This is the template for the image recognition exercise.
Some **general instructions**, read these carefully:

- The final assignment is returned as a clear and understandable *report*
  - define shortly the concepts and explain the phases you use
  - use the Markdown feature of the notebook for larger explanations
- return your output as a *working* Jupyter notebook
- name your file as Exercise_MLPR2023_Partx_uuid.jpynb
  - use the uuid code determined below
  - use this same code for each part of the assignment
- write easily readable code with comments
  - if you exploit code from web, provide a reference
- it is ok to discuss with a friend about the assignment. But it is not ok to copy someone's work. Everyone should submit their own implementation
  - in case of identical submissions, both submissions are failed

Deadlines:

- Part 1: Mon 6.2 at 23:59
- Part 2: Mon 20.2 at 23:59
- Part 3: Mon 6.3 at 23:59

No extensions for the deadlines

- after each deadline, example results are given, and it is not possible to submit anymore

If you encounter problems, Google first and if you can't find an answer, ask for help

- Moodle area for questions
- pekavir@utu.fi
- teacher available for questions
  - Monday 30.1 at 14:00-15:00 room 407B Honka (Agora 4th floor)
  - Monday 13.2 at 14:00-15:00 room 407B Honka (Agora 4th floor)
  - Thursday 2.3 at lecture 10:15-12:00

Grading

The exercise covers a part of the grading in this course. The course exam has 5 questions, 6 points of each. Exercise gives 6 points, i.e. the total score is 36 points.

From the template below, you can see how many exercise points can be acquired from each task. Exam points are given according to the table below:

7 exercise points: 1 exam point
8 exercise points: 2 exam points
9 exercise points: 3 exam points
10 exercise points: 4 exam points
11 exercise points: 5 exam points
12 exercise points: 6 exam points

To pass the exercise, you need at least 7 exercise points, and at least 1 exercise point from each Part.

Each student will grade one submission from a peer and their own submission. After each Part deadline, example results are given. Study them carefully and perform the grading according to the given instructions. Mean value from the peer grading and self-grading is used for the final points.

In [ ]:
```
# import uuid
# # Run this cell only once and save the code. Use the same id code for each Part.
# # Printing random id using uuid1()
# print ("The id code is: ",end="")
# print (uuid.uuid1())
```

The id code is: 1Nw3ELgo7tH-yDZPZh7H8sLHY7QHtipYW

Part 1

Read the original research article:

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.

https://dergipark.org.tr/en/download/article-file/1513632

Introduction (1 p)

Will be written in Part 3

Preparations of the data (1 p)

Make three folders in your working folder: "notebooks", "data" and "training_data". Save this notebook in "notebooks" folder.

Perform preparations for the data

- import all the packages needed for this notebook in one cell
- import the images. Data can be found from (downloading starts as you press the link) https://www.muratkoklu.com/datasets/vtdhnd09.php
  - save the data folders "Arborio", "Basmati" and "Jasmine" in "data" folder
- take a random sample of 100 images from Arborio, Basmati and Jasmine rice species (i.e. 300 images in total)
- determine the contour of each rice (you can use e.g. *findContours* from OpenCV)
- plot one example image of each rice species, including the contour

Feature extraction (2 p)

Gather the feature data

Color features (15)

- Calculate the following color features for each image, including only the pixels within the contour (you can use e.g. *pointPolygonTest* from OpenCV)
  - Mean for each RGB color channel
  - Variance for each RGB color channel
  - Skewness for each RGB color channel
  - Kurtosis for each RGB color channel
  - Entropy for each RGB color channel

Dimension features (6)

- Fit an ellipse to the contour points (you can use e.g. *fitEllipse* from OpenCV)
- Plot one example image of each rice species including the fitted ellipse
- Calculate the following features for each image (for details, see the original article)
  - the major axis length the ellipse
  - the minor axis length of the ellipse
  - area inside the contour (you can use e.g. *contourArea* from OpenCV)
  - perimeter of the contour (you can use e.g. *arcLength* from OpenCV)
  - roundness
  - aspect ratio

Gather all the features in one array or dataframe: one data point in one row, including all feature values in columns.
For each data point, include also information of the original image and the label (rice species). Save the data in "training_data" folder.

Part 2

Data exploration (2 p)

- Standardize the data,
- Plot a boxplot of each feature
- Plot histogram of each feature use a different color for each class
- Plot pairplot (each feature against each feature and the label against each feature)
- Discuss your findings from the above figures, e.g. can you spot features which might be very useful in predicting the correct class?
- Fit PCA using two components
- Plot the PCA figure with two components, color the data points according to their species
- Can you see any clusters in PCA? Does this figure give you any clues, how well you will be able to classify the image types? Explain.
- How many PCA components are needed to cover 99% of the variance?
- Make clear figures, use titles and legends for clarification.

In [7]:
```
import pandas as pd
import numpy as np
import pyarrow.parquet as pq
import os
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

from sklearn.model_selection import GridSearchCV, RepeatedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
```

In [11]:
```
# Reading data from the provided solution of part 1 in moodle.

data_path = os.path.join('..', 'training_data', 'rice_feature_data.parquet')
df = pd.read_parquet(data_path)
df.head(10)
```

Out[11]:

	mean_b	var_b	skew_b	kurt_b	entr_b	mean_g	var_g	skew_g	kurt_g	entr_g	...	kurt_r	entr_r	major_axis_length	minor_axis_length	area	perimeter	roundness	aspect_ratio	class
0	234.887380	355.347614	-0.553562	-0.656321	5.263077	230.505217	477.841226	-0.400259	-0.966816	5.506168	...	-0.977245	5.555513	138.844086	66.395470	7150.0	361.847760	0.686221	2.091168	Arb
1	217.458704	82.110749	-0.028677	2.413071	5.120533	204.133926	83.481241	0.081193	2.053817	5.156789	...	2.228074	5.128429	129.027618	62.808922	6220.0	324.936073	0.740294	2.054288	Arb
2	226.921111	111.567639	-0.474308	0.647867	5.384632	216.544897	122.179148	-0.672709	0.767762	5.407488	...	0.750525	5.396245	130.407501	60.895866	6017.0	331.137083	0.689563	2.141484	Arb
3	233.894178	239.658145	-1.524592	2.450121	5.489109	231.712360	293.609549	-1.386229	1.658084	5.552210	...	1.618729	5.642129	133.773544	68.909790	7129.5	356.492421	0.704966	1.941285	Arb
4	208.110291	224.030886	-0.940787	0.363133	5.547121	208.928796	255.065451	-0.964120	0.253140	5.582591	...	0.247824	5.581644	137.687790	69.746933	7476.5	362.007140	0.716925	1.974105	Arb
5	213.680783	237.934552	-0.404852	-0.215632	5.769709	206.089118	199.506263	-0.515955	0.121052	5.666709	...	0.182664	5.632942	141.298264	70.406998	7622.5	369.806129	0.700421	2.006878	Arb
6	230.017030	175.861492	-0.915677	0.353155	5.458395	226.342310	227.623418	-0.855632	0.070042	5.627506	...	0.118209	5.595609	145.034149	68.884285	7740.5	374.007140	0.695375	2.105475	Arb
7	227.863776	205.027217	-0.518637	1.344448	5.726552	216.697695	215.212673	-0.456130	0.945222	5.731969	...	1.020406	5.716535	153.568954	79.152504	9287.5	407.119838	0.704148	1.940165	Arb
8	211.434884	200.361109	-0.159062	-0.763165	5.701034	207.077674	210.450246	-0.094834	-0.656990	5.727006	...	-0.706007	5.717895	122.196671	69.346169	6596.5	342.634556	0.706092	1.762126	Arb
9	230.416772	245.254692	-0.519230	0.310937	5.786503	221.587139	241.954589	-0.405279	0.129115	5.786233	...	0.148829	5.745217	145.594498	79.186180	8857.0	389.646749	0.733085	1.838635	Arb

10 rows × 23 columns

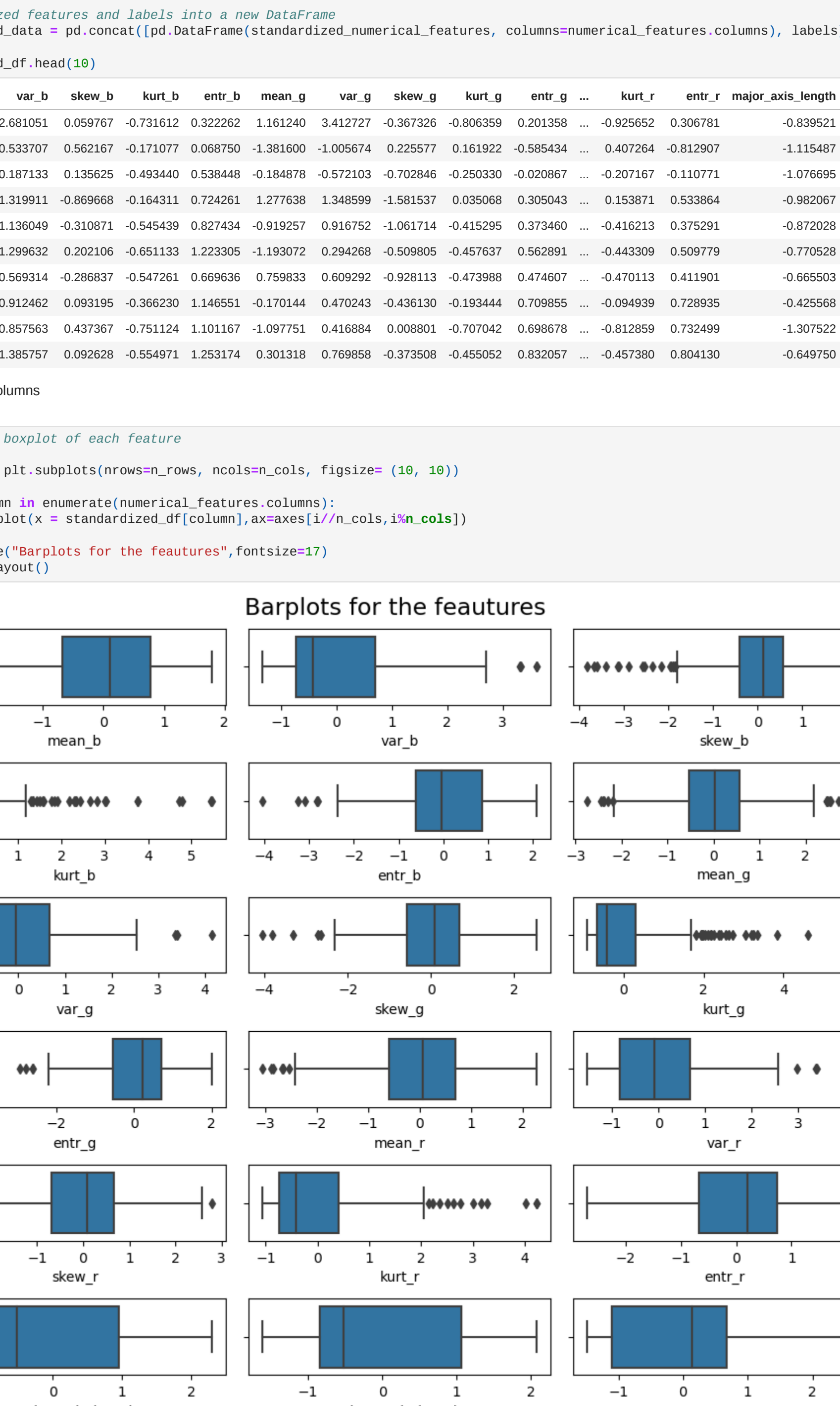
In [16]:
```
# Standardization the data

labels = data.loc[:, data.columns[-2:]]
numerical_features = data.loc[:, data.columns[:-2]]


scaler = StandardScaler()
standardized_numerical_features = scaler.fit_transform(numerical_features)

# Standardized features and labels into a new DataFrame
standardized_data = pd.concat([pd.DataFrame(standardized_numerical_features, columns=numerical_features.columns), labels], axis=1)

standardized_df.head(10)
```

Out[16]:

	mean_b	var_b	skew_b	kurt_b	entr_b	mean_g	var_g	skew_g	kurt_g	entr_g	...	kurt_r	entr_r	major_axis_length	minor_axis_length	area	perimeter	roundness	aspect_ratio	class	cl
0	0.396251	2.681051	0.059767	-0.731612	0.322262	1.161240	3.412727	-0.367326	-0.806359	0.201358	...	-0.925652	0.306781	-0.839521	0.941032	0.136762	-0.533615	0.994183	-1.046037	Arb	
1	-0.971039	-0.533707	0.562167	-0.171077	0.068750	-1.381600	-1.005674	0.225577	0.161922	-0.585434	...	0.407264	-0.812907	-1.115487	0.622454	-0.518992	-1.186728	1.493185	-1.080946	Arb	
2	-0.228707	-0.187133	0.135625	-0.493440	0.538448	-0.184878	-0.572103	-0.702846	-0.250330	-0.020867	...	-0.207167	-0.110771	-1.076695	0.452525	-0.662130	-1.077008	1.025026	-0.998410	Arb	
3	0.318334	1.319911	-0.869668	-0.164311	0.724261	1.277638	1.348599	-1.581537	0.035068	0.305043	...	0.153871	0.533864	-0.982067	1.164369	0.122308	-0.628372	1.167168	-1.187910	Arb	
4	-1.704427	1.136049	-0.310871	-0.545439	0.827434	-0.919257	0.916752	-1.061714	-0.415295	0.373460	...	-0.416213	0.375291	-0.872028	1.238729	0.366982	-0.530795	1.277529	-1.156844	Arb	
5	-1.267418	1.299632	0.202106	-0.651133	1.223305	-1.193072	0.294268	-0.509805	-0.457637	0.562891	...	-0.443309	0.509779	-0.770528	1.297360	0.469928	-0.392800	1.125222	-1.125822	Arb	
6	0.014169	0.569314	-0.286837	-0.547261	0.669636	0.759833	0.609292	-0.928113	-0.473988	0.474607	...	-0.470113	0.411901	-0.665503	1.162104	0.553131	-0.318468	1.078657	-1.032495	Arb	
7	-0.154755	0.912462	0.093195	-0.366230	1.146551	-0.170144	0.470243	-0.436130	-0.193444	0.709855	...	-0.094939	0.728935	-0.425568	2.074187	1.643940	0.267426	1.159620	-1.188970	Arb	
8	-1.443610	0.857563	0.437367	-0.751124	1.101167	-1.097751	0.416884	0.008801	-0.707042	0.698678	...	-0.812859	0.732499	-1.307522	1.203131	-0.253517	-0.873572	1.177554	-1.357495	Arb	
9	0.045529	1.385757	0.092628	-0.554971	1.253174	0.301318	0.769858	-0.373508	-0.455052	0.832057	...	-0.457380	0.804130	-0.649750	2.077179	1.340389	-0.041742	1.426656	-1.285074	Arb	

10 rows × 23 columns

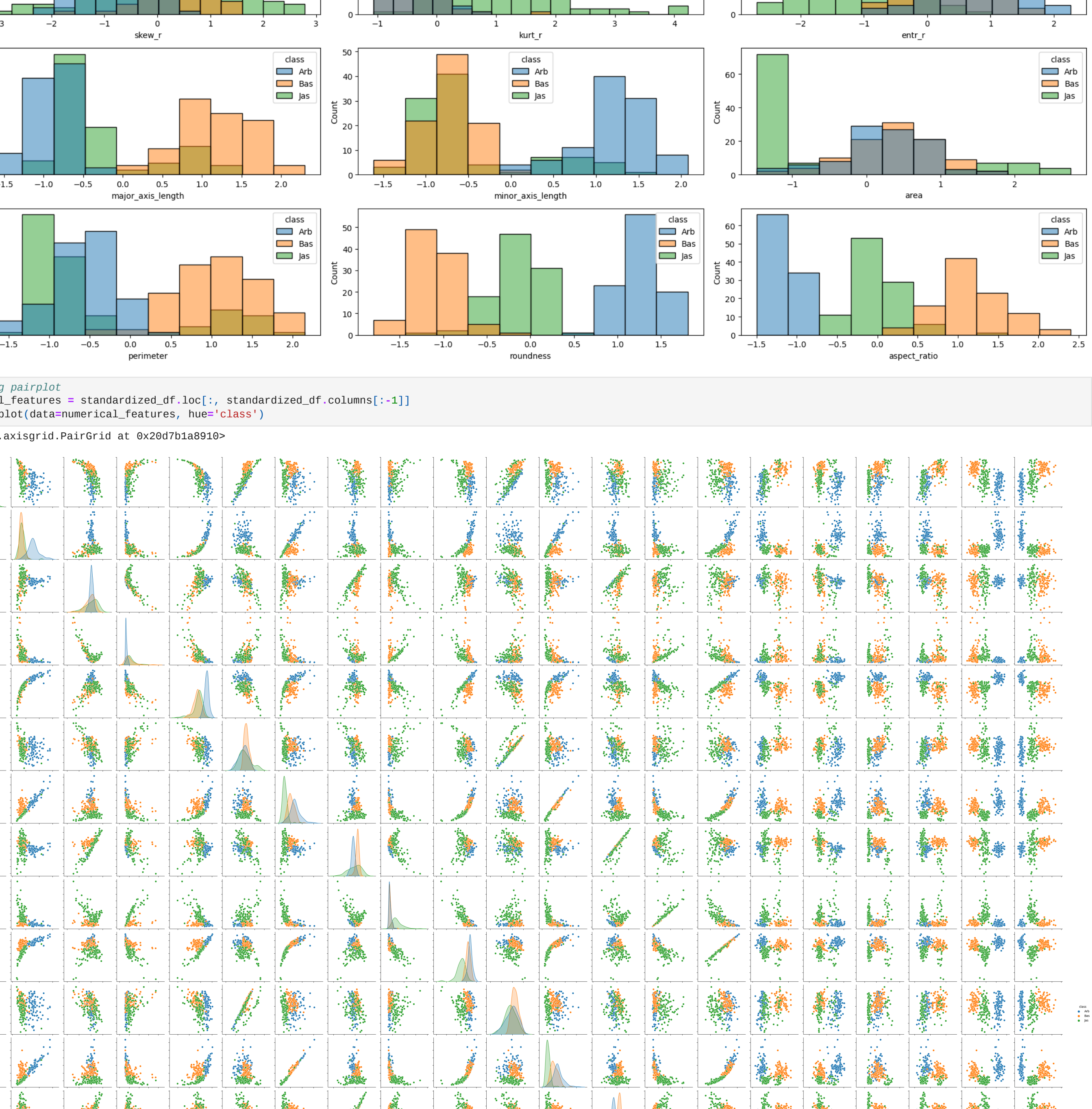
In [20]:
```
# Ploting a boxplot of each feature

fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize= (10, 10))

for i, column in enumerate(numerical_features.columns):
    sns.boxplot(x = standardized_df[column],ax=axes[i//n_cols,i%n_cols])

fig.suptitle("Barplots for the feautures",fontsize=17)
plt.tight_layout()
```



In [29]:
```
# Creating subplots
fig, axs = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(20, 20))

# Ploting a histogram for each feature by using different color for each class
for i, column in enumerate(standardized_data.columns[:-2]):
    row_index, col_index = divmod(i, n_cols)
    sns.histplot(data=standardized_data, x=column, hue='class',ax=axs[row_index, col_index])

# Adding title to the figure
fig.suptitle('Histograms of Features', fontsize=20)

# Adjusting the layout of the subplots
plt.tight_layout()
```

In [35]:
```
# Ploting pairplot
numerical_features = standardized_df.loc[:, standardized_df.columns[:-1]]
sns.pairplot(data=numerical_features, hue='class')
```

Out[35]: <seaborn.axisgrid.PairGrid at 0x20d7b1a8910>

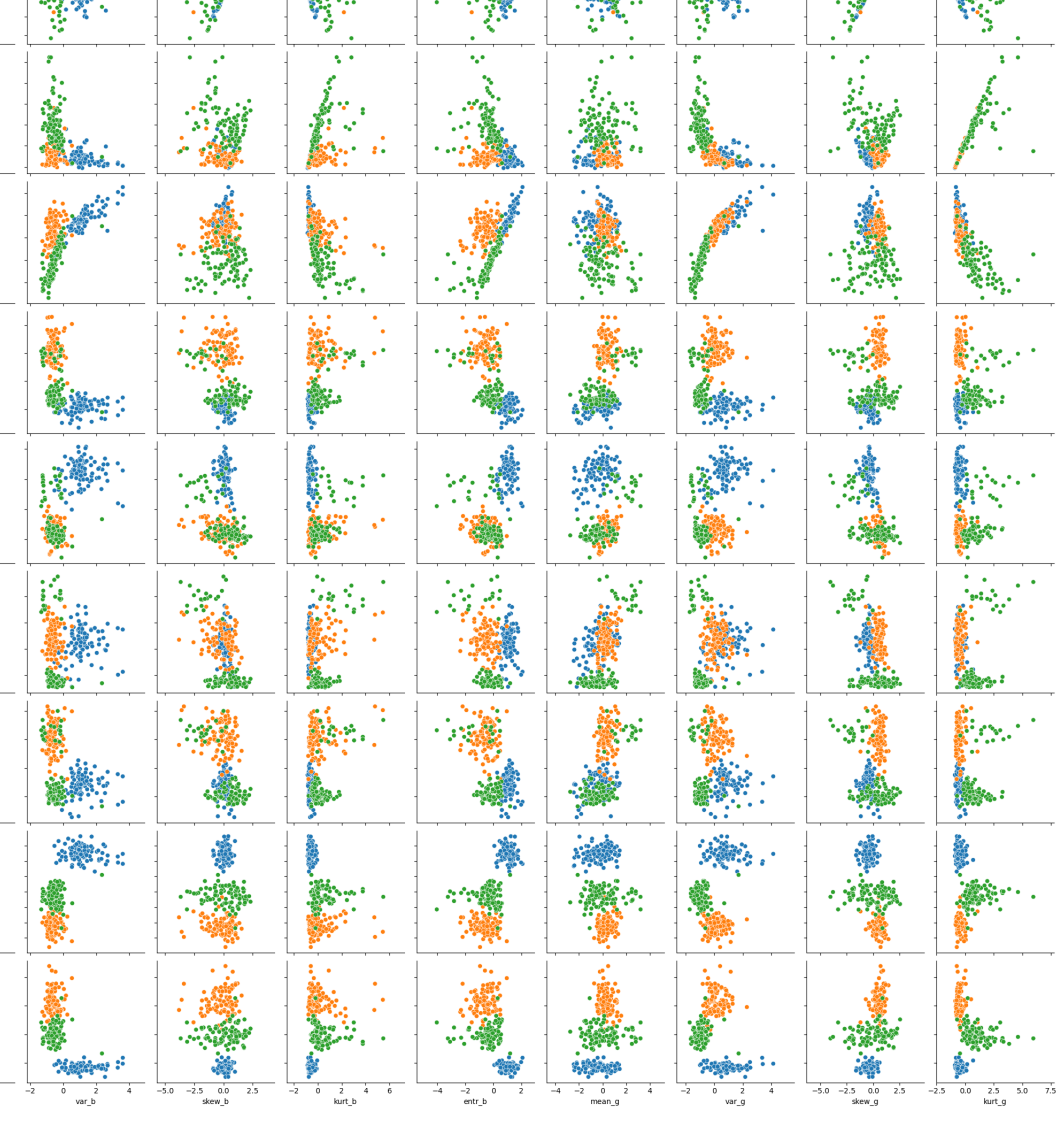


- Discuss your findings from the above figures, e.g. can you spot features which might be very useful in predicting the correct class?

The features of the classes are distinctive but vaguely seen.

In [46]:
```
# Fitting PCA with 2 components
pca = PCA(n_components=2)

# fit PCA to the standardized features
pca.fit(standardized_numerical_features)

# transform the features into principal components
principal_components = pca.transform(standardized_numerical_features)

# create a DataFrame to hold the principal components
principal_df = pd.DataFrame(data = principal_components, columns = ['PC1', 'PC2'])

# concatenate the labels to the principal components DataFrame
principal_df['class'] = standardized_df['class']
```

In [48]:
```
#Ploting PCA figure with two components, color the data points according to their species

sns.scatterplot(data=principal_df, x='PC1', y='PC2', hue='class')
plt.title('PCA Figure with Two Components')
plt.show()
```



In [49]:
```
# Number of PCA components are needed to cover 99% of the variance

# Create a PCA object with all the components
pca = PCA()

# Fit the PCA model to the data
pca.fit(standardized_numerical_features)

# Calculate the cumulative explained variance ratio
cumulative_variance_ratio = np.cumsum(pca.explained_variance_ratio_)

# Find the number of components that explain 99% of the variance
n_components = np.argmax(cumulative_variance_ratio >= 0.99) + 1

print(f"{n_components} components are needed to cover 99% of the variance.")
```

```
11 components are needed to cover 99% of the variance.
```

- Can you see any clusters in PCA? Does this figure give you any clues, how well you will be able to classify the image types? Explain?

Yes, cluster can be seen easily in the PCA. In the PCA we can classify the every class of rice. It is also manifested that the there are more similarities between the features of Basmati & Jasmine rice comparing to the Arborio & Jasmine rice.

Model selection (2 p)

Select the best model for each classifier. Use 5-fold repeated cross validation with 3 repetitions (RepeatedKFold from sklearn). You can choose the hyperparameter ranges to use (i.e. from which values the best hyperparameters are selected if they are not stated below.)

- k Nearest Neighbors classifier: hyperparameter k
- random forest: hyperparameters max_depth and max_features
- MLP: use one hidden layer and Early stopping. Hyperparameters:
  - number of neurons in the hidden layer
  - activation function: logistic sigmoid function and rectified linear unit function
  - solver: stochastic gradient descent and adam
  - validation_fraction: 0.1 and 0.5

For each classifier:

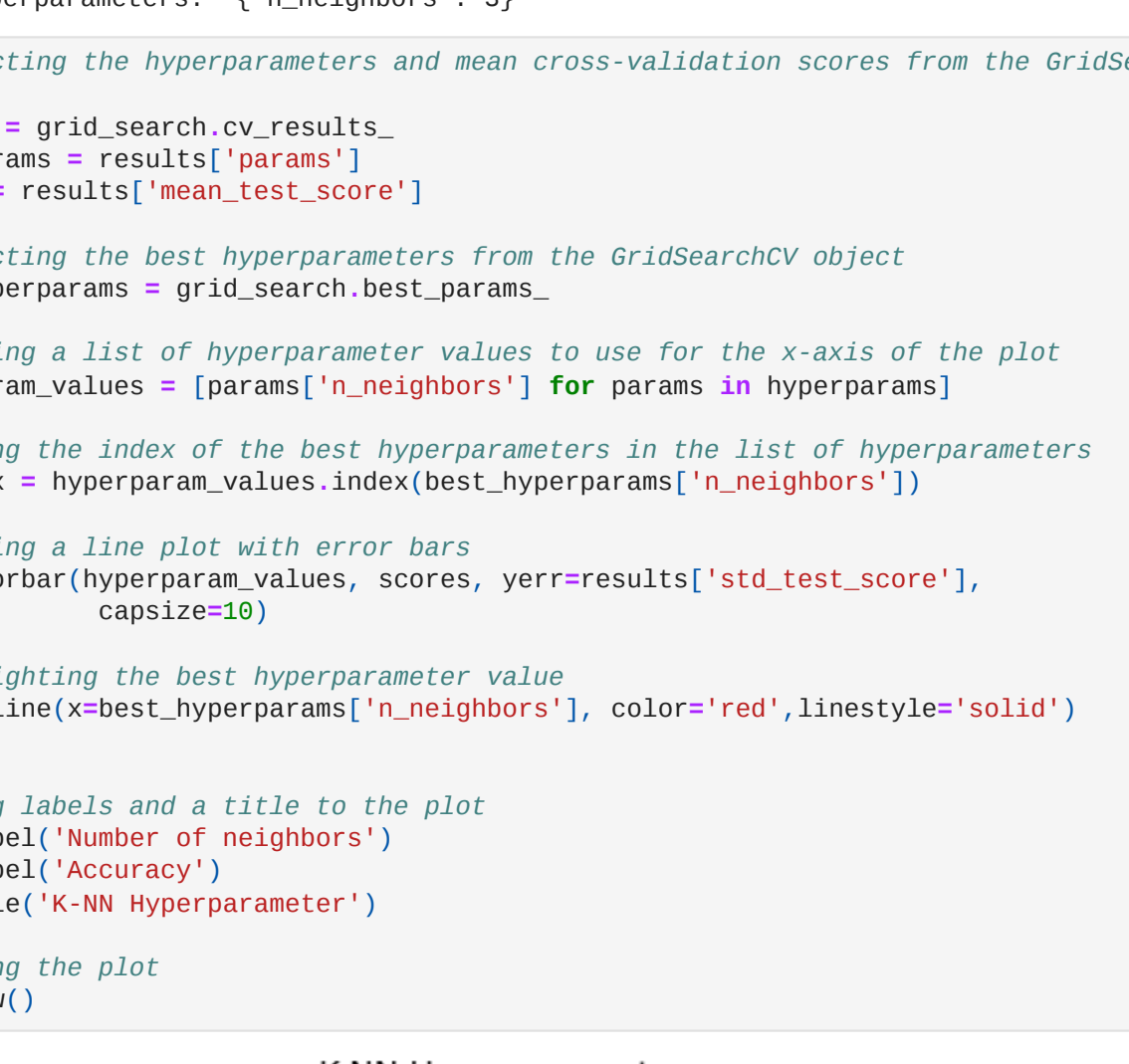
- Report the best hyperparameter or the best combination of hyperparameters.
- Plot the accuracy versus the hyperparameter/hyperparameter combination and highlight the best value.

For random forest model, report the feature importance for each feature. Which features seem to be the most important? Does this correspond with the observations you made in the data exploration? Ponder the model selection process. What things should be considered when selecting the model to be used?

In [103…
```
# Separating the features and target variable
x = standardized_numerical_features
y = data['class']

# Creating a KNeighborsClassifier object
knn = KNeighborsClassifier()

# Defining the hyperparameters to search over
parameters = {'n_neighbors': range(1, 24)}

# Defining the number of splits and repetitions for cross-validation
num_splits = 5
num_repeats = 3

# Creating a cross-validation object with the specified number of splits and repetitions
cv = RepeatedKFold(n_splits=num_splits, n_repeats=num_repeats, random_state=123)


# Performing a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(knn, parameters, cv=cv, n_jobs=-1, scoring='accuracy',
                           return_train_score=True, verbose=1)

# Fitting the grid search to the data
grid_search.fit(x, y)

# Printing the best hyperparameters
print("Best hyperparameters: ", grid_search.best_params_)
```

```
Fitting 15 folds for each of 23 candidates, totalling 345 fits
Best hyperparameters:  {'n_neighbors': 3}
```

In [104…
```
# Extracting the hyperparameters and mean cross-validation scores from the GridSearchCV object

results = grid_search.cv_results_
hyperparams = results['params']
scores = results['mean_test_score']

# Extracting the best hyperparameters from the GridSearchCV object
best_hyperparams = grid_search.best_params_

# Creating a list of hyperparameter values to use for the x-axis of the plot
hyperparam_values = [params['n_neighbors'] for params in hyperparams]

# Finding the index of the best hyperparameters in the list of hyperparameters
best_idx = hyperparam_values.index(best_hyperparams['n_neighbors'])

# Creating a line plot with error bars
plt.errorbar(hyperparam_values, scores, yerr=results['std_test_score'],
             capsize=10)

# Highlighting the best hyperparameter value
plt.axvline(x=best_hyperparams['n_neighbors'], color='red',linestyle='solid')


# Adding labels and a title to the plot
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.title('K-NN Hyperparameter')

# Showing the plot
plt.show()
```



In [117…
```
# random forest: hyperparameters max_depth and max_features

# Creating a random forest classifier
random_forest = RandomForestClassifier()

# Defining the parameter grid to search over

parameters = {
'n_estimators': [5],
'max_features': ['sqrt', 'log2', None],
'max_depth': [3, 6, 9],
'max_leaf_nodes': [3, 6, 9],
}

# Defining the number of splits and repetitions for cross-validation
num_splits = 5
num_repeats = 3

# Creating a cross-validation object with the specified number of splits and repetitions
cv = RepeatedKFold(n_splits=num_splits, n_repeats=num_repeats, random_state=123)

# perform the grid search
grid_search = GridSearchCV(rf, parameters, cv=rkf, n_jobs=-1, verbose=1, scoring='accuracy',
                           return_train_score=True)

# Fitting the grid search to the data
grid_search.fit(x, y)

# Printing the best hyperparameters and corresponding mean cross-validation score
print("Best hyperparameters: ", grid_search.best_params_)
```

```
Fitting 15 folds for each of 27 candidates, totalling 405 fits
Best hyperparameters:  {'max_depth': 6, 'max_features': 'sqrt', 'max_leaf_nodes': 9, 'n_estimators': 5}
```

In [119…
```
# Define the hyperparameters to search
parameters = {
    'hidden_layer_sizes': [(10,), (20,), (30,)],
    'activation': ['logistic', 'relu'],
    'solver': ['sgd', 'adam'],
    'learning_rate': ['constant', 'adaptive'],
    'alpha': [0.0001, 0.001, 0.01],
    'early_stopping': [True, False],
}

# Define the MLP classifier with early stopping
mlp = MLPClassifier(early_stopping=True, random_state=123)

# Defining the number of splits and repetitions for cross-validation
num_splits = 5
num_repeats = 3

# Creating a cross-validation object with the specified number of splits and repetitions
cv = RepeatedKFold(n_splits=num_splits, n_repeats=num_repeats, random_state=123)

# Perform grid search
grid_search = GridSearchCV(estimator=mlp, param_grid=parameters, cv=rkf, n_jobs=-1,
                           scoring='accuracy', return_train_score=True)

# Fit the grid search to the data
grid_search.fit(X, y)

# Print the best hyperparameters and corresponding mean cross-validation score
print('Best hyperparameters:', grid_search.best_params_)
```

```
Best hyperparameters: {'activation': 'relu', 'alpha': 0.0001, 'early_stopping': False, 'hidden_layer_sizes': (30,), 'learning_rate': 'constant', 'solver': 'adam'}
C:\Users\USER\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the o
ptimization hasn't converged yet.
  warnings.warn(
```