a1-COMP5318-2025-group155

April 6, 2025

1 COMP5318 Assignment 1: Rice Classification

Group number: 155

Student 1 SID: 540114883

Student 2 SID: 540282735

[40]:
```python
# Import all libraries
import pandas as pd
import numpy as np

from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import f1_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

[41]:
```python
# Ignore future warnings
from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)
```

[42]:
```python
# Load the rice dataset: rice-final2.csv
data = pd.read_csv('rice-final2.csv')
```

[43]:

```
# Pre-process dataset
data = data.replace('?', np.nan)
for l in data.columns[:-1]:
    data[l] = pd.to_numeric(data[l])

X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

imputer = SimpleImputer(strategy = 'mean')
X = imputer.fit_transform(X)

scaler = MinMaxScaler()
X = scaler.fit_transform(X)

y = np.where(y == 'class1',0,1)
```

[44]:

```
# 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])

```

[45]:

```
print_data(X, y, 10)
```

```
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
```

1.0.1 Part 1: Cross-validation without parameter tuning

[46]:

```python
## 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
```

[47]:

```python
# Logistic Regression
def logregClassifier(X, y):

    model = LogisticRegression(random_state=0, max_iter=1000)
    scores = cross_val_score(model, X, y, cv=cvKFold)

    return scores.mean()
```

[48]:

```python
#Naïve Bayes
def nbClassifier(X, y):

    model = GaussianNB()
    scores = cross_val_score(model, X, y, cv=cvKFold)

    return scores.mean()
```

[49]:

```python
# Decision Tree
def dtClassifier(X, y):

    model = DecisionTreeClassifier(criterion='entropy', random_state=0)
    scores = cross_val_score(model, X, y, cv=cvKFold)

    return scores.mean()
```

[50]:

```python
# Ensembles: Bagging, Ada Boost and Gradient Boosting
def bagDTClassifier(X, y, n_estimators, max_samples, max_depth):

    base = DecisionTreeClassifier(criterion='entropy', max_depth=max_depth,␣
 ↪random_state=0)
    model = BaggingClassifier(estimator=base, n_estimators=n_estimators,␣
 ↪max_samples=max_samples, random_state=0)
    scores = cross_val_score(model, X, y, cv=cvKFold)

    return scores.mean()

def adaDTClassifier(X, y, n_estimators, learning_rate, max_depth):
    base = DecisionTreeClassifier(criterion='entropy', max_depth=max_depth,␣
 ↪random_state=0)
    model = AdaBoostClassifier(estimator=base, n_estimators=n_estimators,␣
 ↪learning_rate=learning_rate, random_state=0)
    scores = cross_val_score(model, X, y, cv=cvKFold)

    return scores.mean()

def gbClassifier(X, y, n_estimators, learning_rate):

    model = GradientBoostingClassifier(n_estimators=n_estimators,␣
 ↪learning_rate=learning_rate, random_state=0)
    scores = cross_val_score(model, X, y, cv=cvKFold)

    return scores.mean()
```

1.0.2 Part 1 Results

[51]:

```
# 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(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, 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,␣
 ↪gb_n_estimators, gb_learning_rate)))
```

```
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: 0.9264
GB average cross-validation accuracy: 0.9321
```

1.0.3 Part 2: Cross-validation with parameter tuning

[52]:

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


def bestKNNClassifier(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,␣
 ↪random_state=0)
```

```
    param_grid = {'n_neighbors': k, 'p':p}
    knn = KNeighborsClassifier()
    grid = GridSearchCV(knn, param_grid=param_grid, cv=cvKFold,␣
 ↪return_train_score=True)
    grid.fit(X_train, y_train)

    test_score = grid.score(X_test, y_test)
    best_val_score = grid.best_score_
    best_params = grid.best_params_

    return best_params['n_neighbors'], best_params['p'], best_val_score,␣
 ↪test_score
```

[53]:

```
# 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):
    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,␣
 ↪random_state=0)

    param_grid = {'C': C, 'gamma': gamma}
    svm = SVC(kernel='rbf', random_state=0)
    grid = GridSearchCV(svm, param_grid=param_grid, cv=cvKFold,␣
 ↪return_train_score=True)
    grid.fit(X_train, y_train)

    test_score = grid.score(X_test, y_test)
    best_val_score = grid.best_score_
    best_params = grid.best_params_

    return best_params['C'], best_params['gamma'], best_val_score, test_score
```

[54]:

```
# Random Forest
# You should use RandomForestClassifier from sklearn.ensemble with information␣
 ↪gain and max_features set to ‘sqrt’.
n_estimators = [10, 30, 60, 100]
max_leaf_nodes = [6, 12]

def bestRFClassifier(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,␣
 ↪random_state=0)

    param_grid = {'n_estimators': n_estimators, 'max_leaf_nodes':␣
 ↪max_leaf_nodes}
```

```
    rf = RandomForestClassifier(criterion='entropy', max_features='sqrt',␣
  ↪random_state=0)
    grid = GridSearchCV(rf, param_grid=param_grid, cv=cvKFold,␣
  ↪return_train_score=True)
    grid.fit(X_train, y_train)


    test_score = grid.score(X_test, y_test)
    best_val_score = grid.best_score_
    best_params = grid.best_params_

    y_pred = grid.predict(X_test)
    macro_f1 = f1_score(y_test, y_pred, average='macro')
    weighted_f1 = f1_score(y_test, y_pred, average='weighted')

    return best_params['n_estimators'], best_params['max_leaf_nodes'],␣
  ↪best_val_score, test_score, macro_f1, weighted_f1
```

1.0.4 Part 2: Results

[55]:

```
# Perform Grid Search with 10-fold stratified cross-validation (GridSearchCV in␣
  ↪sklearn).
# The stratified folds from cvKFold should be provided to GridSearchV

# 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.
knn_res = bestKNNClassifier(X,y)
print("KNN best k:", knn_res[0])
print("KNN best p:", knn_res[1])
print(f"KNN cross-validation accuracy: {knn_res[2]:.4f}")
print(f"KNN test set accuracy: {knn_res[3]:.4f}")

print()

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

print()

rf_res = bestRFClassifier(X, y)
```

```python
print("RF best n_estimators: ", rf_res[0])
print("RF best max_leaf_nodes: ", rf_res[1])
print(f"RF cross-validation accuracy: {rf_res[2]:.4f}")
print(f"RF test set accuracy: {rf_res[3]:.4f}")
print(f"RF test set macro avg F1: {rf_res[4]:.4f}")
print(f"RF test set weighted avg F1: {rf_res[5]:.4f}")
```

```
KNN best k: 5
KNN best p: 1
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
RF best max_leaf_nodes:  12
RF cross-validation accuracy: 0.9390
RF test set accuracy: 0.9371
RF test set macro avg F1: 0.9355
RF test set weighted avg F1: 0.9370
```

1.0.5 Part 3: Reflection

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

Student 1: This assignment gave me hands-on experience with machine learning, teaching me key skills like data preprocessing (handling missing values, scaling), implementing classification algorithms, and evaluating models through cross-validation. I learned how crucial parameter tuning (e.g., grid search) is for improving performance. Overall, it helped me connect theory to practice, strengthening my understanding of the full ML workflow and preparing me for more complex projects.

Student 2: The most important thing learned in this project is how important it is to sort and properly preprocess data before applying a machine learning model. At first, I didn't realize that the datasets were full of object types and not numerical types. So I came across a lot of mistakes. We need to transform data. This may seem like a small step, but I realize that they are very important to start the model.