```
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 [ ]:
```

```