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 e
# # Printing random id using uuid1()
# print ("The id code is: ",end="")
# print (uuid.uuid1())
```

9299e162-ae1f-11ed-a7b1-274065d8b9ff

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

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

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 [ ]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import matplotlib.image as mimg
from matplotlib import markers
import os, os.path as path
from scipy.stats import kurtosis, skew, entropy

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import RepeatedKFold, GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier

import seaborn as sns
from sklearn import metrics
```

```
import math
```

In [ ]:
```
rice_df = pd.read_csv("../training_data/data.csv",index_col=0)
```

In [ ]:
```
features = list(rice_df.select_dtypes(exclude="object"))
features
```

Out[ ]:
```
['RGB_R_mean',
 'RGB_G_mean',
 'RGB_B_mean',
 'RGB_R_kurtosis',
 'RGB_G_kurtosis',
 'RGB_B_kurtosis',
 'RGB_R_skew',
 'RGB_G_skew',
 'RGB_B_skew',
 'RGB_R_entropy',
 'RGB_G_entropy',
 'RGB_B_entropy',
 'RGB_R_var',
 'RGB_G_var',
 'RGB_B_var',
 'area',
 'perimeter',
 'major_axis_ellipse',
 'minor_axis_ellipse',
 'cx',
 'cy',
 'aspect_ratio']
```

In [ ]:
```
rice_df.sample(5)
```

Out[ ]:

	path	label	RGB_R_mean	RGB_G_mean	RGB_B_mean	RGB_R_ku
144	../data/Jasmine/Jasmine (10588).jpg	Jasmine	89.523404	84.742047	84.957042	-1.6
12	../data/Basmati/basmati (6238).jpg	Basmati	107.637722	103.150642	102.179118	-1.9
285	../data/Arborio/Arborio (4018).jpg	Arborio	140.676812	137.430351	137.066959	-1.6
176	../data/Jasmine/Jasmine (14886).jpg	Jasmine	84.571274	83.195762	74.938960	-1.6
170	../data/Jasmine/Jasmine (4625).jpg	Jasmine	146.998120	141.838487	140.251567	-1.5

5 rows × 25 columns

In [ ]:
```
rice_df[features].sample(10)
```

Out[ ]:

	RGB_R_mean	RGB_G_mean	RGB_B_mean	RGB_R_kurtosis	RGB_G_kurtosis	RGB_B_
30	206.346291	181.674498	181.470454	2.296589	2.108693	
84	79.705591	76.360626	71.928974	-1.435648	-1.423064	-
73	160.282399	147.255802	147.378873	-1.523270	-1.520287	-
56	62.736797	57.563018	57.973902	-0.692484	-0.691393	-
10	185.757198	166.405085	164.311479	-0.466210	-0.495345	-
97	57.822295	51.400658	51.554230	-0.436550	-0.411306	-
258	192.961906	187.892525	188.567875	1.299240	1.262270	
291	142.747228	139.021925	139.958190	-1.772636	-1.772052	-
109	145.336950	137.756710	136.498953	-1.744669	-1.745249	-
83	56.852805	53.646200	53.385164	-0.639938	-0.631511	-

10 rows × 22 columns

In [ ]:

```python
scaler = StandardScaler()

rice_df[features] = scaler.fit_transform(rice_df[features])

rice_df.sample(10)
```

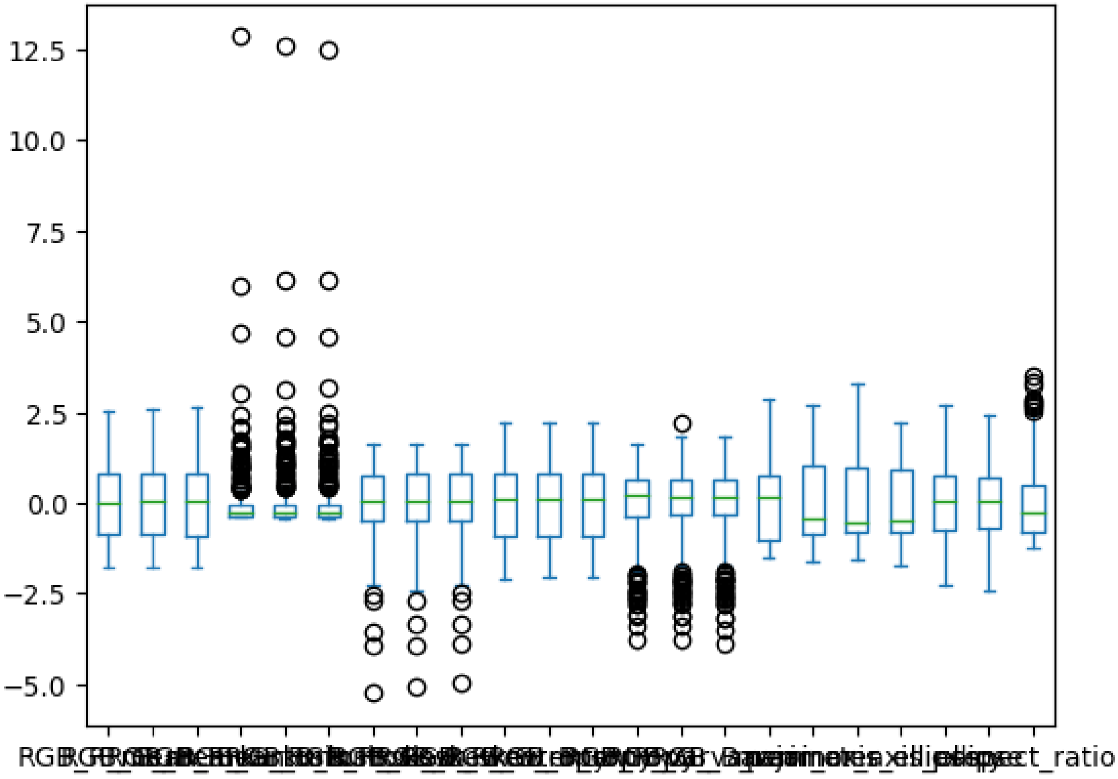
Out[ ]:

	path	label	RGB_R_mean	RGB_G_mean	RGB_B_mean	RGB_R_kur
78	../data/Basmati/basmati (610).jpg	Basmati	1.923439	1.551334	1.599516	1.14
89	../data/Basmati/basmati (8616).jpg	Basmati	-1.391266	-1.406023	-1.383900	-0.13
299	../data/Arborio/Arborio (13898).jpg	Arborio	0.511908	0.721750	0.759949	-0.24
213	../data/Arborio/Arborio (6948).jpg	Arborio	0.949583	0.931764	0.951793	-0.01
298	../data/Arborio/Arborio (13824).jpg	Arborio	0.334747	0.441158	0.452099	-0.28
77	../data/Basmati/basmati (4060).jpg	Basmati	0.029006	-0.169809	-0.139377	-0.38
52	../data/Basmati/basmati (9959).jpg	Basmati	0.977773	0.614243	0.642039	-0.16
265	../data/Arborio/Arborio (12964).jpg	Arborio	1.761888	1.805848	1.803488	1.70
208	../data/Arborio/Arborio (8852).jpg	Arborio	0.421384	0.565888	0.590545	-0.33
236	../data/Arborio/Arborio (5557).jpg	Arborio	0.952385	1.012284	1.017353	0.03

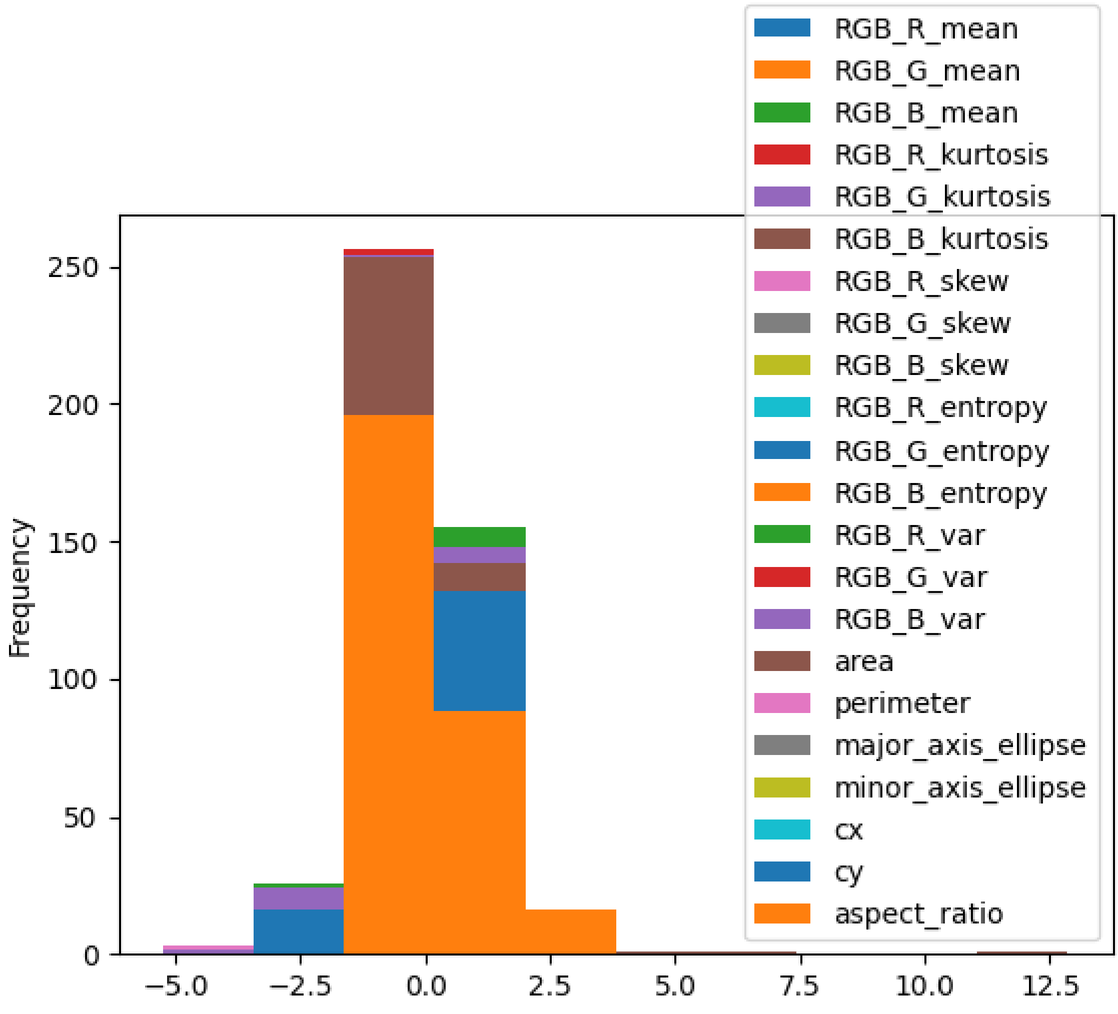
10 rows × 25 columns

In [ ]:

```python
ax = rice_df[features].plot.box()
```



In [ ]:
```
ax = rice_df[features].plot.hist()
```



In [ ]:
```
# sns.pairplot(rice_df[features])
```

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

In [ ]:
```
pca = PCA(n_components=2)
rice_pca = pca.fit_transform(rice_df[features])
pca.explained_variance_ratio_.cumsum()
```

Out[ ]:
```
array([0.52429976, 0.66296196])
```

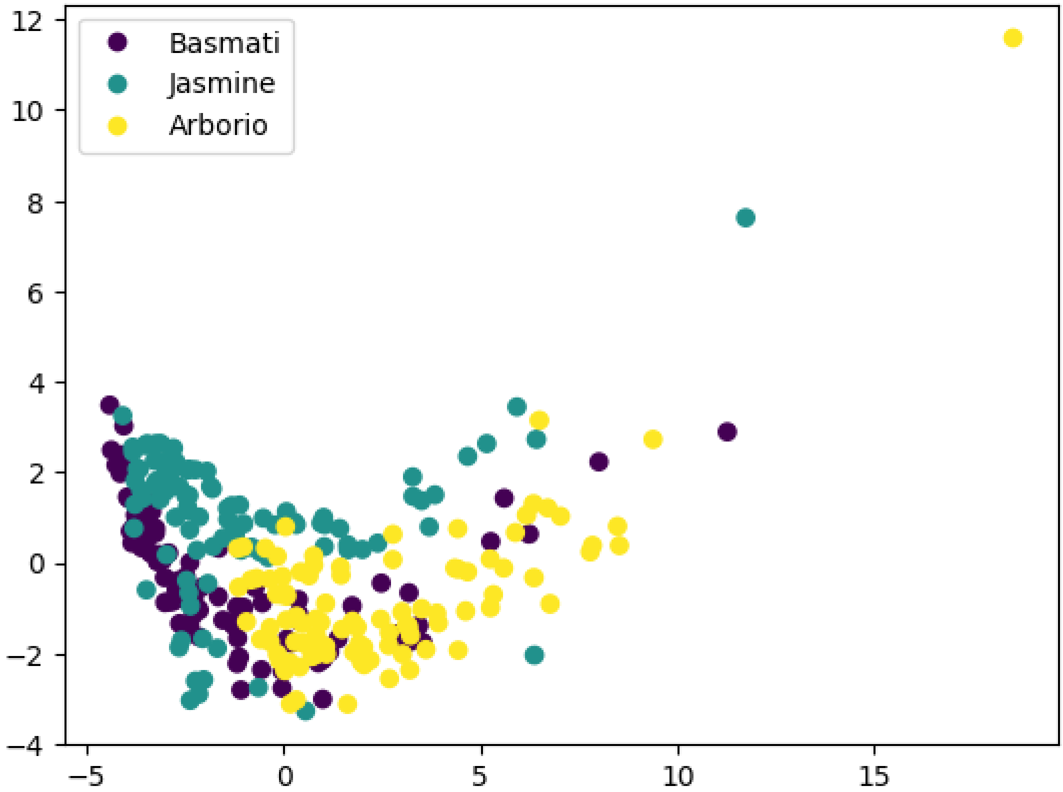
In [ ]:
```
rice_df.shape
```

Out[ ]:
```
(300, 25)
```

In [ ]:
```
labels = list(rice_df["label"].unique())
color =list(map(lambda x: labels.index(x) ,list(rice_df["label"])  ))
```

In [ ]:
```
fig = plt.scatter(rice_pca[:,0], rice_pca[:,1],c=color, )
plt.legend(handles=fig.legend_elements()[0], labels=list(labels))
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7fb5883879d0>
```



In [ ]:
```
exp_var = []
for i in range(1,len(features)+1):
    pca = PCA(n_components=i)
    rice_pca = pca.fit_transform(rice_df[features])
    if pca.explained_variance_ratio_.cumsum()[-1] >=0.99:
        exp_var.append(i)


print(f"The First {exp_var[0]} explain 99% variance.")
```

```
The First 8 explain 99% variance.
```

In [ ]:
```
cv = RepeatedKFold(n_splits=5, n_repeats=3)
```

In [ ]:
```
param_grid = dict(n_neighbors=range(1, 11))
knn = KNeighborsClassifier()
grid = GridSearchCV(knn, cv=cv, param_grid=param_grid )
grid.fit(rice_df[features], list(rice_df["label"]))
grid.best_params_
```

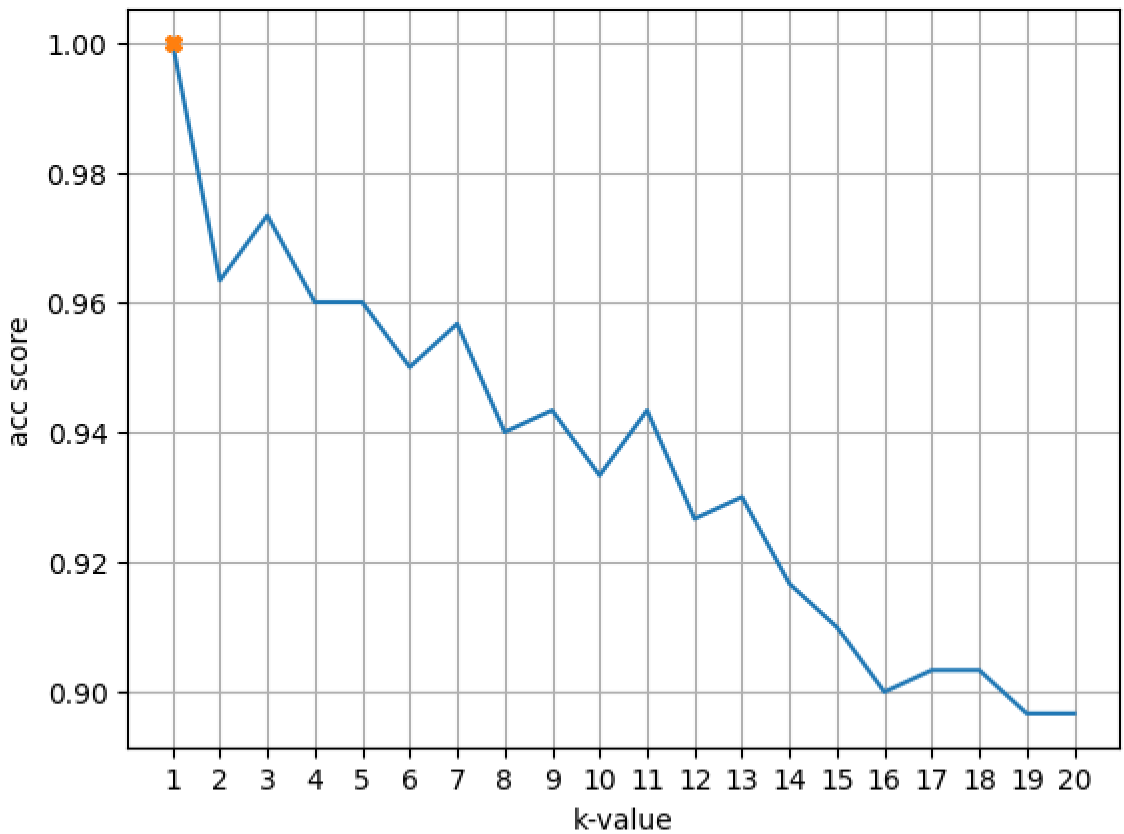
Out[ ]:
```
{'n_neighbors': 3}
```

In [ ]:
```
# Trying different k-values (0-20)
accuracies = []
for k in range(1,21):
    train_knn = KNeighborsClassifier(n_neighbors=k) #define the model
    train_knn.fit(rice_df[features], list(rice_df["label"])) #train/fit m
    predictions_knn = train_knn.predict(rice_df[features]) #predictions

    #print(metrics.confusion_matrix(testing_labels, predictions_knn)) #pr
    acc = metrics.accuracy_score(list(rice_df["label"]), predictions_knn)
    #print("accuracy:",acc) #print accuracy score
    accuracies.append(acc)

plt.plot(range(1,21),accuracies,)
plt.ylabel('acc score')
plt.xlabel('k-value')
plt.xticks(range(1,21))
plt.grid()
mark = accuracies.index(max(accuracies))
plt.plot(mark + 1, accuracies[mark],marker="X")
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x7fb5884df2e0>]
```



In [ ]:
```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 110, num =
# Number of features to consider at every split
max_features = ['sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(1, 10, num = 2)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
param_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

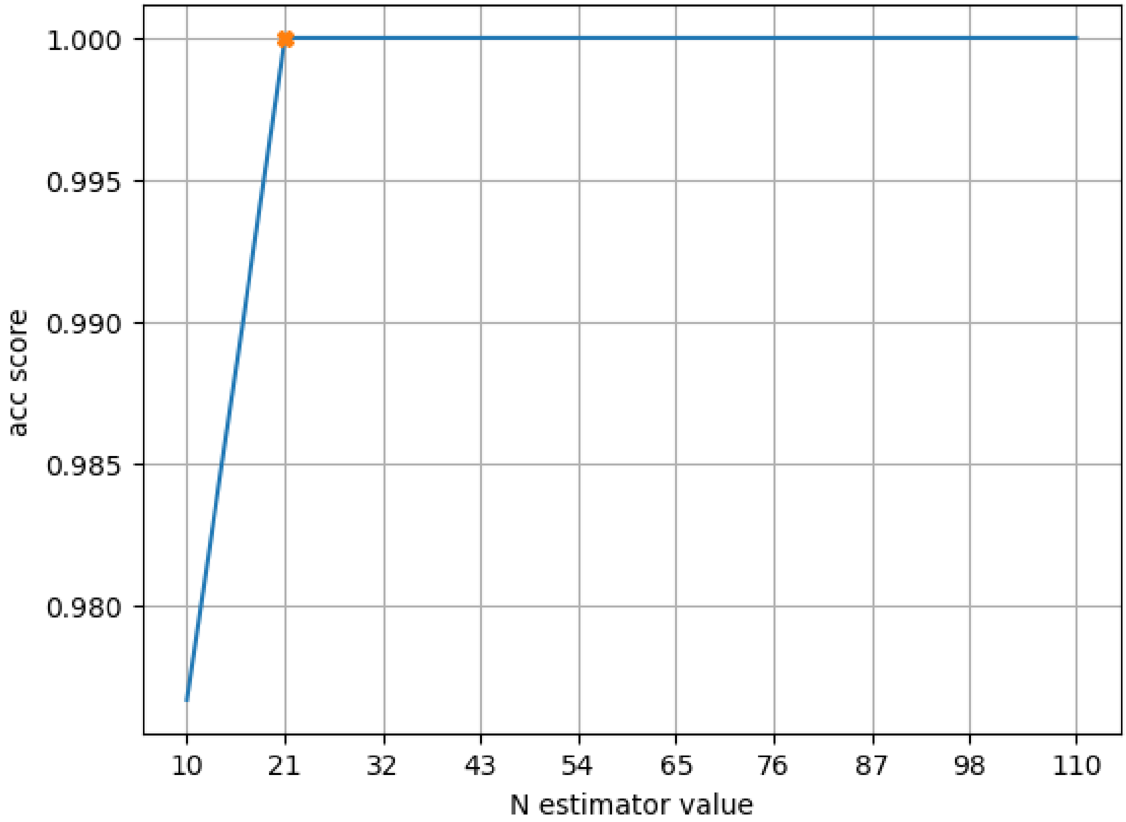
In [ ]:
```
rfc = RandomForestClassifier()
grid = GridSearchCV(rfc, cv=cv, param_grid=param_grid )
grid.fit(rice_df[features], list(rice_df["label"]))
grid.best_params_
```

Out[ ]:
```
{'bootstrap': True,
 'max_depth': 10,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 5,
 'n_estimators': 98}
```

In [ ]:
```
accuracies = []
for k in [int(x) for x in np.linspace(start = 1, stop = 110, num = 10)]:
    train_rf = RandomForestClassifier(n_estimators=k, max_features="sqrt"
    train_rf.fit(rice_df[features], list(rice_df["label"])) #train/fit mo
    predictions_rf = train_rf.predict(rice_df[features]) #predictions

    #print(metrics.confusion_matrix(testing_labels, predictions_knn)) #pr
    acc = metrics.accuracy_score(list(rice_df["label"]), predictions_rf)
    #print("accuracy:",acc) #print accuracy score
    accuracies.append(acc)
x = [int(x) for x in np.linspace(start = 10, stop = 110, num = 10)]
plt.plot(x,accuracies)
plt.ylabel('acc score')
plt.xlabel('N estimator value')
plt.xticks([int(x) for x in np.linspace(start = 10, stop = 110, num = 10)
plt.grid()
mark = accuracies.index(max(accuracies))
plt.plot(x[mark], accuracies[mark],marker="X")
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x7fb5885892a0>]
```



{'bootstrap': True, 'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}

In [ ]:
```
param_grid = {
    'activation': ["relu", "logistic"],
    'solver': ["sgd", "adam"],
    'validation_fraction': [0.1, 0.5],
}
```

In [ ]:
```
mlp = MLPClassifier(hidden_layer_sizes=1, early_stopping=True)
grid = GridSearchCV(mlp, cv=cv, param_grid=param_grid)
grid.fit(rice_df[features], list(rice_df["label"]))
grid.best_params_
```

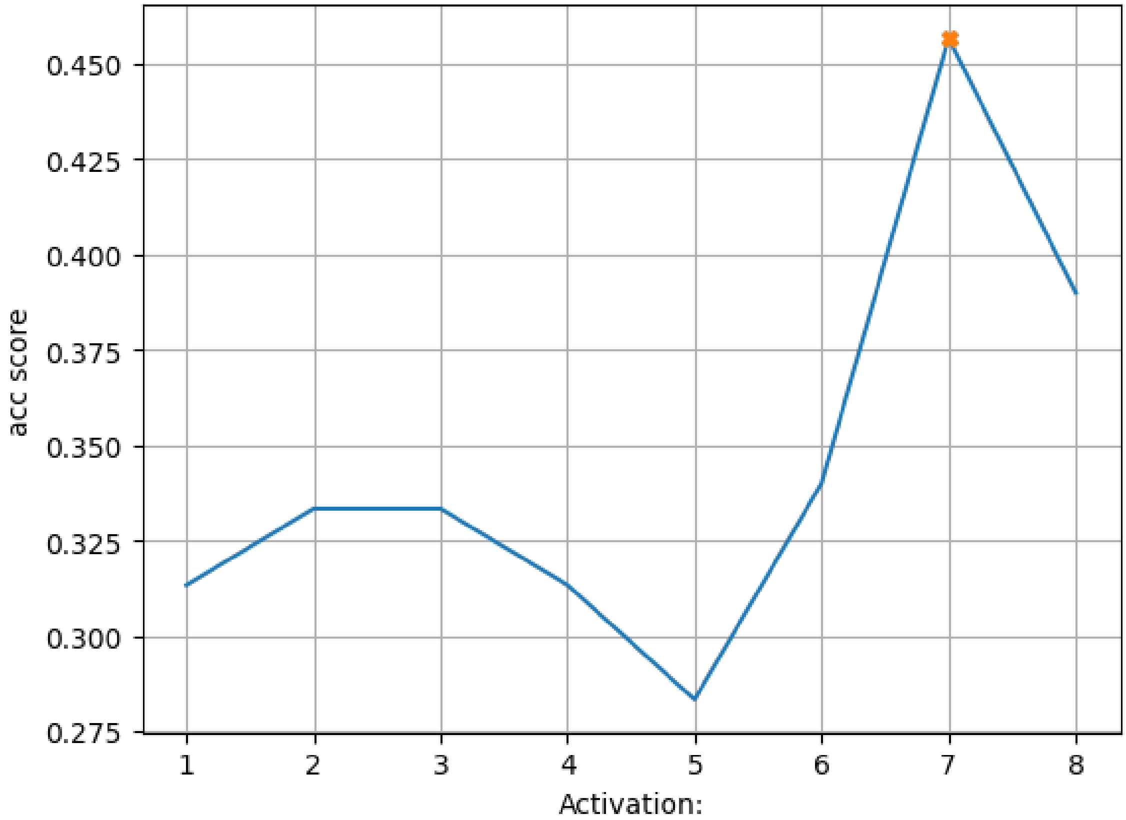
Out[ ]: {'activation': 'relu', 'solver': 'adam', 'validation_fraction': 0.1}

In [ ]:
```
accuracies = []
for activation in param_grid["activation"]:
    for solver in param_grid["solver"]:
        for validation_fraction in param_grid["validation_fraction"]:
            train_mlp = MLPClassifier(hidden_layer_sizes=1, early_stoppin
            train_mlp.fit(rice_df[features], list(rice_df["label"])) #tra
            predictions_mlp = train_mlp.predict(rice_df[features]) #predi

            #print(metrics.confusion_matrix(testing_labels, predictions_k
            acc = metrics.accuracy_score(list(rice_df["label"]), predicti
            #print("accuracy:",acc) #print accuracy score
            accuracies.append(acc)
x = range(1,len(accuracies)+1)
plt.plot(x,accuracies)
plt.ylabel('acc score')
plt.xlabel(f'Activation:')
```

```
plt.xticks(range(1,21))
plt.grid()
mark = accuracies.index(max(accuracies))
plt.plot(mark + 1, accuracies[mark],marker="X")
```

Out[ ]: [<matplotlib.lines.Line2D at 0x7fb580d586d0>]



In [ ]: