COMP5318 Assignment 1: Rice Classification

Group number: Set 1, Group 153

Student 1 SID: 550403876

Student 2 SID: 540453036

Import Libraries

In [1]:
```
# Import all libraries
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, StratifiedKFold, tra
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier, Gradi
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score
from sklearn.svm import SVC

# Ignore future warnings
from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)
```

Load Dataset

In [2]:
```
# Load the rice dataset: rice-final2.csv

data = pd.read_csv('rice-final2.csv')
# data = pd.read_csv('test/test-before.csv')
```

Pre-Process Dataset

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

#Map to numeric values in target class
y = data['class'].map({'class1': 0, 'class2': 1})

# Convert feature columns to numeric, swapping errors with NaN
X = data.drop(columns='class')
for col in X.columns:
    X[col] = pd.to_numeric(data[col], errors='coerce')

# Impute missing values with mean
imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
imp_mean.fit(X)
X = imp_mean.transform(X)
```

```
# Normalize
scaler = MinMaxScaler()
scaler.fit(X)
X = scaler.transform(X)
```

Print first 10 rows

In [4]:
```
# Print first ten rows of pre-processed dataset to 4 decimal places as pe
# 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(X, y)
```

```
0.4628,0.5406,0.5113,0.4803,0.7380,0.4699,0.1196,1
0.4900,0.5547,0.5266,0.5018,0.7319,0.4926,0.8030,1
0.6109,0.6847,0.6707,0.5409,0.8032,0.6253,0.1185,0
0.6466,0.6930,0.6677,0.5961,0.7601,0.6467,0.2669,0
0.6712,0.6233,0.4755,0.8293,0.3721,0.6803,0.4211,1
0.2634,0.2932,0.2414,0.4127,0.5521,0.2752,0.2825,1
0.8175,0.9501,0.9515,0.5925,0.9245,0.8162,0.0000,0
0.3174,0.3588,0.3601,0.3908,0.6921,0.3261,0.8510,1
0.3130,0.3050,0.2150,0.5189,0.3974,0.3159,0.4570,1
0.5120,0.5237,0.4409,0.6235,0.5460,0.5111,0.3155,1
```

Part 1: Cross-validation without parameter tuning

Stratified K-Fold

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
```

Logistic Regression Classifier

In [6]:
```
# Logistic Regression
def logregClassifier(X, y):
    model = LogisticRegression(random_state=0)
    scores = cross_val_score(model, X, y, cv=cvKFold, scoring='accuracy')
    return scores.mean()
```

Naïve Bayes Classifier

In [7]:
```
#Naïve Bayes
def nbClassifier(X, y):
    model = GaussianNB()
    scores = cross_val_score(model, X, y, cv=cvKFold, scoring='accuracy')
    return scores.mean()
```

Decision Tree Classifier

In [8]:
```
# Decision Tree
def dtClassifier(X, y):
    model = DecisionTreeClassifier(criterion='entropy',random_state=0)
    scores = cross_val_score(model, X, y, cv=cvKFold, scoring='accuracy')
    return scores.mean()
```

Bagging, Ada Boost and Gradient Boosting Classifiers

In [9]:
```
# Ensembles: Bagging, Ada Boost and Gradient Boosting
def bagDTClassifier(X, y, n_estimators, max_samples, max_depth):
    model = BaggingClassifier(DecisionTreeClassifier(max_depth=max_depth,
                      n_estimators=n_estimators, max_samples=max_samples,
    scores = cross_val_score(model, X, y, cv=cvKFold, scoring='accuracy')
    return scores.mean()

def adaDTClassifier(X, y, n_estimators, learning_rate, max_depth):
    model = AdaBoostClassifier(DecisionTreeClassifier(max_depth=max_depth
                      n_estimators=n_estimators, learning_rate=learning_r
    scores = cross_val_score(model, X, y, cv=cvKFold, scoring='accuracy')
    return scores.mean()

def gbClassifier(X, y, n_estimators, learning_rate):
    model = GradientBoostingClassifier(n_estimators=n_estimators, learnin
    scores = cross_val_score(model, X, y, cv=cvKFold, scoring='accuracy')
    return scores.mean()
```

Part 1 Results

In [10]:
```
# Parameters for Part 1:

#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("LogR average cross-validation accuracy: {:.4f}".format(logregClass
print("NB average cross-validation accuracy: {:.4f}".format(nbClassifier(
print("DT average cross-validation accuracy: {:.4f}".format(dtClassifier(
print("Bagging average cross-validation accuracy: {:.4f}".format(bagDTCla
print("AdaBoost average cross-validation accuracy: {:.4f}".format(adaDTCl
print("GB average cross-validation accuracy: {:.4f}".format(gbClassifier(
```

```
LogR average cross-validation accuracy: 0.9386
NB average cross-validation accuracy: 0.9264
DT average cross-validation accuracy: 0.9179
Bagging average cross-validation accuracy: 0.9414
AdaBoost average cross-validation accuracy: 0.9407
GB average cross-validation accuracy: 0.9321
```

Part 2: Cross-validation with parameter tuning

In [11]:
```
# split dataset into training and testing dataset 75% training, 25% testi
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
```

KNN Classifier

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

def bestKNNClassifier(X, y):
    param_grid = {'n_neighbors': k, 'p': p}
    knn = KNeighborsClassifier()
    grid_search = GridSearchCV(knn, param_grid, cv=cvKFold, scoring='accu
    grid_search.fit(X_train, y_train)
    best_knn = grid_search.best_estimator_
    y_pred = best_knn.predict(X_test)

    return grid_search.best_params_['n_neighbors'], grid_search.best_para
```

SVM Classifier

In [13]:
```
# SVM
# You should use SVC from sklearn.svm with kernel set to 'rbf'
C = [0.01, 0.1, 1, 5]
gamma = [0.01, 0.1, 1, 10]

def bestSVMClassifier(X, y):
    param_grid = {'C': C, 'gamma': gamma}
    svm = SVC(kernel='rbf', random_state=0)
    grid_search = GridSearchCV(svm, param_grid, cv=cvKFold, scoring='accu
    grid_search.fit(X_train, y_train)
    best_svm = grid_search.best_estimator_
    y_pred = best_svm.predict(X_test)

    return grid_search.best_params_['C'], grid_search.best_params_['gamma
```

RandomForest Classifier

In [14]:
```
# Random Forest
# You should use RandomForestClassifier from sklearn.ensemble with inform
n_estimators = [10, 30, 60, 100]
max_leaf_nodes = [6, 12]
```

```python
def bestRFClassifier(X, y):
    param_grid = {'n_estimators': n_estimators, 'max_leaf_nodes': max_lea
    rf = RandomForestClassifier(criterion='entropy', max_features='sqrt',
    grid_search = GridSearchCV(rf, param_grid, cv=cvKFold, scoring='accur
    grid_search.fit(X_train, y_train)
    best_svm = grid_search.best_estimator_
    y_pred = best_svm.predict(X_test)

    return grid_search.best_params_['n_estimators'], grid_search.best_par
        grid_search.best_score_, accuracy_score(y_test, y_pred), \
        f1_score(y_test, y_pred, average='macro'), \
        f1_score(y_test, y_pred, average='weighted')
```

Part 2: Results

In [15]:

```python
# Perform Grid Search with 10-fold stratified cross-validation (GridSearc
# The stratified folds from cvKFold should be provided to GridSearchV

# This should include using train_test_split from sklearn.model_selection
# Print results for each classifier here. All results should be printed t
# "k", "p", n_estimators" and "max_leaf_nodes" which should be printed as
knn = bestKNNClassifier(X,y)
print("KNN best k: {:.4f}".format(knn[0]))
print("KNN best p: {:.4f}".format(knn[1]))
print("KNN cross-validation accuracy: {:.4f}".format(knn[2]))
print("KNN test set accuracy: {:.4f}".format(knn[3]))

print()
svm = bestSVMClassifier(X,y)
print("SVM best C: {:.4f}".format(svm[0]))
print("SVM best gamma: {:.4f}".format(svm[1]))
print("SVM cross-validation accuracy: {:.4f}".format(svm[2]))
print("SVM test set accuracy: {:.4f}".format(svm[3]))

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

```
KNN best k: 5.0000
KNN best p: 1.0000
KNN cross-validation accuracy: 0.9371
KNN test set accuracy: 0.9257

SVM best C: 5.0000
SVM best gamma: 1.0000
SVM cross-validation accuracy: 0.9457
SVM test set accuracy: 0.9343

RF best n_estimators: 30.0000
RF best max_leaf_nodes: 12.0000
RF cross-validation accuracy: 0.9390
RF test set accuracy: 0.9371
RF test set macro average F1: 0.9355
RF test set weighted average F1: 0.9370
```

Part 3: Reflection

Write one paragraph describing the most important thing that you have learned throughout this assignment.

Student 1:

I focused on data preprocessing and Part 1: "Cross-validation without parameter tuning" in this assignment. I expected the ensemble methods (Bagging, AdaBoost and Gradient boost) to perform sifnificantly better than the individual classifiers (logistic regression, naive bayes, and decision trees). While they did perform better than two of them, they were all beaten by decision trees in terms of accuracy. There can be several reasons for this:

- The dataset is not complex and/or noisy enough for ensemble methods to shine.
- The dataset has very clear boundaries between classes, making it a good fit for decision trees.
- Lack of parameer tuning - the ensemble methods should have been tuned to perform better.

Either way, the key takeaway for me was that there is no one-size-fits-all model. While one can make educated guesses about which models will perform better than others, you should always test them out before making a decision.

Student 2:

I focused on Part-2: "Cross-validation with parameter tuning" and overall review of the codes in this assignment. Throughout the experimentation I saw the trend that increase in cross validation (CV) scores improves testing accuracy which means that CV set is correlated with testing set, which means models which have the highest CV will perform the best on unseen data, so in my part RandomForest was the model with highest CV score of 0.7883 which perfectly makes sense as RandomForest is more robust to noise and overfitting because of it's tree base ensemble. Parameter Tuned RandomForest outperforms Bagging, AdaBoost and Gradient boost because hyper-parameter tuning provides a performance boost to the models, and thus random forest has optimal parameters compared to other models. Had the other ensembles been tuned, their performance might have been better, but this would have required more time and complexity.