1 Part 1

Mini-exam question:
Question 1: Write a proper introductory chapter for your report:

- Explain the task and the objectives **0.1p**
- Describe, what kind of data were used? **0.1p** Where did it originate? **0.1p** Give the correct reference. **0.1p** Explain shortly, how the data has been prepared in the original article. **0.2p** Explain, how you deployed the data **0.2p**
- Report shortly the process used in this task (preprocessing **0.2p**, feature derivation **0.2p**, methods **0.2p**, training **0.2p**, evaluation **0.2p**)
- Describe the results **0.2p**

Introduction
This is an exercise for Machine Learning and Pattern Recognition course. The purpose of this task was to learn how to train and evaluate machine learning (ML) models, which classifiy images into classes. The exercise had different tasks: to import the data, to derive essential features, to analyse the gathered data, to build and select best ML models using hyperparameter selection, and finally to evaluate the performance of the selected models.

The exercise was based on the work of Çınar and Koklu [1]. The task was to classify rice species. The data consisted of images (size: 250 x 250 pixels) including one rice grain. The rice images were taken with a camera placed in a closed box to ensure the same lighting environment for all samples. The rices were captured on a black background. The original images were color images, which then were converted to grayscale images. The data is openly available at https://www.muratkoklu.com/datasets (https://www.muratkoklu.com/datasets). The original data includes images for five rice species, 75000 images in total. In this exercise, we used only a small random sample of the original data (100 images for each rice species).

Different features were derived from the rice images. First, a contour around the rice grain was detected for each image. Then, the detected contour was used to calculate color features using the pixels within the contour, such as mean, variance, skewness, and kurtosis of the YCbCr-color channel values. Morphological features such as area, perimeter, major and minor axis length, equivalent diameter, compactness and shape factors were determined using an ellipse fitted to the detected contour. The gathered data was explored using histograms, pair plots and Principal Component Analysis (PCA).

The feature data was standardized. Three classifiers were used for classification: Random Forest (RF), Support Vector Machine (SVM) and Multi-Layer Perceptron (MLP). The best models were selected using hyperparameter selection and 5-fold 3 times repeated Kfold cross validation (cv). The model performance was estimated using nested cv (stratified 10-fold cv in the outer loop and 3 times repeated 5-fold in the inner loop). The performance of the models was evaluated using accuracy.

The estimated mean accuracies were 0.994/0.998/0.968 for RF/SVM/MLP, respectively. Morphological and shape features seemed to be the most important features according to RF feature importance.

Reference

[1] İ. Çınar and M. Koklu. Identification of rice varieties using machine learning algorithms. Journal of Agricultural Sciences, 28(2):307–325, 2022. doi: 10.15832/ankutbd.862482.

2 Part 2

3 Part 3

In [1]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
# import cv2 as cv
# import glob, os
import itertools
# from random import sample
# from random import seed
# from scipy.stats import skew
# from scipy.stats import kurtosis
# from scipy.stats import entropy
import scipy.stats
from sklearn.decomposition import PCA
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_predict

from IPython.display import display, HTML
display(HTML("<style>.container { width:90% !important; }<
import time
```

3.1 Performance estimation

3.1.1 Random forest

Let's estimate the performance of each model using nested cross-validation. Again, we'll use 5-fold repeated cross validation with 3 repetitions and the same parameter ranges as in Part 2 for the inner loop.
For the outer loop we'll use 10-fold Stratified Kfold cross-validation.

Study the following implementation of performance estimation for Random Forest model.
You do not need to run the cell below, you can import the results and start from there.

In [ ]:

```
df = pd.read_parquet('../training_data/rice_feature_data.p

feats = ['mean_y', 'var_y', 'skew_y', 'kurt_y',
         'mean_cr', 'var_cr', 'skew_cr', 'kurt_cr',
         'mean_cb', 'var_cb', 'skew_cb', 'kurt_cb',
         'major_axis_length', 'minor_axis_length',
         'area', 'perimeter',
         'equivalent_diameter', 'compactness',
         'shape_factor1', 'shape_factor2']

# Z-Score using pandas
for feat in feats:
    df['{}_Z'.format(feat)] = (
        df[feat] - df[feat].mean()) / df[feat].std()

# Z-scored feature names
feats_Z = [feat + '_Z' for feat in feats]

y = df['class']
X = df[feats_Z]

kf_outer = StratifiedKFold(
    n_splits=10,
    random_state=10,
    shuffle=True)
kf_inner = RepeatedKFold(
    n_splits=5,
    n_repeats=3,
    random_state=5)

n_estimators = range(100, 350, 50)
max_features = ['sqrt', 'log2', None]
bootstrap = [True, False]

parameters={
    'n_estimators': n_estimators,
    'max_features': max_features,
    'bootstrap': bootstrap}

rf = RandomForestClassifier(random_state=20)

results = []

start_time = time.time()

for fold, (train_index, test_index) in enumerate(
    kf_outer.split(X, y)):
    X_train, X_test = X.iloc[train_index], X.iloc[test_ind
    y_train, y_test = y.iloc[train_index], y.iloc[test_ind

    # temporary dataframe for saving the results
    temp_res = df.iloc[test_index].copy()
    temp_res['fold'] = [fold] * temp_res.shape[0]

    # select the best model
    gscv = GridSearchCV(
        rf,
        parameters,
        cv=kf_inner,
        return_train_score=False,
        n_jobs=-1)
```

```
    gscv.fit(X_train, y_train)

    # use the selected model to predict the classes for the
    temp_res['pred_class'] = gscv.predict(X_test)

    # save the best hyperparameter combination
    temp_res['best_params'] = [
        gscv.best_params_] * temp_res.shape[0]

    results.append(temp_res)

results = pd.concat(results, axis=0)
results['accuracy'] = results.apply(
    lambda i: 1 if i['class'] == i['pred_class'] else 0, a
results.index.name = 'orig_index'
# sort the results in the same order as in df
results = results.sort_values(
    'orig_index').reset_index(drop=True)
results.to_parquet('../results/rf_results.parquet')

print("--- %s seconds ---" % (time.time() - start_time))
```

In [3]:

```
# results.index.name = 'orig_index'
# results = results.sort_values('orig_index').reset_index(
# results.to_parquet('../training_data/rf_results.parquet'
```

Mini-exam question:

Question 2. Nested cross validation

- is used to estimate the perfomance of a model selected in a particular way

Save the supplied (can be found from Moodle) rf_results.parquet-file in a folder named "results".

In [2]:

```
# start from here!
results = pd.read_parquet('../results/rf_results.parquet')
```

Answer the questions/tasks:

- Which parameter combination is selected the best most often?
- Calculate the mean accuracy of each fold and the mean accuracy over all folds
- Calculate the confusion matrix over all samples

3

```
# just check that there are 50 test samples in each fold
results.groupby('fold').size()
```

Out[3]:

```
fold
0    50
1    50
2    50
3    50
4    50
5    50
6    50
7    50
8    50
9    50
dtype: int64
```

In [4]:

```
results['param_comb'] = results.apply(
    lambda i: str(i.best_params['bootstrap']) +
    '_' + str(i.best_params['max_features']) +
    '_' + str(i.best_params['n_estimators']), axis=1)
results.groupby(['param_comb']).size() / int(results.shape
```

Out[4]:

```
param_comb
False_sqrt_100    4.0
False_sqrt_150    1.0
True_sqrt_100     1.0
True_sqrt_150     2.0
True_sqrt_250     1.0
True_sqrt_300     1.0
dtype: float64
```

The following parameter combination is chosen most often as the best (4 times ot of 10):

- bootstrap: False
- max_features: sqrt
- n_estimators: 100

As we can see, different hyperparameter combinations can be selected as the best in different cv rounds.

5

```
display(results.groupby('fold').accuracy.mean())
print('Mean accuracy over all folds:',
      results.groupby(
          'fold').accuracy.mean().mean().round(3))
```

```
fold
0    0.98
1    1.00
2    1.00
3    1.00
4    1.00
5    1.00
6    0.96
7    1.00
8    1.00
9    1.00
Name: accuracy, dtype: float64

Mean accuracy over all folds: 0.994
```

Mini-exam question:
Question 3. The smallest fold-wise accuracy for Random Forest is
Answer: 0.96

Note: The following answers were accepted: 0.96, 0,96, 96, 96%

Here's another way to implement nested cross-validation which uses *cross_val_predict* function from scikit-learn.
Modify it to make the perfomance estimation for MLP and SVM (again, use the paramater ranges given in the Part 2).
For both models:

- Calculate the accuracy over all samples
- Show the confusion matrix (you can combine all samples in one confusion matrix as all folds are stratified and include the same number of samples)

Compare the three models. Which one performs the best?

In [6]:

```
feats_Z = ['mean_y_Z', 'var_y_Z',
 'skew_y_Z', 'kurt_y_Z',
 'mean_cr_Z', 'var_cr_Z',
 'skew_cr_Z', 'kurt_cr_Z',
 'mean_cb_Z', 'var_cb_Z',
 'skew_cb_Z', 'kurt_cb_Z',
 'major_axis_length_Z',
 'minor_axis_length_Z',
 'area_Z', 'perimeter_Z',
 'equivalent_diameter_Z',
 'compactness_Z',
 'shape_factor1_Z',
 'shape_factor2_Z']
```

7

```
# another way
start_time = time.time()
y = results['class']
X = results[feats_Z]

rf = RandomForestClassifier(random_state=20)

kf_outer = StratifiedKFold(
    n_splits=10,
    random_state=10,
    shuffle=True)
kf_inner = RepeatedKFold(
    n_splits=5,
    n_repeats=3,
    random_state=5)

n_estimators = range(100, 350, 50)
max_features = ['sqrt', 'log2', None]
bootstrap = [True, False]

parameters={
    'n_estimators': n_estimators,
    'max_features': max_features,
    'bootstrap': bootstrap}

gscv = GridSearchCV(
    rf,
    parameters,
    cv=kf_inner,
    return_train_score=False,
    n_jobs=-1)
nested_pred = cross_val_predict(
    gscv, X, y, cv=kf_outer)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 169.75516724586487 seconds ---
```

In [8]:

```
# Predicted values on columns, true values on rows
display(pd.DataFrame(
    confusion_matrix(y, nested_pred)).rename(
    mapper=dict(
        zip(range(5),
            ['Arb', 'Bas', 'Ips', 'Jas', 'Kar'])),
    axis=0).rename(
    mapper=dict(
        zip(range(5),
            ['Arb', 'Bas', 'Ips', 'Jas', 'Kar'])),
    axis=1))
```

	Arb	Bas	Ips	Jas	Kar
Arb	99	0	0	0	1
Bas	0	100	0	0	0
Ips	0	0	99	1	0
Jas	0	0	0	100	0
Kar	1	0	0	0	99

9

```
1 accuracy_score(y, nested_pred)
```

Out[9]: 0.994

In [10]:

```
1 # number of hyperparameter combinations
2 a = [item[1] for item in parameters.items()]
3 len(list(itertools.product(*a)))
```

Out[10]: 30

3.1.2 SVM

In [11]:

```
y = results['class']
X = results[feats_Z]
```

In [12]:

```
start_time = time.time()
kf_outer = StratifiedKFold(
    n_splits=10,
    random_state=10,
    shuffle=True)
kf_inner = RepeatedKFold(
    n_splits=5,
    n_repeats=3,
    random_state=5)

gamma = ['scale', 'auto']
C = [0.1, 1, 10, 100]
kernel = ['linear', 'rbf', 'poly']

parameters = {'C': C,
              'gamma': gamma,
             'kernel': kernel}

svm = SVC()

gscv = GridSearchCV(
    svm,
    parameters,
    cv=kf_inner,
    return_train_score=False,
    n_jobs=-1)
nested_pred = cross_val_predict(
    gscv, X, y, cv=kf_outer)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 2.7521047592163086 seconds ---
```

In [13

```
# Predicted values on columns, trues on rows
display(pd.DataFrame(
    confusion_matrix(y, nested_pred)).rename(
    mapper=dict(
        zip(range(5),
            ['Arb', 'Bas', 'Ips', 'Jas', 'Kar'])),
    axis=0).rename(
    mapper=dict(
        zip(range(5),
            ['Arb', 'Bas', 'Ips', 'Jas', 'Kar'])),
    axis=1))
```

	Arb	Bas	Ips	Jas	Kar
Arb	99	0	0	1	0
Bas	0	100	0	0	0
Ips	0	0	100	0	0
Jas	0	0	0	100	0
Kar	0	0	0	0	100

In [14]:

```
accuracy_score(y, nested_pred)
```

Out[14]: 0.998

In [15]:

```
# number of hyperparameter combinations
a = [item[1] for item in parameters.items()]
len(list(itertools.product(*a)))
```

Out[15]: 24

3.1.3 MLP

In [16]:

```
start_time = time.time()
kf_outer = StratifiedKFold(
    n_splits=10,
    random_state=10,
    shuffle=True)
kf_inner = RepeatedKFold(
    n_splits=5,
    n_repeats=3,
    random_state=5)

mlp = MLPClassifier(
    max_iter=500,
    early_stopping=True,
    random_state=20)

hidden_layer_sizes = [(i,) for i in range(15, 45, 5)]
activation = ['tanh', 'relu']
solver = ['sgd', 'adam']
val_fr = [0.1, 0.3]
alpha = [0.01, 0.1, 1]

parameters = {
    'hidden_layer_sizes': hidden_layer_sizes,
    'activation': activation,
    'solver': solver,
    'validation_fraction': val_fr
}

gscv = GridSearchCV(
    mlp,
    parameters,
    cv=kf_inner,
    return_train_score=False,
    n_jobs=-1)
nested_pred = cross_val_predict(
    gscv, X, y, cv=kf_outer)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 43.2096381187439 seconds ---
```

In [17]:

```
# Predicted values on columns, trues on rows
display(pd.DataFrame(
    confusion_matrix(y, nested_pred)).rename(
    mapper=dict(
        zip(range(5),
            ['Arb', 'Bas', 'Ips', 'Jas', 'Kar'])),
    axis=0).rename(
    mapper=dict(
        zip(range(5),
            ['Arb', 'Bas', 'Ips', 'Jas', 'Kar'])),
    axis=1))
```

	Arb	Bas	Ips	Jas	Kar
Arb	93	0	0	1	6
Bas	0	100	0	0	0
Ips	0	0	100	0	0
Jas	4	2	0	93	1
Kar	1	0	0	1	98

In [18]:

```
accuracy_score(y, nested_pred)
```

Out[18]: 0.968

In [19]:

```
# number of hyperparameter combinations
a = [item[1] for item in parameters.items()]
len(list(itertools.product(*a)))
```

Out[19]: 48

Mini-exam question:

Question 4. Analyze the confusion matrices. Were the acquired results as expected? Why?

Answer: SVM had the highest accuracy, only once Arborio was predicted as Jasmine. RF gave the second best result: Arborio and Karacadag were once mislabeled as each other and Ipsala once as Jasmine. MLP made more errors than the other two classifiers: it struggled with Arborio and Jasmine the most. According to the PCA figure from part 2, it was expected that it is possible to train well-behaving classifiers, however more mislabeling might have been expected between Karacadag and Arborio. We used only two PCA components, probably we could have seen better discrimination between Karacadag and Arborio already with 3-dimensional PCA.

3.2 Discussion

Mini-exam question:
Question 5. Discuss you results:

- Ponder the limitations and generalization of the models. How well will they perform for data outside this data set? Explain. **1p**
- Compare your results with the original article. Are they comparable? **1p**

- Ponder applications for these kind of models, who could benefit from them? Ponder also what would be interesting to study more on this area. **1p**

SVM gave the best results. With this data set size, it was very fast to train and evaluate. RF gave almost as good results as SVM, and it also offers an interpretation, how important each feature is. However, it took rather long time to evaluate its performance. MLP reached a bit lower accuracy than the other classifiers, but it would probably work better with a larger data set and when using a more extensive hyperparameter selection.

The sample images were prepared with a special preprocessing procedure. We can expect that the model produces similar results for rices from the same distribution (similar growing environment, same image preprocessing pipeline). The generalization outside this dataset, i.e. for the same rice species but from a different growing environment and conditions and with another preprocessing pipeline (differing lighting, resolution, cropping) is something which we cannot determine directly from the acquired results but should be tested separately. It would be interesting to test the models just by taking random rice samples available e.g. from our own kitchen.

Our results are rather comparable with the original article, however some differences exist: The original article compared six classifiers, including the three studied classifiers here. They used more features than we did. Performance of the classifiers were estimated using 10-fold cross validation. It seems they did not perform model selection, but the model parameters were fixed, e.g. they used four hidden layers and sigmoid activation for MLP. In this exercise we used only a small sample, 500 images in total. We made model selection by selecting the best model from several hyperparameter combinations for each studied classifier and estimated their performance using nested cross validation. The accuracies in the original article were 99.91/99.88/99.85% for MLP/RF/SVM, respectively. The results for RF and SVM classifiers were very close in our subset, whereas the accuracy for MLP was somewhat lower. This was probably due to the fact that we had only one hidden layer. In small data sets outliers may have a large impact. Generally, it is important to use cross validation or repeated splits, single splits to training and test set would not give a reliable estimation of the performance. In this subset, the performance evaluation results were very close to the accuracies obtained in the model selection phase, which shows that the selected models are performing consistently across different partitions of the dataset, so overfitting was not a problem here.

It is interesting, how these models could be deployed. The article states that human decisions are inconsistent, subjective and slow, and machine vision systems could be cost-effective, fast and accurate technique. The machine learning algorithm requires carefully prepared samples with only one rice grain present, which must limit the otherwise fast process of making the classification. The same type of machine learning classification scheme can be (and has been) used for other plant classification as well. It could also be used for detecting rotten or broken individuals from good ones. The results of these kind of models could benefit biodiversity researchers, and ultimately farmers and sellers. Personally, I'd like to see a research which studies more in detail the variation (due to different growing conditions and production stages) within one species, and how to tackle that (e.g. the bimodal feature values for Jasmine).

3.3 Reflection

Mini-exam question:

Question 6. Reflect your work: (0.5 points per answered question)

- Did you have any (mis)conceptions which changed while doing this work?
- What did you learn?
- What was difficult?
- In which aspects of work did you succeed well?
- How could you improve you working process?
- How can you deploy the learnings from this exercise in future?