

Name : Pankaj Chawla

Roll : 001811001052

Subject : Machine Learning Lab
Assignment – 2

Department : I.T (4th year, 1st
semester)

```

# BREAST CANCER DATASET
# Random Forest Classifier(Without Tuning)[70-30 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name

X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



Confusion Matrix:

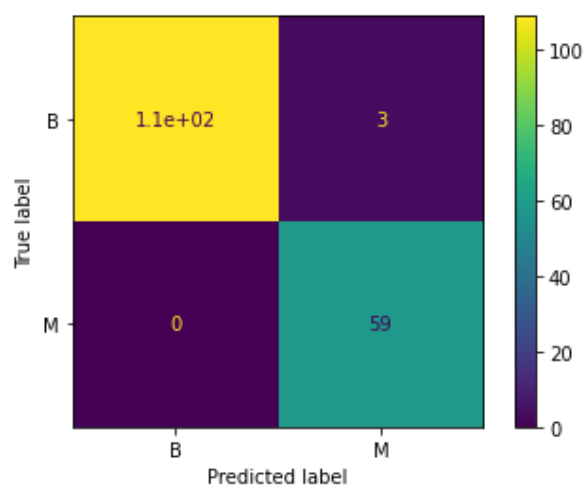
```
[[109  3]
 [  0 59]]
```

Performance Evaluation

	precision	recall	f1-score	support
B	1.00	0.97	0.99	112
M	0.95	1.00	0.98	59
accuracy			0.98	171
macro avg	0.98	0.99	0.98	171
weighted avg	0.98	0.98	0.98	171

Accuracy:

0.9824561403508771



BREAST CANCER DATASET

Random Forest Classifier(Without Tuning)[60-40 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wdbc.data",header=None)
```

```
col_name = ['1', 'Class', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17',
            '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32']
```

```
df.columns = col_name
```

```
df.columns = col_name
```

```
X = df.drop(['1', 'Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,randome
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier(n_estimators=20, random_state=0)
```

```
classifier.fit(X_train,y_train)
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Performance Evaluation")
```

```
print(classification_report(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Accuracy:")
```

```
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import plot_confusion_matrix
```

```
plot_confusion_matrix(classifier, X_test, y_test)
```

```
plt.show()
```

Confusion Matrix:

```
[[145  4]
 [ 1 78]]
```

Performance Evaluation

	precision	recall	f1-score	support
B	0.99	0.97	0.98	149
M	0.95	0.99	0.97	79
accuracy			0.98	228
macro avg	0.97	0.98	0.98	228
weighted avg	0.98	0.98	0.98	228

Accuracy:

0.9780701754385965



BREAST CANCER DATASET

Random Forest Classifier(Without Tuning)[50-50 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wdbc.data",header=None)
```

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

```
df.columns = col_name
```

```
X = df.drop(['1','Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

```
[[179  4]
 [  0 102]]
```

Performance Evaluation

	precision	recall	f1-score	support
B	1.00	0.98	0.99	183
M	0.96	1.00	0.98	102
accuracy			0.99	285

BREAST CANCER DATASET

Random Forest Classifier(Without Tuning)[40-60 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wdbc.data",header=None)
```

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

```
df.columns = col_name
```

```
X = df.drop(['1','Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=0)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Classification

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

```
[[219  8]
 [ 2 113]]
```

```
-----
-----
```

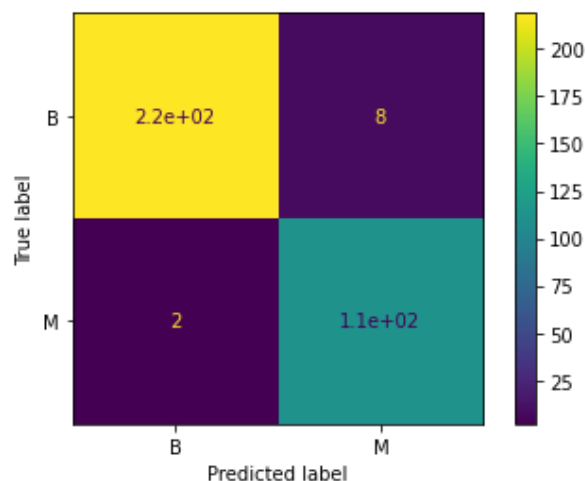
Performance Evaluation

	precision	recall	f1-score	support
B	0.99	0.96	0.98	227
M	0.93	0.98	0.96	115
accuracy			0.97	342
macro avg	0.96	0.97	0.97	342
weighted avg	0.97	0.97	0.97	342

```
-----
-----
```

Accuracy:

0.9707602339181286



BREAST CANCER DATASET

Random Forest Classifier(Without Tuning)[30-70 split]


```
import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name

X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

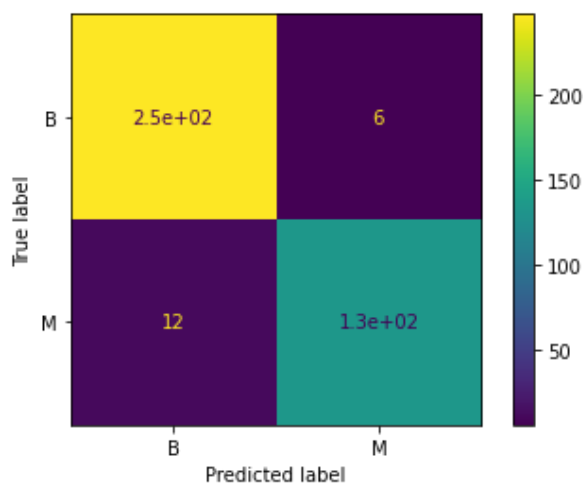
```
[[247  6]
 [ 12 134]]
```

Performance Evaluation

	precision	recall	f1-score	support
B	0.95	0.98	0.96	253
M	0.96	0.92	0.94	146
accuracy			0.95	399
macro avg	0.96	0.95	0.95	399
weighted avg	0.95	0.95	0.95	399

Accuracy:

0.9548872180451128



BREAST CANCER DATASET

Random Forest Classifier(With Tuning)[70-30 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wdbc.data", header=None)
```

```
col_name = ['1', 'Class', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17',
            '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}
pprint(random_grid)

#####
```

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done 37 tasks      | elapsed: 57.6s
[Parallel(n_jobs=-1)]: Done 158 tasks    | elapsed: 3.9min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 7.3min finished
```

Confusion Matrix:

```
[[109  3]
 [  0 59]]
```

BREAST CANCER DATASET

Random Forest Classifier(With Tuning)[60-40 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wdbc.data", header=None)
```

```
col_name = ['1', 'Class', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17',
            '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier()
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
```

```
# Look at parameters used by our current forest
```

```
print('Parameters currently in use:\n')
```

```
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```
from sklearn.model_selection import RandomizedSearchCV
```

```
# Number of trees in random forest
```

```
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
```

```
# Number of features to consider at every split
```

```
max_features = ['auto', 'sqrt']
```

```
# Maximum number of levels in tree
```

```
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
```

```
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
```

```
random_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}
```

```
pprint(random_grid)
```

```
#####
```

```
# Use the random grid to search for best hyperparameters
```

```
# First create the base model to tune
```

```
classifier = RandomForestClassifier()
```

```
# Random search of parameters, using 3 fold cross validation,
```

```
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done 37 tasks      | elapsed: 53.9s
[Parallel(n_jobs=-1)]: Done 158 tasks    | elapsed: 3.7min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 7.0min finished
```

BREAST CANCER DATASET

Random Forest Classifier(With Tuning)[50-50 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wdbc.data",header=None)
```

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando
```

Feature Scaling


```

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}
pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

```

```
rf_random.predict(X_test, y_test)
```

```
y_pred = rf_random.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Performance Evaluation")
```

```
print(classification_report(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Accuracy:")
```

```
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import plot_confusion_matrix
```

```
plot_confusion_matrix(rf_random, X_test, y_test)
```

```
plt.show()
```

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Fitting 3 folds for each of 100 candidates. totalling 300 fits

BREAST CANCER DATASET

Random Forest Classifier(With Tuning)[40-60 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("wdbc.data",header=None)

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

df.columns = col_name

X = df.drop(['Class'], axis=1)

y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=42)

Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

```

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}
pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'best']}
```

BREAST CANCER DATASET

Random Forest Classifier(With Tuning)[30-70 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("wdbc.data",header=None)

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

df.columns = col_name

X = df.drop(['Class'], axis=1)

y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=42)

Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

Classification

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier()
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
```

```
# Look at parameters used by our current forest
```

```
print('Parameters currently in use:\n')
```

```
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```
from sklearn.model_selection import RandomizedSearchCV
```

```
# Number of trees in random forest
```

```
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
```

```
# Number of features to consider at every split
```

```
max_features = ['auto', 'sqrt']
```

```
# Maximum number of levels in tree
```

```
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
```

```
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
```

```
random_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}
```

```
pprint(random_grid)
```

```
#####
```

```
# Use the random grid to search for best hyperparameters
```

```
# First create the base model to tune
```

```
classifier = RandomForestClassifier()
```

```
# Random search of parameters, using 3 fold cross validation,
```

```
# search across 100 different combinations, and use all available cores
```

```
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
```

```
# Fit the random search model
```

```
rf_random.fit(X_train, y_train)
```

```
y_pred = rf_random.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, y_pred))
```

```
print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```


Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
```

#####

```
{ 'bootstrap': [True, False],
```

BREAST CANCER DATASET

Multi Layer Perceptron(Without Tuning)[70-30 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("wdbc.data",header=None)

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

df.columns = col_name

X = df.drop(['1','Class'], axis=1)

y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand

Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

Classification using MLP

from sklearn.neural_network import MLPClassifier

```
classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
```

```
% self.max_iter, ConvergenceWarning)
```

```
Confusion Matrix:
```

```
[[110  2]
 [ 2  57]]
```

```
-----
-----
```

```
Performance Evaluation
```

	precision	recall	f1-score	support
B	0.98	0.98	0.98	112
M	0.97	0.97	0.97	59
accuracy	0.98	0.98	0.98	171

```
# BREAST CANCER DATASET
```

```
# Multi Layer Perceptron(Without Tuning)[60-40 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("wdbc.data",header=None)
```

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

```
df.columns = col_name
```

```
X = df.drop(['1','Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification using MLP
```

```
from sklearn.neural_network import MLPClassifier
```

```
classifier = MLPClassifier()
```

```
classifier.fit(X_train,y_train)
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[147   2]
 [  2  77]]
-----

# BREAST CANCER DATASET
# Multi Layer Perceptron(Without Tuning)[50-50 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name

X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,random_state=0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
```

```
% self.max_iter, ConvergenceWarning)
```

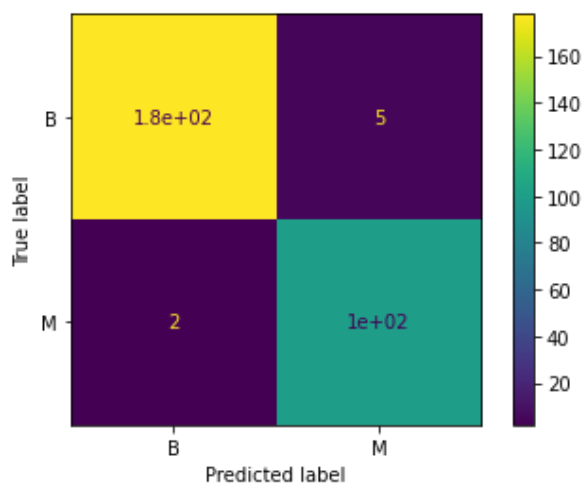
```
Confusion Matrix:
```

```
[[178  5]
 [ 2 100]]
```

```
Performance Evaluation
```

	precision	recall	f1-score	support
B	0.99	0.97	0.98	183
M	0.95	0.98	0.97	102
accuracy			0.98	285
macro avg	0.97	0.98	0.97	285
weighted avg	0.98	0.98	0.98	285

```
Accuracy:
0.9754385964912281
```



```
# BREAST CANCER DATASET
# Multi Layer Perceptron(Without Tuning)[40-60 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name

X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)
```

Confusion Matrix:

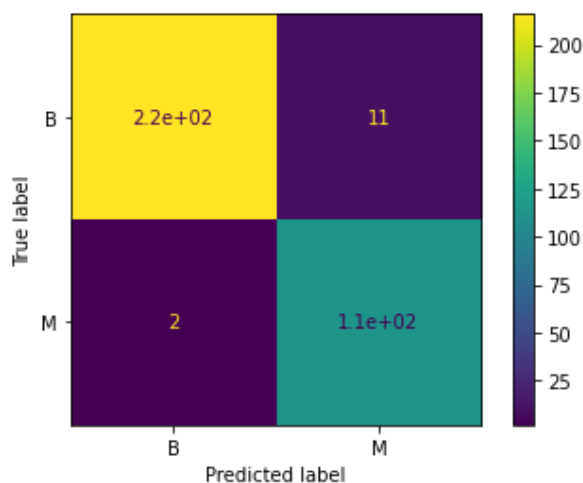
```
[[216  11]
 [  2 113]]
```

Performance Evaluation

	precision	recall	f1-score	support
B	0.99	0.95	0.97	227
M	0.91	0.98	0.95	115
accuracy			0.96	342
macro avg	0.95	0.97	0.96	342
weighted avg	0.96	0.96	0.96	342

Accuracy:

0.9619883040935673



BREAST CANCER DATASET

Multi Layer Perceptron(Without Tuning)[30-70 split]

```
import pandas as pd
import numpy as np
```

Dataset Description


```

# dataset preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name

X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()

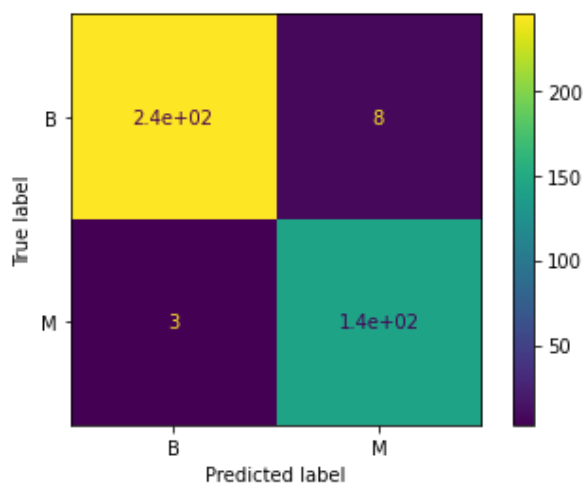
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
    % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[245   8]
 [  3 143]]
```

```
-----
-----
Performance Evaluation
```

	precision	recall	f1-score	support
B	0.99	0.97	0.98	253
M	0.95	0.98	0.96	146
accuracy			0.97	399
macro avg	0.97	0.97	0.97	399
weighted avg	0.97	0.97	0.97	399

```
-----
-----
Accuracy:
0.9724310776942355
```



```
# BREAST CANCER DATASET
# Multi Layer Perceptron(With Tuning)[70-30 split]
```

```
import pandas as pd
import numpy as np
```

```
# Dataset Preparation
df = pd.read_csv("wdbc.data", header=None)
```

```
col_name = ['1', 'Class', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17',
            '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32']
```

```
df.columns = col_name
```

```
df.columns = col_name
```

```
X = df.drop(['1', 'Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.neural_network import MLPClassifier
```

```
classifier = MLPClassifier(max_iter=100)
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
```

```
# Look at parameters used by our current forest
```

```
print('Parameters currently in use:\n')
```

```
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```
parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
```

```
pprint(parameter_space)
```

```
#####
```

```
from sklearn.model_selection import GridSearchCV
```

```
# Use the random grid to search for best hyperparameters
```

```
# First create the base model to tune
```

```
classifier = MLPClassifier(max_iter=100)
```

```
# Random search of parameters, using 3 fold cross validation,
```

```
# search across 100 different combinations, and use all available cores
```

```
rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
```

```
rf_random.fit(X_train, y_train)
```

```
y_pred = rf_random.predict(X_test)
```

```
y_pred = rf_random.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Performance Evaluation")
```

```
print(classification_report(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Accuracy:")
```

```
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import plot_confusion_matrix
```

```
plot_confusion_matrix(rf_random, X_test, y_test)
```

```
plt.show()
```

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05]}
```

BREAST CANCER DATASET

Multi Layer Perceptron(With Tuning)[60-40 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wdbc.data",header=None)
```

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

```
df.columns = col_name
```

```
X = df.drop(['1','Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
```

```

X_test = sc.transform(X_test)

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant', 'adaptive'],
}
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")

```

```
print("-----")
```

```
print("Accuracy:")
```

```
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import plot_confusion_matrix
```

```
plot_confusion_matrix(rf_random, X_test, y_test)
```

```
plt.show()
```

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
```

BREAST CANCER DATASET

Multi Layer Perceptron(With Tuning)[50-50 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("wdbc.data",header=None)

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

df.columns = col_name

X = df.drop(['1','Class'], axis=1)

y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,random_state=42)

Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

Classification

from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier(max_iter=100)

#####

Showing all the parameters

from pprint import pprint

Look at parameters used by our current forest

print('Parameters currently in use:\n')

pprint(classifier.get_params())

#####

Creating a set of important sample features


```

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()

```

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)
```

Confusion Matrix:

```
[[177  6]
 [  0 102]]
```

Performance Evaluation

	precision	recall	f1-score	support
B	1.00	0.97	0.98	183
M	0.94	1.00	0.97	102
accuracy			0.98	285
macro avg	0.97	0.98	0.98	285
weighted avg	0.98	0.98	0.98	285

Accuracy:

0.9789473684210527



BREAST CANCER DATASET

Multi Layer Perceptron(With Tuning)[40-60 split]

```
import pandas as pd
import numpy as np
```

```

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name

X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune

```

```
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[220   7]
 [  2 113]]
```

BREAST CANCER DATASET

Multi Layer Perceptron(With Tuning)[30-70 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("wdbc.data",header=None)

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

df.columns = col_name

X = df.drop(['1','Class'], axis=1)

y = df['Class']

from sklearn.model_selection import train_test_split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2, random
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.neural_network import MLPClassifier
```

```
classifier = MLPClassifier(max_iter=100)
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
```

```
# Look at parameters used by our current forest
```

```
print('Parameters currently in use:\n')
```

```
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```
parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant', 'adaptive'],
}
```

```
pprint(parameter_space)
```

```
#####
```

```
from sklearn.model_selection import GridSearchCV
```

```
# Use the random grid to search for best hyperparameters
```

```
# First create the base model to tune
```

```
classifier = MLPClassifier(max_iter=100)
```

```
# Random search of parameters, using 3 fold cross validation,
```

```
# search across 100 different combinations, and use all available cores
```

```
rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
```

```
rf_random.fit(X_train, y_train)
```

```
y_pred = rf_random.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5}
```

```
#####
```

```
'solver': 'adam'
```

```
# BREAST CANCER DATASET
```

```
# SVM(Without Tuning)[70-30 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("wdbc.data",header=None)
```

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

```
df.columns = col_name
```

```
X = df.drop(['1','Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.svm import SVC
```

```
classifier = SVC()
```



```
-----,
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

```
[[109   3]
 [  0  59]]
```

Performance Evaluation

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

B	1.00	0.97	0.99	112
---	------	------	------	-----

BREAST CANCER DATASET

SVM(Without Tuning)[60-40 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
'20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name

X = df.drop(['1','Class'], axis=1)

y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=42)

Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

Classification

from sklearn.svm import SVC

classifier = SVC()

classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")

print(confusion_matrix(y_test, y_pred))

print("-----")

```
print("-----")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

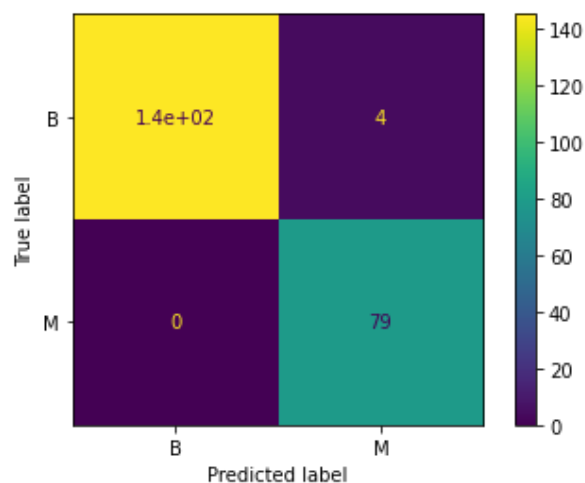
```
[[145  4]
 [ 0 79]]
```

Performance Evaluation

	precision	recall	f1-score	support
B	1.00	0.97	0.99	149
M	0.95	1.00	0.98	79
accuracy			0.98	228
macro avg	0.98	0.99	0.98	228
weighted avg	0.98	0.98	0.98	228

Accuracy:

0.9824561403508771



```
# SVM(Without Tuning)[50-50 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name

X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,random_state=0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

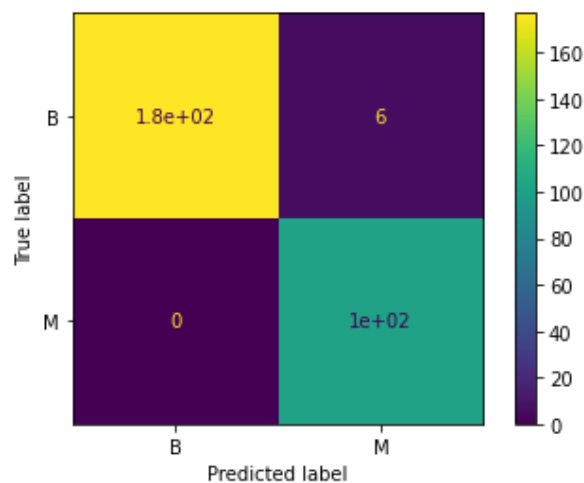
```
[[177  6]
 [  0 102]]
```

Performance Evaluation

	precision	recall	f1-score	support
B	1.00	0.97	0.98	183
M	0.94	1.00	0.97	102
accuracy			0.98	285
macro avg	0.97	0.98	0.98	285
weighted avg	0.98	0.98	0.98	285

Accuracy:

0.9789473684210527



```
# BREAST CANCER DATASET
# SVM(Without Tuning)[40-60 split]
```

```
import pandas as pd
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("wdbc.data", header=None)
```

```
col_name = ['1', 'Class', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17']
```

```
, '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32']

df.columns = col_name

X = df.drop(['1', 'Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.4, test_size=0.6, random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

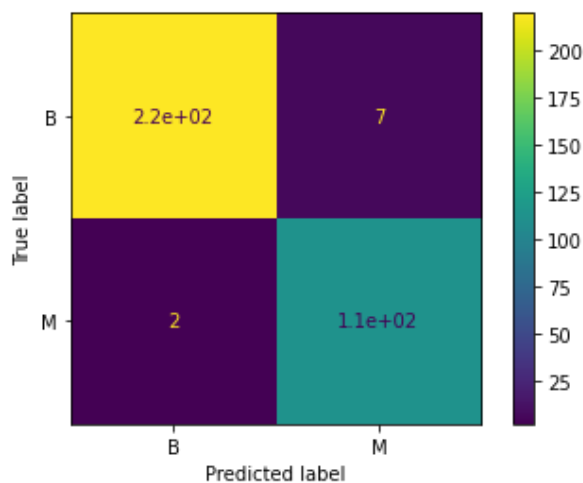
```
[[220  7]
 [ 2 113]]
```

Performance Evaluation

	precision	recall	f1-score	support
B	0.99	0.97	0.98	227
M	0.94	0.98	0.96	115
accuracy			0.97	342
macro avg	0.97	0.98	0.97	342
weighted avg	0.97	0.97	0.97	342

Accuracy:

0.9736842105263158



BREAST CANCER DATASET

SVM(Without Tuning)[30-70 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("wdbc.data", header=None)

col_name = ['1', 'Class', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32']

df.columns = col_name

X = df.drop(['1', 'Class'], axis=1)

y = df['Class']

from sklearn.model_selection import train_test_split

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```


Confusion Matrix:

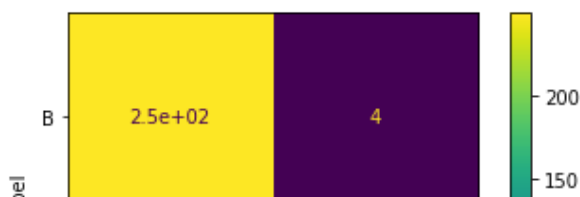
```
[[249  4]
 [ 10 136]]
```

Performance Evaluation

	precision	recall	f1-score	support
B	0.96	0.98	0.97	253
M	0.97	0.93	0.95	146
accuracy			0.96	399
macro avg	0.97	0.96	0.96	399
weighted avg	0.97	0.96	0.96	399

Accuracy:

0.9649122807017544



BREAST CANCER DATASET

SVM(With Tuning)[70-30 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
, '20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name

X = df.drop(['1','Class'], axis=1)

y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando

Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

```

# Classification
from sklearn.svm import SVC

classifier = SVC()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel': ['rbf', 'poly'],

pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
```

```
# BREAST CANCER DATASET
```

```
# SVM(With Tuning)[60-40 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("wdbc.data",header=None)
```

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

```
df.columns = col_name
```

```
X = df.drop(['1','Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.svm import SVC
```

```
classifier = SVC()
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
```

```
# Look at parameters used by our current forest
```

```
print('Parameters currently in use:\n')
```

```
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',
```

```
pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
```

Fitting 5 folds for each of 48 candidates, totalling 240 fits

```
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
```

BREAST CANCER DATASET

SVM(With Tuning)[50-50 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wdbc.data",header=None)
```

```
col_name = ['1', 'Class', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17',
            '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32']
```

```
df.columns = col_name
```

```
X = df.drop(['1','Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.svm import SVC
```

```
classifier = SVC()
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
```

```
# Look at parameters used by our current forest
```

```
print('Parameters currently in use:\n')
```

```
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001], 'kernel': ['rbf', 'poly'],
```

```
pprint(param_grid)
```

```
#####
```

```
from sklearn.model_selection import GridSearchCV
```

```
# Use the random grid to search for best hyperparameters
```

```
# First create the base model to tune
```

```
classifier = SVC()
```

```
# Random search of parameters, using 3 fold cross validation,
```

```
# search across 100 different combinations, and use all available cores
```

```
rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
```

```
rf_random.fit(X_train, y_train)
```

```
y_pred = rf_random.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```


Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
```

Fitting 5 folds for each of 48 candidates, totalling 240 fits

```
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1 gamma=1 kernel=rbf
```

BREAST CANCER DATASET

SVM(With Tuning)[40-60 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wdbc.data",header=None)
```

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

```
df.columns = col_name
```

```
X = df.drop(['1','Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```

# Classification
from sklearn.svm import SVC

classifier = SVC()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001], 'kernel': ['rbf', 'poly'],

pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 10}
```

```
# BREAST CANCER DATASET
```

```
# SVM(With Tuning)[30-70 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("wdbc.data",header=None)
```

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

```
df.columns = col_name
```

```
X = df.drop(['1','Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.svm import SVC
```

```
classifier = SVC()
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
```

```
# Look at parameters used by our current forest
```

```
print('Parameters currently in use:\n')
```

```
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',
```

```
               'linear']
```

```
pprint(param_grid)
```

```
#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
```

Fitting 5 folds for each of 48 candidates, totalling 240 fits

```
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
```

#####

```
[CV] C=0.1, gamma=1, kernel=sigmoid .....
```

BREAST CANCER DATASET

Random Forest Classifier(Without Tuning)[70-30 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wdbc.data",header=None)
```

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

```
df.columns = col_name

X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=30)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
```

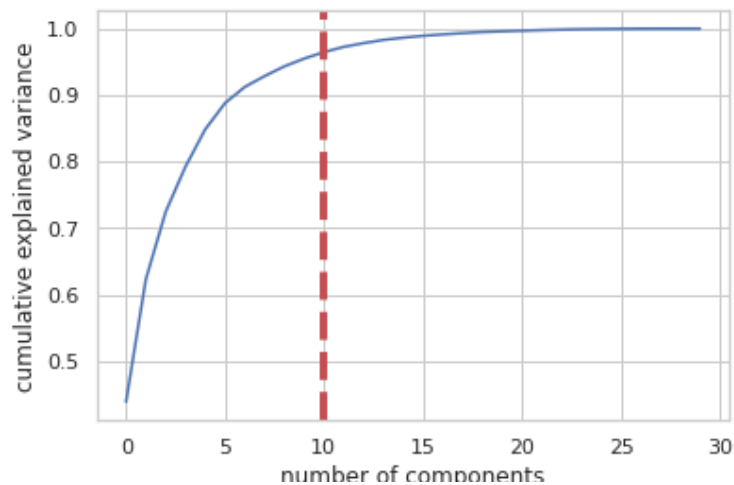
```
print("-----")
```

```
print("Performance Evaluation")  
print(classification_report(y_test, y_pred))
```

```
print("-----")  
print("-----")
```

```
print("Accuracy:")  
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt  
from sklearn.metrics import plot_confusion_matrix  
plot_confusion_matrix(classifier, X_test, y_test)  
plt.show()
```

```
# BREAST CANCER DATASET
# Random Forest Classifier(With Tuning)[70-30 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name

X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=30)
nca_test.fit(X_train)
```

```

sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

```

So we can see that we have 10 important parameters

```

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

```

Classification

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier()
```

```
#####
```

Showing all the parameters

```

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

```

```
#####
```

Creating a set of important sample features

```

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}
pprint(random_grid)

```

```
#####
```

```
.....

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

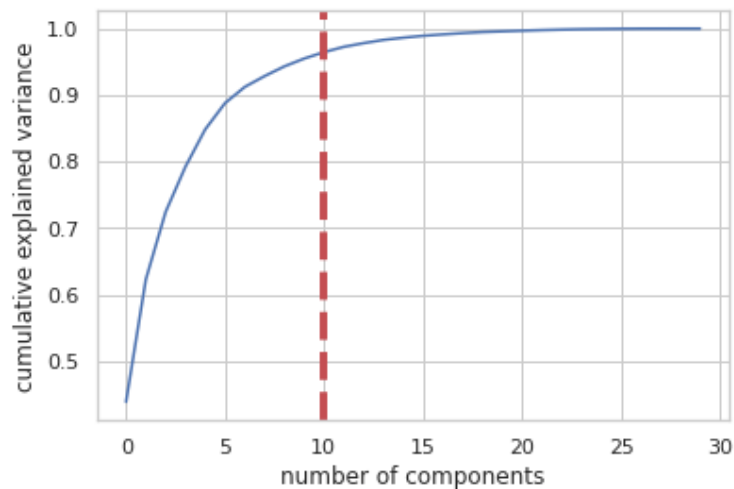
print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```



None

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

BREAST CANCER DATASET

Multi Layer Perceptron(Without Tuning)[70-30 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wdbc.data",header=None)
```

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

```
df.columns = col_name
```

```
X = df.drop(['1','Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=30)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

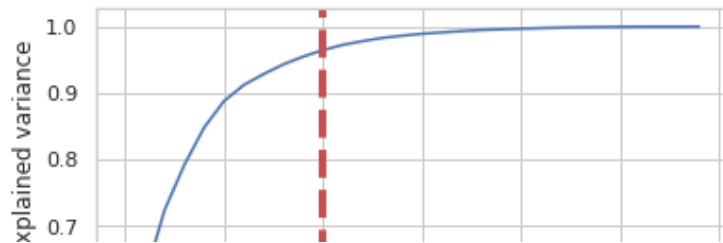
print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



```
# BREAST CANCER DATASET
# Multi Layer Perceptron(With Tuning)[70-30 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wdbc.data",header=None)

col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']

df.columns = col_name

X = df.drop(['1','Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=30)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())
```

```

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

```

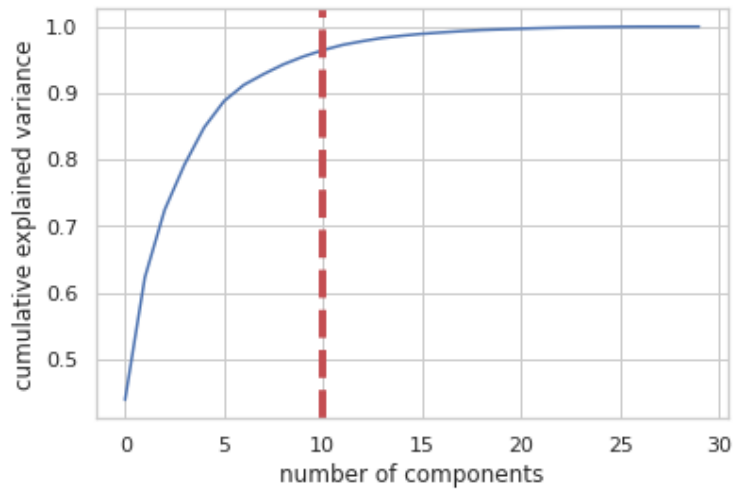


```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```



None

```
# BREAST CANCER DATASET
```

```
# SVM(Without Tuning)[70-30 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("wdbc.data",header=None)
```

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

```
df.columns = col_name
```

```
X = df.drop(['1','Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Finding the important parameters that contribute to most of the variance in the data.
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
from sklearn.decomposition import PCA
```

```
pca_test = PCA(n_components=30)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

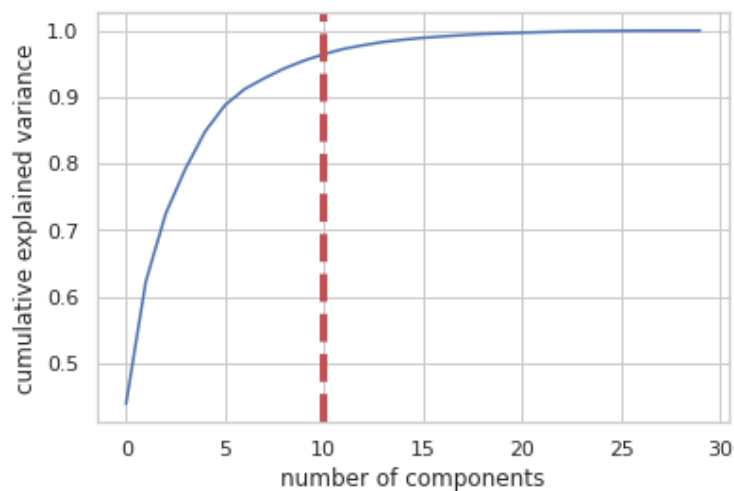
print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



None

Confusion Matrix:

```
[[109  3]
 [  0 59]]
```

Performance Evaluation

	precision	recall	f1-score	support
B	1.00	0.97	0.99	112
M	0.95	1.00	0.98	59
accuracy			0.98	171
macro avg	0.98	0.99	0.98	171
weighted avg	0.98	0.98	0.98	171

Accuracy:

0.9824561403508771



BREAST CANCER DATASET

SVM(With Tuning)[70-30 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wdbc.data",header=None)
```

```
col_name = ['1','Class','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17',
            '20','21','22','23','24','25','26','27','28','29','30','31','32']
```

```
df.columns = col_name

X = df.drop(['1', 'Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=30)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())
```

```
#####
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001], 'kernel': ['rbf', 'poly',

pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

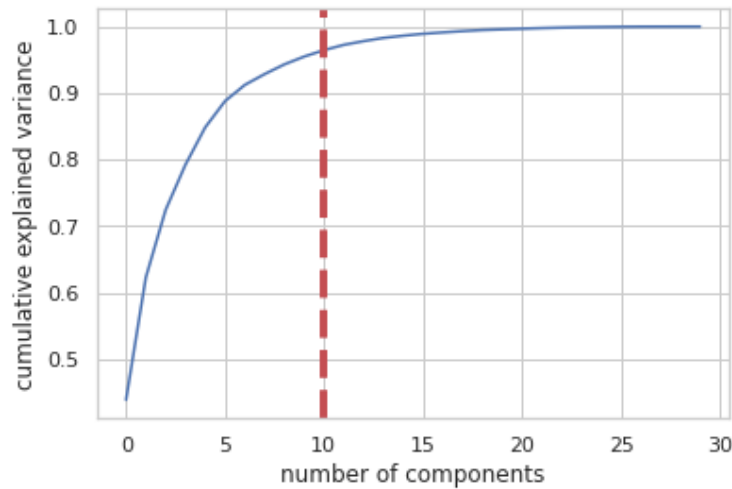
print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```



None

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
```

Fitting 5 folds for each of 48 candidates, totalling 240 fits

```
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
```

```
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
```

```
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf .....
[CV] ..... C=0.1, gamma=0.1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf .....
[CV] ..... C=0.1, gamma=0.1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf .....
[CV] ..... C=0.1, gamma=0.1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf .....
[CV] ..... C=0.1, gamma=0.1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf .....
```




```

# IONOSPHERE DATASET
# Random Forest Classifier(Without Tuning)[70-30 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")

```

```
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



Confusion Matrix:

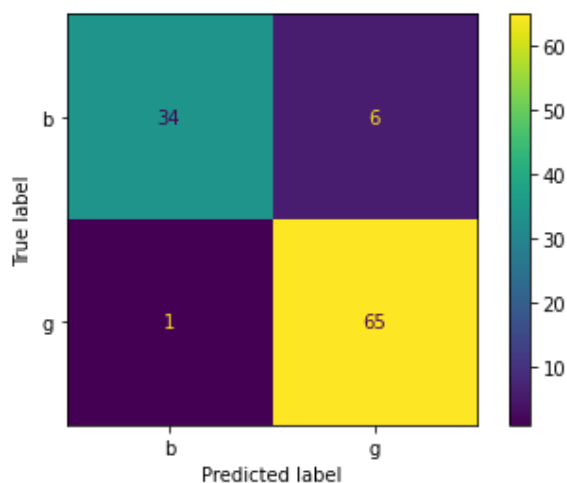
```
[[34  6]
 [ 1 65]]
```

Performance Evaluation

	precision	recall	f1-score	support
b	0.97	0.85	0.91	40
g	0.92	0.98	0.95	66
accuracy			0.93	106
macro avg	0.94	0.92	0.93	106
weighted avg	0.94	0.93	0.93	106

Accuracy:

0.9339622641509434



```
# IONOSPHERE DATASET
```

```
# Random Forest Classifier(Without Tuning)[60-40 split]
```

```
import pandas as pd
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("ionosphere.data", header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

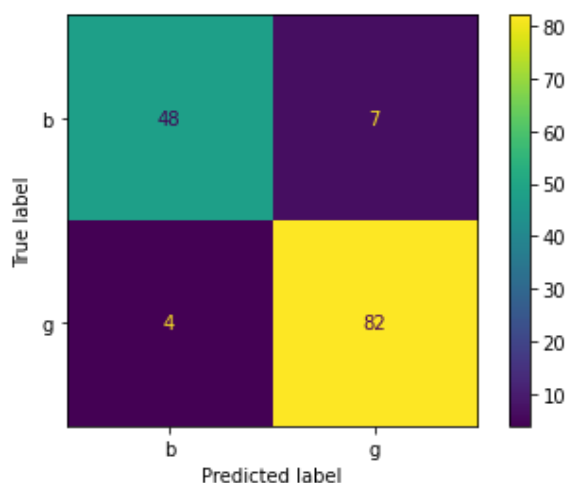
```
[[48  7]
 [ 4 82]]
```

Performance Evaluation

	precision	recall	f1-score	support
b	0.92	0.87	0.90	55
g	0.92	0.95	0.94	86
accuracy			0.92	141
macro avg	0.92	0.91	0.92	141
weighted avg	0.92	0.92	0.92	141

Accuracy:

0.9219858156028369



```
# IONOSPHERE DATASET
```

```
# Random Forest Classifier(Without Tuning)[50-50 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("ionosphere.data", header=None)
```

```
col_name = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', 'Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,random_state=0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

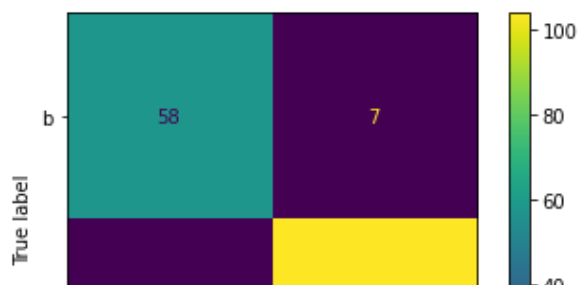
```
[[ 58   7]
 [   7 104]]
```

Performance Evaluation

	precision	recall	f1-score	support
b	0.89	0.89	0.89	65
g	0.94	0.94	0.94	111
accuracy			0.92	176
macro avg	0.91	0.91	0.91	176
weighted avg	0.92	0.92	0.92	176

Accuracy:

0.9204545454545454



IONOSPHERE DATASET

Random Forest Classifier(Without Tuning)[40-60 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

```
[[ 66  10]
 [  7 128]]
```

Performance Evaluation

	precision	recall	f1-score	support
b	0.90	0.87	0.89	76
g	0.93	0.95	0.94	135
accuracy			0.92	211
macro avg	0.92	0.91	0.91	211
weighted avg	0.92	0.92	0.92	211

IONOSPHERE DATASET

Random Forest Classifier(Without Tuning)[30-70 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)

y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=42)

Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

Classification

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)

classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)


```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

```
[[ 76   8]
 [ 13 149]]
```

```
-----
-----
```

```
#####
```

```
# IONOSPHERE DATASET
```

```
# Random Forest Classifier(With Tuning)[70-30 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier()
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
```

```
# Look at parameters used by our current forest
```

```
print('Parameters currently in use:\n')
```

```
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}

pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```



```

# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}

pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)

```

```
plt.show()
```

```

# IONOSPHERE DATASET
# Random Forest Classifier(With Tuning)[50-50 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]

```



```

max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}

pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()

```

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 49.3s

[Parallel(n_jobs=-1)]: Done 158 tasks | elapsed: 3.4min

[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 6.4min finished

Confusion Matrix:

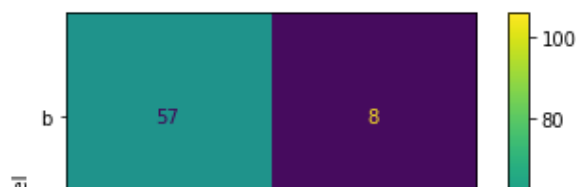
```
[[ 57   8]
 [  5 106]]
```

Performance Evaluation

	precision	recall	f1-score	support
b	0.92	0.88	0.90	65
g	0.93	0.95	0.94	111
accuracy			0.93	176
macro avg	0.92	0.92	0.92	176
weighted avg	0.93	0.93	0.93	176

Accuracy:

0.9261363636363636



IONOSPHERE DATASET

Random Forest Classifier(With Tuning)[40-60 split]

[https://colab.research.google.com/github/stepupgithub/Machine-Learning-Assignments/blob/main/Assignment 2/Ionosphere Dataset.ipynb#pri...](https://colab.research.google.com/github/stepupgithub/Machine-Learning-Assignments/blob/main/Assignment%20Ionosphere%20Dataset.ipynb#pri...) 19/88

```

# 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
random_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}
pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()

```

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 47.1s

[Parallel(n_jobs=-1)]: Done 158 tasks | elapsed: 3.5min

[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 6.9min finished

Confusion Matrix:

```
[[ 66  10]
 [  6 129]]
```

Performance Evaluation

	precision	recall	f1-score	support
b	0.92	0.87	0.89	76
g	0.93	0.96	0.94	135
accuracy			0.92	211
macro avg	0.92	0.91	0.92	211
weighted avg	0.92	0.92	0.92	211

Accuracy:

0.9241706161137441



IONOSPHERE DATASET

Random Forest Classifier(With Tuning)[30-70 split]

```
import pandas as pd
import numpy as np
```

```

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}

pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 48.1s

[Parallel(n_jobs=-1)]: Done 158 tasks | elapsed: 3.4min

[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 6.4min finished

Confusion Matrix:

```
[[ 75   9]
 [  9 153]]
```


Performance Evaluation

	precision	recall	f1-score	support
b	0.89	0.89	0.89	84
g	0.94	0.94	0.94	162
accuracy			0.93	246
macro avg	0.92	0.92	0.92	246
weighted avg	0.93	0.93	0.93	246

#####

IONOSPHERE DATASET

Multi Layer Perceptron(Without Tuning)[70-30 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation


```
df = pd.read_csv("ionosphere.data", header=None)

col_name = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', 'Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=0.3, random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
```

```
% self.max_iter, ConvergenceWarning)
```

```
Confusion Matrix:
```

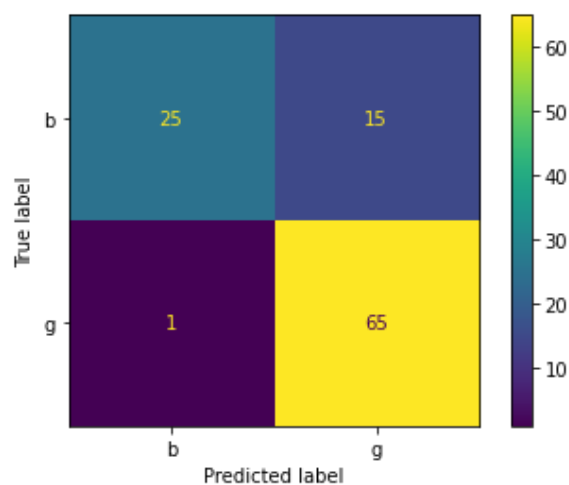
```
[[25 15]
 [ 1 65]]
```

```
-----
Performance Evaluation
```

	precision	recall	f1-score	support
b	0.96	0.62	0.76	40
g	0.81	0.98	0.89	66
accuracy			0.85	106
macro avg	0.89	0.80	0.82	106
weighted avg	0.87	0.85	0.84	106

```
-----
Accuracy:
```

```
0.8490566037735849
```



```
# IONOSPHERE DATASET
```

```
# Multi Layer Perceptron(Without Tuning)[60-40 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
```

```
% self.max_iter, ConvergenceWarning)
```

```
Confusion Matrix:
```

```
[[42 13]
```

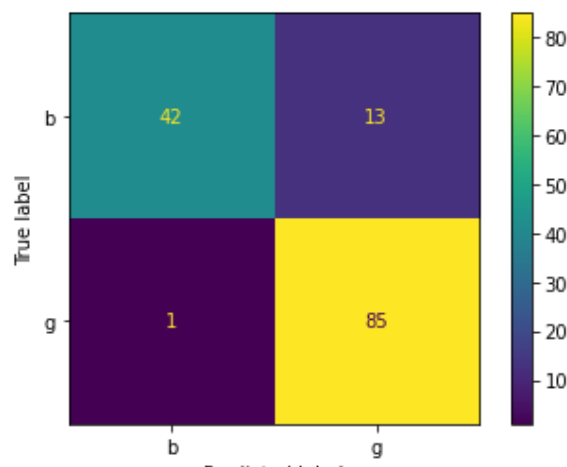
```
 [ 1 85]]
```

```
Performance Evaluation
```

	precision	recall	f1-score	support
b	0.98	0.76	0.86	55
g	0.87	0.99	0.92	86
accuracy			0.90	141
macro avg	0.92	0.88	0.89	141
weighted avg	0.91	0.90	0.90	141

```
Accuracy:
```

```
0.900709219858156
```



```
# IONOSPHERE DATASET
```

```
# Multi Layer Perceptron(Without Tuning)[50-50 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,randome
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification using MLP
```

```
from sklearn.neural_network import MLPClassifier
```

```
classifier = MLPClassifier()
```

```
classifier.fit(X_train,y_train)
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Performance Evaluation")
```

```
print(classification_report(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Accuracy:")
```

```
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import plot_confusion_matrix
```

```
plot_confusion_matrix(classifier, X_test, y_test)
```

```
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
```

```
% self.max_iter, ConvergenceWarning)
```

```
Confusion Matrix:
```

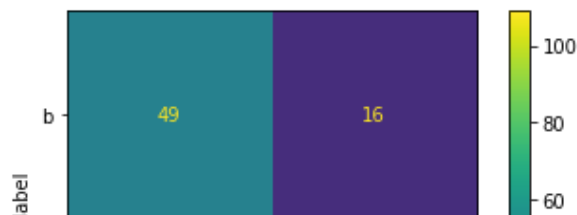
```
[[ 49  16]
 [   2 109]]
```

```
Performance Evaluation
```

	precision	recall	f1-score	support
b	0.96	0.75	0.84	65
g	0.87	0.98	0.92	111
accuracy			0.90	176
macro avg	0.92	0.87	0.88	176
weighted avg	0.90	0.90	0.89	176

```
Accuracy:
```

```
0.8977272727272727
```



```
# IONOSPHERE DATASET
```

```
# Multi Layer Perceptron(Without Tuning)[40-60 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[ 55  21]
 [  0 135]]

```

```

-----
Performance Evaluation
-----

```

	precision	recall	f1-score	support
b	1.00	0.72	0.84	76
g	0.87	1.00	0.93	135
accuracy			0.90	211
macro avg	0.93	0.86	0.88	211
weighted avg	0.91	0.90	0.90	211

```

-----
-----

```

Accuracy:

```

# IONOSPHERE DATASET
# Multi Layer Perceptron(Without Tuning)[30-70 split]

```

```

import pandas as pd
import numpy as np

```

```

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

```

```

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
            '20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1

```

```

df.columns = col_name

```

```

X = df.drop(['Class'], axis=1)
y = df['Class']

```

```

from sklearn.model_selection import train_test_split

```

```

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando

```

```

# Feature Scaling
from sklearn.preprocessing import StandardScaler

```

```

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

```

```

# Classification using MLP
from sklearn.neural_network import MLPClassifier

```

```

classifier = MLPClassifier()

```



```
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[ 63  21]
 [  3 159]]
-----
-----
Performance Evaluation
              precision    recall  f1-score   support

         b            0.95         0.75         0.84            84
         g            0.88         0.98         0.93           162

#####
      macro avg              0.92              0.87              0.88              246

# IONOSPHERE DATASET
# Multi Layer Perceptron(With Tuning)[70-30 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
            ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest

```

```

print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()

```

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)
```

Confusion Matrix:

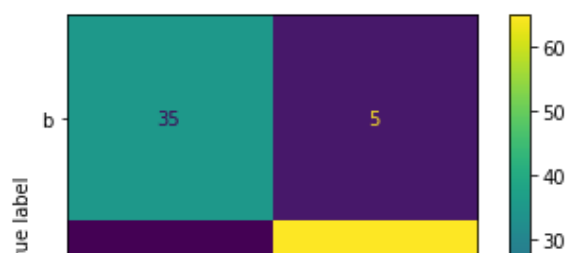
```
[[35  5]
 [ 1 65]]
```

Performance Evaluation

	precision	recall	f1-score	support
b	0.97	0.88	0.92	40
g	0.93	0.98	0.96	66
accuracy			0.94	106
macro avg	0.95	0.93	0.94	106
weighted avg	0.95	0.94	0.94	106

Accuracy:

0.9433962264150944



IONOSPHERE DATASET

Multi-Layer Perceptron (With Tuning) [60, 40, split]

```
# Multi Layer Perceptron (with Tuning) [60-40 Split]
```

```
import pandas as pd
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("ionosphere.data", header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','35']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```
parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)
```

```
#####
```

```
from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)
```

Confusion Matrix:

```
[[48  7]
 [ 0 86]]
```

```
-----
-----
Performance Evaluation
      precision    recall  f1-score   support

b         1.00      0.87      0.93         55
```

IONOSPHERE DATASET

Multi Layer Perceptron(With Tuning)[50-50 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("ionosphere.data", header=None)
```

```
col_name = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', 'C1']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```



```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'activation': 'relu',
  'alpha': 0.0001,
  'batch_size': 'auto',
  'beta_1': 0.9,
  'beta_2': 0.999,
  'early_stopping': False,
  'epsilon': 1e-08,
  'hidden_layer_sizes': (100,),
  'learning_rate': 'constant',
  'learning_rate_init': 0.001,
  'max_fun': 15000,
  'max_iter': 100,
  'momentum': 0.9,
  'n_iter_no_change': 10,
  'nesterovs_momentum': True,
  'power_t': 0.5,
  'random_state': None,
  'shuffle': True,
  'solver': 'adam',
  'tol': 0.0001,
  'validation_fraction': 0.1,
  'verbose': False}
```

IONOSPHERE DATASET

Multi Layer Perceptron(With Tuning)[40-60 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Classification

```

from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")

```

```
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt  
from sklearn.metrics import plot_confusion_matrix  
plot_confusion_matrix(rf_random, X_test, y_test)  
plt.show()
```

Parameters currently in use:

```
{'activation': 'relu',
  'alpha': 0.0001,
  'batch_size': 'auto',
  'beta_1': 0.9,
```

IONOSPHERE DATASET

Multi Layer Perceptron(With Tuning)[30-70 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','35','36',
            '37','38','39','40','41','42','43','44','45','46','47','48','49','50','51','52','53','54','55',
            '56','57','58','59','60','61','62','63','64','65','66','67','68','69','70','71','72','73','74',
            '75','76','77','78','79','80','81','82','83','84','85','86','87','88','89','90','91','92',
            '93','94','95','96','97','98','99','100','101','102','103','104','105','106','107','108',
            '109','110','111','112','113','114','115','116','117','118','119','120','121','122','123',
            '124','125','126','127','128','129','130','131','132','133','134','135','136','137',
            '138','139','140','141','142','143','144','145','146','147','148','149','150','151',
            '152','153','154','155','156','157','158','159','160','161','162','163','164','165',
            '166','167','168','169','170','171','172','173','174','175','176','177','178','179',
            '180','181','182','183','184','185','186','187','188','189','190','191','192',
            '193','194','195','196','197','198','199','200','201','202','203','204','205',
            '206','207','208','209','210','211','212','213','214','215','216','217','218',
            '219','220','221','222','223','224','225','226','227','228','229','230','231',
            '232','233','234','235','236','237','238','239','240','241','242','243','244',
            '245','246','247','248','249','250','251','252','253','254','255','256','257',
            '258','259','260','261','262','263','264','265','266','267','268','269','270',
            '271','272','273','274','275','276','277','278','279','280','281','282','283',
            '284','285','286','287','288','289','290','291','292','293','294','295','296',
            '297','298','299','300','301','302','303','304','305','306','307','308','309',
            '310','311','312','313','314','315','316','317','318','319','320','321','322',
            '323','324','325','326','327','328','329','330','331','332','333','334','335',
            '336','337','338','339','340','341','342','343','344','345','346','347','348',
            '349','350','351','352','353','354','355','356','357','358','359','360',
            '361','362','363','364','365','366','367','368','369','370','371','372',
            '373','374','375','376','377','378','379','380','381','382','383','384',
            '385','386','387','388','389','390','391','392','393','394','395','396',
            '397','398','399','400','401','402','403','404','405','406','407','408',
            '409','410','411','412','413','414','415','416','417','418','419','420',
            '421','422','423','424','425','426','427','428','429','430','431','432',
            '433','434','435','436','437','438','439','440','441','442','443','444',
            '445','446','447','448','449','450','451','452','453','454','455','456',
            '457','458','459','460','461','462','463','464','465','466','467','468',
            '469','470','471','472','473','474','475','476','477','478','479',
            '480','481','482','483','484','485','486','487','488','489','490',
            '491','492','493','494','495','496','497','498','499','500','501',
            '502','503','504','505','506','507','508','509','510','511','512',
            '513','514','515','516','517','518','519','520','521','522','523',
            '524','525','526','527','528','529','530','531','532','533','534',
            '535','536','537','538','539','540','541','542','543','544','545',
            '546','547','548','549','550','551','552','553','554','555','556',
            '557','558','559','560','561','562','563','564','565','566','567',
            '568','569','570','571','572','573','574','575','576','577','578',
            '579','580','581','582','583','584','585','586','587','588','589',
            '590','591','592','593','594','595','596','597','598','599','600',
            '601','602','603','604','605','606','607','608','609','610','611',
            '612','613','614','615','616','617','618','619','620','621','622',
            '623','624','625','626','627','628','629','630','631','632','633',
            '634','635','636','637','638','639','640','641','642','643','644',
            '645','646','647','648','649','650','651','652','653','654','655',
            '656','657','658','659','660','661','662','663','664','665','666',
            '667','668','669','670','671','672','673','674','675','676','677',
            '678','679','680','681','682','683','684','685','686','687','688',
            '689','690','691','692','693','694','695','696','697','698','699',
            '700','701','702','703','704','705','706','707','708','709','710',
            '711','712','713','714','715','716','717','718','719','720','721',
            '722','723','724','725','726','727','728','729','730','731','732',
            '733','734','735','736','737','738','739','740','741','742','743',
            '744','745','746','747','748','749','750','751','752','753','754',
            '755','756','757','758','759','760','761','762','763','764','765',
            '766','767','768','769','770','771','772','773','774','775','776',
            '777','778','779','780','781','782','783','784','785','786','787',
            '788','789','790','791','792','793','794','795','796','797','798',
            '799','800','801','802','803','804','805','806','807','808','809',
            '810','811','812','813','814','815','816','817','818','819','820',
            '821','822','823','824','825','826','827','828','829','830','831',
            '832','833','834','835','836','837','838','839','840','841','842',
            '843','844','845','846','847','848','849','850','851','852','853',
            '854','855','856','857','858','859','860','861','862','863','864',
            '865','866','867','868','869','870','871','872','873','874','875',
            '876','877','878','879','880','881','882','883','884','885','886',
            '887','888','889','890','891','892','893','894','895','896','897',
            '898','899','900','901','902','903','904','905','906','907','908',
            '909','910','911','912','913','914','915','916','917','918','919',
            '920','921','922','923','924','925','926','927','928','929','930',
            '931','932','933','934','935','936','937','938','939','940','941',
            '942','943','944','945','946','947','948','949','950','951','952',
            '953','954','955','956','957','958','959','960','961','962','963',
            '964','965','966','967','968','969','970','971','972','973','974',
            '975','976','977','978','979','980','981','982','983','984','985',
            '986','987','988','989','990','991','992','993','994','995','996',
            '997','998','999','1000','1001','1002','1003','1004','1005','1006',
            '1007','1008','1009','1010','1011','1012','1013','1014','1015','1016',
            '1017','1018','1019','1020','1021','1022','1023','1024','1025','1026',
            '1027','1028','1029','1030','1031','1032','1033','1034','1035','1036',
            '1037','1038','1039','1040','1041','1042','1043','1044','1045','1046',
            '1047','1048','1049','1050','1051','1052','1053','1054','1055','1056',
            '1057','1058','1059','1060','1061','1062','1063','1064','1065','1066',
            '1067','1068','1069','1070','1071','1072','1073','1074','1075','1076',
            '1077','1078','1079','1080','1081','1082','1083','1084','1085','1086',
            '1087','1088','1089','1090','1091','1092','1093','1094','1095','1096',
            '1097','1098','1099','1100','1101','1102','1103','1104','1105','1106',
            '1107','1108','1109','1110','1111','1112','1113','1114','1115','1116',
            '1117','1118','1119','1120','1121','1122','1123','1124','1125','1126',
            '1127','1128','1129','1130','1131','1132','1133','1134','1135','1136',
            '1137','1138','1139','1140','1141','1142','1143','1144','1145','1146',
            '1147','1148','1149','1150','1151','1152','1153','1154','1155','1156',
            '1157','1158','1159','1160','1161','1162','1163','1164','1165','1166',
            '1167','1168','1169','1170','1171','1172','1173','1174','1175','1176',
            '1177','1178','1179','1180','1181','1182','1183','1184','1185','1186',
            '1187','1188','1189','1190','1191','1192','1193','1194','1195','1196',
            '1197','1198','1199','1200','1201','1202','1203','1204','1205','1206',
            '1207','1208','1209','1210','1211','1212','1213','1214','1215','1216',
            '1217','1218','1219','1220','1221','1222','1223','1224','1225','1226',
            '1227','1228','1229','1230','1231','1232','1233','1234','1235','1236',
            '1237','1238','1239','1240','1241','1242','1243','1244','1245','1246',
            '1247','1248','1249','1250','1251','1252','1253','1254','1255','1256',
            '1257','1258','1259','1260','1261','1262','1263','1264','1265','1266',
            '1267','1268','1269','1270','1271','1272','1273','1274','1275','1276',
            '1277','1278','1279','1280','1281','1282','1283','1284','1285','1286',
            '1287','1288','1289','1290','1291','1292','1293','1294','1295','1296',
            '1297','1298','1299','1300','1301','1302','1303','1304','1305','1306',
            '1307','1308','1309','1310','1311','1312','1313','1314','1315','1316',
            '1317','1318','1319','1320','1321','1322','1323','1324','1325','1326',
            '1327','1328','1329','1330','1331','1332','1333','1334','1335','1336',
            '1337','1338','1339','1340','1341','1342','1343','1344','1345','1346',
            '1347','1348','1349','1350','1351','1352','1353','1354','1355','1356',
            '1357','1358','1359','1360','1361','1362','1363','1364','1365','1366',
            '1367','1368','1369','1370','1371','1372','1373','1374','1375','1376',
            '1377','1378','1379','1380','1381','1382','1383','1384','1385','1386',
            '1387','1388','1389','1390','1391','1392','1393','1394','1395','1396',
            '1397','1398','1399','1400','1401','1402','1403','1404','1405','1406',
            '1407','1408','1409','1410','1411','1412','1413','1414','1415','1416',
            '1417','1418','1419','1420','1421','1422','1423','1424','1425','1426',
            '1427','1428','1429','1430','1431','1432','1433','1434','1435','1436',
            '1437','1438','1439','1440','1441','1442','1443','1444','1445','1446',
            '1447','1448','1449','1450','1451','1452','1453','1454','1455','1456',
            '1457','1458','1459','1460','1461','1462','1463','1464','1465','1466',
            '1467','1468','1469','1470','1471','1472','1473','1474','1475','1476',
            '1477','1478','1479','1480','1481','1482','1483','1484','1485','1486',
            '1487','1488','1489','1490','1491','1492','1493','1494','1495','1496',
            '1497','1498','1499','1500','1501','1502','1503','1504','1505','1506',
            '1507','1508','1509','1510','1511','1512','1513','1514','1515','1516',
            '1517','1518','1519','1520','1521','1522','1523','1524','1525','1526',
            '1527','1528','1529','1530','1531','1532','1533','1534','1535','1536',
            '1537','1538','1539','1540','1541','1542','1543','1544','1545','1546',
            '1547','1548','1549','1550','1551','1552','1553','1554','1555','1556',
            '1557','1558','1559','1560','1561','1562','1563','1564','1565','1566',
            '1567','1568','1569','1570','1571','1572','1573','1574','1575','1576',
            '1577','1578','1579','1580','1581','1582','1583','1584','1585','1586',
            '1587','1588','1589','1590','1591','1592','1593','1594','1595','1596',
            '1597','1598','1599','1600','1601','1602','1603','1604','1605','1606',
            '1607','1608','1609','1610','1611','1612','1613','1614','1615','1616',
            '1617','1618','1619','1620','1621','1622','1623','1624','1625','1626',
            '1627','1628','1629','1630','1631','1632','1633','1634','1635','1636',
            '1637','1638','1639','1640','1641','1642','1643','1644','1645','1646',
            '1647','1648','1649','1650','1651','1652','1653','1654','1655','1656',
            '1657','1658','1659','1660','1661','1662','1663','1664','1665','1666',
            '1667','1668','1669','1670','1671','1672','1673','1674','1675','1676',
            '1677','1678','1679','1680','1681','1682','1683','1684','1685','1686',
            '1687','1688','1689','1690','1691','1692','1693','1694','1695','1696',
            '1697','1698','1699','1700','1701','1702','1703','1704','1705','1706',
            '1707','1708','1709','1710','1711','1712','1713','1714','1715','1716',
            '1717','1718','1719','1720','1721','1722','1723','1724','1725','1726',
            '1727','1728','1729','1730','1731','1732','1733','1734','1735','1736',
            '1737','1738','1739','1740','1741','1742','1743','1744','1745','1746',
            '1747','1748','1749','1750','1751','1752','1753','1754','1755','1756',
            '1757','1758','1759','1760','1761','1762','1763','1764','1765','1766',
            '1767','1768','1769','1770','1771','1772','1773','1774','1775','1776',
            '1777','1778','1779','1780','1781','1782','1783','1784','1785','1786',
            '1787','1788','1789','1790','1791','1792','1793','1794','1795','1796',
            '1797','1798','1799','1800','1801','1802','1803','1804','1805','1806',
            '1807','1808','1809','1810','1811','1812','1813','1814','1815','1816',
            '1817','1818','1819','1820','1821','1822','1823','1824','1825','1826',
            '1827','1828','1829','1830','1831','1832','1833','1834','1835','1836',
            '1837','1838','1839','1840','1841','1842','1843','1844','1845','1846',
            '1847','1848','1849','1850','1851','1852','1853','1854','1855','1856',
            '1857','1858','1859','1860','1861','1862','1863','1864','1865','1866',
            '1867','1868','1869','1870','1871','1872','1873','1874','1875','1876',
            '1877','1878','1879','1880','1881','1882','1883','1884','1885','1886',
            '1887','1888','1889','1890','1891','1892','1893','1894','1895','1896',
            '1897','1898','1899','1900','1901','1902','1903','1904','1905','1906',
            '1907','1908','1909','1910','1911','1912','1913','1914','1915','1916',
            '1917','1918','1919','1920','1921','1922','1923','1924','1925','1926',
            '1927','1928','1929','1930','1931','1932','1933','1934','1935','1936',
            '1937','1938','1939','1940','1941','1942','1943','1944','1945','1946',
            '1947','1948','1949','1950','1951','1952','1953','1954','1955','1956',
            '1957','1958','1959','1960','1961','1962','1963','1964','1965','1966',
            '1967','1968','1969','1970','1971','1972','1973','1974','1975','1976',
            '1977','1978','1979','1980','1981','1982','1983','1984','1985','1986',
            '1987','1988','1989','1990','1991','1992','1993','1994','1995','1996',
            '1997','1998','1999','2000','2001','2002','2003','2004','2005','2006',
            '2007','2008','2009','2010','2011','2012','2013','2014','2015','2016',
            '2017','2018','2019','2020','2021','2022','2023','2024','2025','2026',
            '2027','2028','2029','2030','2031','2032','2033','2034','2035','2036',
            '2037','2038','2039','2040','2041','2042','2043','2044','2045','2046',
            '2047','2048','2049','2050','2051','2052','2053','2054','2055','2056',
            '2057','2058','2059','2060','2061','2062','2063','2064','2065','2066',
            '2067','2068','2069','2070','2071','2072','2073','2074','2075','2076',
            '2077','2078','2079','2080','2081','2082','2083','2084','2085','2086',
            '2087','2088','2089','2090','2091','2092','2093','2094','2095','2096',
            '2097','2098','2099','2100','2101','2102','2103','2104','2105','2106',
            '2107','2108','2109','2110','2111','2112','2113','2114','2115','2116',
            '2117','2118','2119','2120','2121','2122','2123','2124','2125','2126',
            '2127','2128','2129','2130','2131','2132','2133','2134','2135','2136',
            '2137','2138','2139','2140','2141','2142','2143','2144','2145','2146',
            '2147','2148','2149','2150','2151','2152','2153','2154','2155','2156',
            '2157','2158','2159','2160','2161','2162','2163','2164','2165','2166',
            '2167','2168','2169','2170','2171','2172','2173','2174','2175','2176',
            '2177','2178','2179','2180','2181','2182','2183','2184','2185','2186',
            '2187','2188','2189','2190','2191','2192','2193','2194','2195','2196',
            '2197','2198','2199','2200','2201','2202','2203','2204','2205','2206',
            '2207','2208','2209','2210','2211','2212','2213','2214','2215','2216',
            '2217','2218','2219','2220','2221','2222','2223','2224','2225','2226',
            '2227','2228','2229','2230','2231','2232','2233','2234','2235','2236',
            '2237','2238','2239','2240','2241','2242','2243','2244','2245','2246',
            '2247','2248','2249','2250','2251','2252','2253','2254','2255','2256',
            '2257','2258','2259','2260','2261','2262','2263','2264','2265','2266',
            '2267','2268','2269','2270','2271','2272','2273','2274','2275','2276',
            '2277','2278','2279','2280','2281','2282','2283','2284','2285','2286',
            '2287','2288','2289','2290','2291','2292','2293','2294','2295','2296',
            '2297','2298','2299','2300','2301','2302','2303','2304','2305','2306',
            '2307','2308','2309','2310','2311','2312','2313','2314','2315','2316',
            '2317','2318','2319','2320','2321','2322','2323','2324','2325','2326',
            '2327','2328','2329','2330','2331','2332','2333','2334','2335','2336',
            '2337','2338','2339','2340','2341','2342','2343','2344','2345','2346',
            '2347','2348','2349','2350','2351','2352','2353','2354','2355','2356',
            '2357','2358','2359','2360','2361','2362','2363','2364','2365','2366',
            '2367','2368','2369','2370','2371','2372','2373','2374','2375','2376',
            '2377','2378','2379','2380','2381','2382','2383','2384','2385','2386',
            '2387','2388','2389','2390','2391','2392','2393','2394','2395','2396',
            '2397','2398','2399','2400','2401','2402','2403','2404','2405','2406',
            '2407','2408','2409','2410','2411','2412','2413','2414','2415','2416',
            '2417','2418','2419','2420','2421','2422','2423','2424','2425','2426',
            '2427','2428','2429','2430','2431','2432','2433','2434','2435','2436',
            '2437','2438','2439','2440','2441','2442','2443','2444','2445','2446',
            '2447','2448','2449','2450','2451','2452','2453','2454','2455','2456',
            '2457','2458','2459','2460','2461','2462','2463','2464','2465','2466',
            '2467','2468','2469','247
```

```

        'activation': ['tanh', 'relu'],
        'solver': ['sgd', 'adam'],
        'alpha': [0.0001, 0.05],
        'learning_rate': ['constant', 'adaptive'],
    }
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()

```

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)
```

Confusion Matrix:

```
[[ 69  15]
 [   0 162]]
```


Performance Evaluation

	precision	recall	f1-score	support
b	1.00	0.82	0.90	84
g	0.92	1.00	0.96	162
accuracy			0.94	246
macro avg	0.96	0.91	0.93	246
weighted avg	0.94	0.94	0.94	246

Accuracy:

#####



IONOSPHERE DATASET

SVM(Without Tuning)[70-30 split]

import pandas as pd

import numpy as np

```
# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
```



```
plt.show()
```

Confusion Matrix:

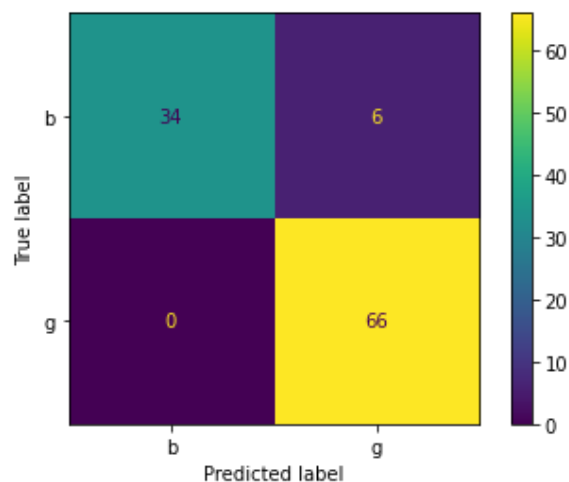
```
[[34  6]
 [ 0 66]]
```

Performance Evaluation

	precision	recall	f1-score	support
b	1.00	0.85	0.92	40
g	0.92	1.00	0.96	66
accuracy			0.94	106
macro avg	0.96	0.93	0.94	106
weighted avg	0.95	0.94	0.94	106

Accuracy:

0.9433962264150944



```
# IONOSPHERE DATASET
```

```
# SVM(Without Tuning)[60-40 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

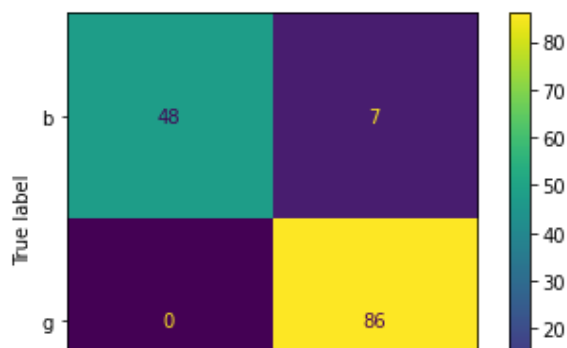
```
[[48  7]
 [ 0 86]]
```

Performance Evaluation

	precision	recall	f1-score	support
b	1.00	0.87	0.93	55
g	0.92	1.00	0.96	86
accuracy			0.95	141
macro avg	0.96	0.94	0.95	141
weighted avg	0.95	0.95	0.95	141

Accuracy:

0.950354609929078



```
# IONOSPHERE DATASET
```

```
# SVM(Without Tuning)[50-50 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

```
[[ 55  10]
 [   2 109]]
```

Performance Evaluation

	precision	recall	f1-score	support
b	0.96	0.85	0.90	65
g	0.92	0.98	0.95	111
accuracy			0.93	176
macro avg	0.94	0.91	0.92	176
weighted avg	0.93	0.93	0.93	176

Accuracy:

```
0.9318181818181818
```

IONOSPHERE DATASET

SVM(Without Tuning)[40-60 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Classification

```
from sklearn.svm import SVC
```

```
classifier = SVC()
classifier.fit(X_train,y_train)
```

```
y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```

Confusion Matrix:
[[ 64  12]
 [  1 134]]
-----
-----
Performance Evaluation
              precision    recall  f1-score   support

# IONOSPHERE DATASET
# SVM(Without Tuning)[30-70 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
            ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

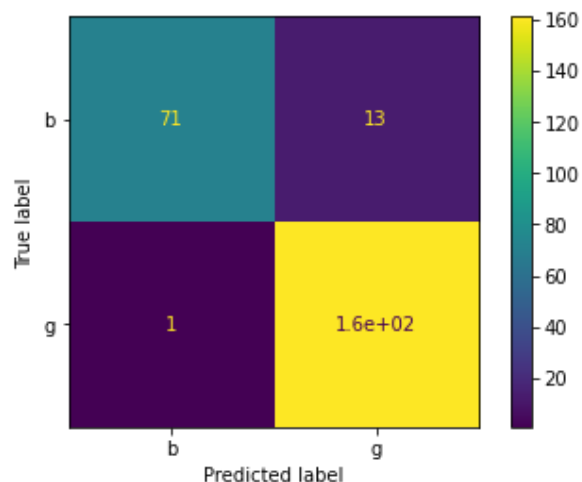
```
[[ 71  13]
 [   1 161]]
```

Performance Evaluation

	precision	recall	f1-score	support
b	0.99	0.85	0.91	84
g	0.93	0.99	0.96	162
accuracy			0.94	246
macro avg	0.96	0.92	0.93	246
weighted avg	0.95	0.94	0.94	246

Accuracy:

0.943089430894309



#####


```
# IONOSPHERE DATASET
# SVM(With Tuning)[70-30 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','35']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly'],
              'degree': [1,2,3,4,5,6,7,8,9,10]}

pprint(param_grid)

#####
```

```
from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
```

Fitting 5 folds for each of 48 candidates, totalling 240 fits

```
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
```

IONOSPHERE DATASET

SVM(With Tuning)[60-40 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001], 'kernel': ['rbf', 'poly'],

pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")

```

```
print("-----")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [0.1, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000]}
```

IONOSPHERE DATASET

SVM(With Tuning)[50-50 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Classification

```
from sklearn.svm import SVC
```

```
classifier = SVC()
```

```
#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001], 'kernel': ['rbf', 'poly'],

pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
```

Fitting 5 folds for each of 48 candidates, totalling 240 fits

```
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf .....
[CV] ..... C=0.1, gamma=0.1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf .....
[CV] ..... C=0.1, gamma=0.1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf .....
[CV] ..... C=0.1, gamma=0.1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf .....
[CV] ..... C=0.1, gamma=0.1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf .....
[CV] ..... C=0.1, gamma=0.1, kernel=rbf, total= 0.0s
```

IONOSPHERE DATASET

SVM(10-fold Training) 100 0.0 0.0 1.0


```
import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data", header=None)

col_name = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', 'Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.4, test_size=0.6, random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel': ['rbf', 'poly', 'linear']}

pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV
```

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
```

Fitting 5 folds for each of 48 candidates, totalling 240 fits

```
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
```

IONOSPHERE DATASET

SVM(With Tuning)[30-70 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
            '20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001], 'kernel': ['rbf', 'poly'],
               'degree': [1,2,3,4,5]}

pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

```
#####
    'kernel': ['rbf', 'poly', 'sigmoid']}]
```

IONOSPHERE DATASET

Random Forest Classifier(Without Tuning)[70-30 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','1
            ,'20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

Finding the important parameters that contribute to most of the variance in the data.

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=34)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 15 important parameters

pca = PCA(n_components=15)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

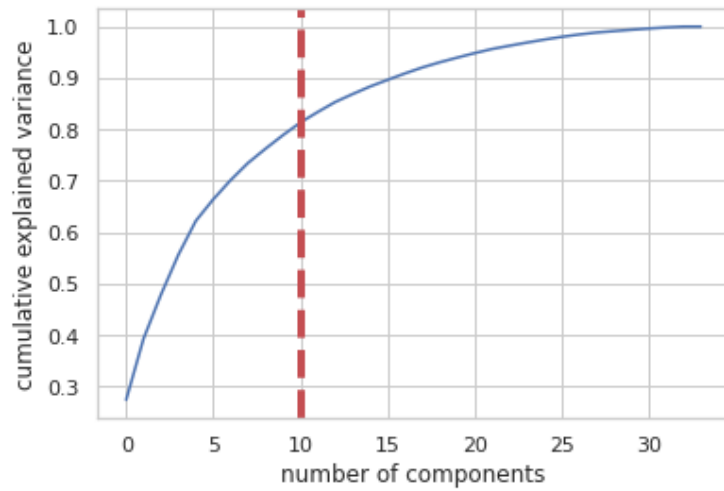
print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



None

Confusion Matrix:

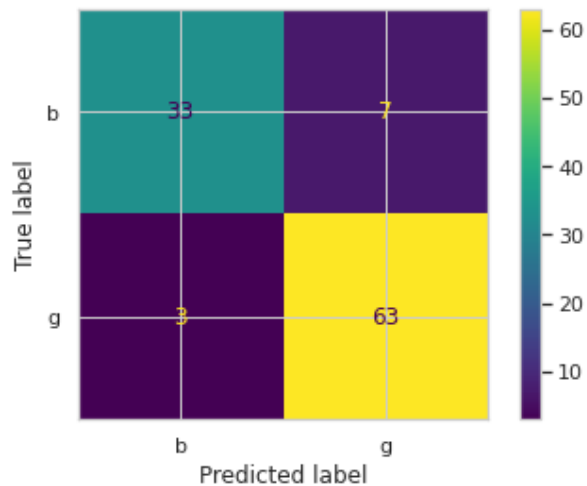
```
[[33  7]
 [ 3 63]]
```

Performance Evaluation

	precision	recall	f1-score	support
b	0.92	0.82	0.87	40
g	0.90	0.95	0.93	66
accuracy			0.91	106
macro avg	0.91	0.89	0.90	106
weighted avg	0.91	0.91	0.90	106

Accuracy:

0.9056603773584906



IONOSPHERE DATASET

Random Forest Classifier(With Tuning)[70-30 split]

```
import pandas as pd
import numpy as np
```



```
# Dataset Preparation
df = pd.read_csv("ionosphere.data",header=None)

col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=34)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 15 important parameters

pca = PCA(n_components=15)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}
pprint(random_grid)
```

```
#####
```

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

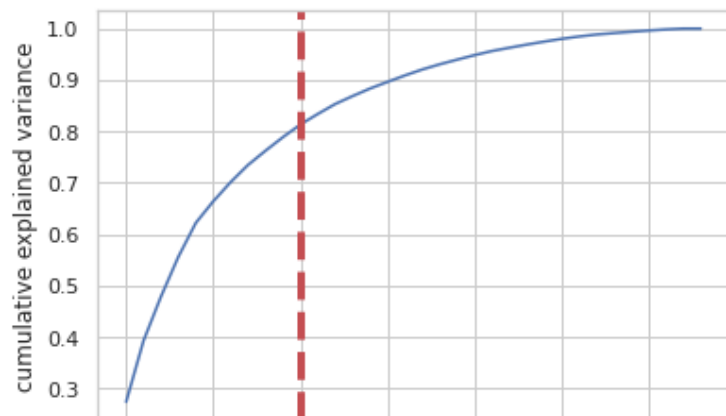
```
print("-----")
print("-----")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```



```
# IONOSPHERE DATASET
```

```
# Multi Layer Perceptron(Without Tuning)[70-30 split]
```

```
import pandas as pd
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Finding the important parameters that contribute to most of the variance in the data.
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
```

```
pca_test = PCA(n_components=34)
pca_test.fit(X_train)
sns.set(style='whitegrid')
```

```
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=15)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

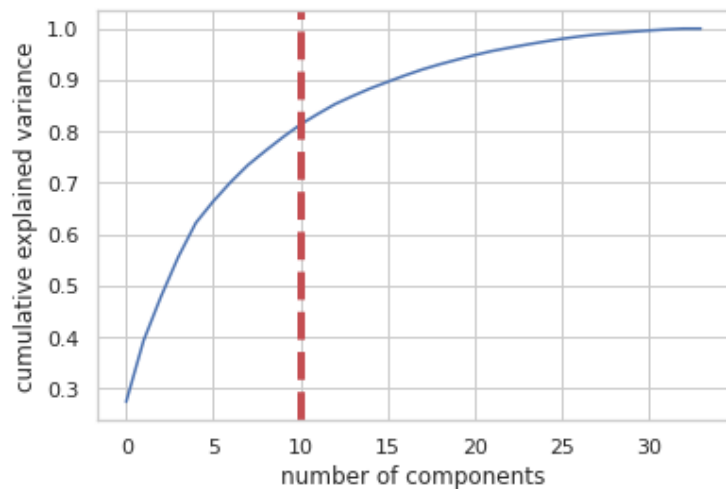
print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



None

Confusion Matrix:

```
[[27 13]
 [ 1 65]]
```

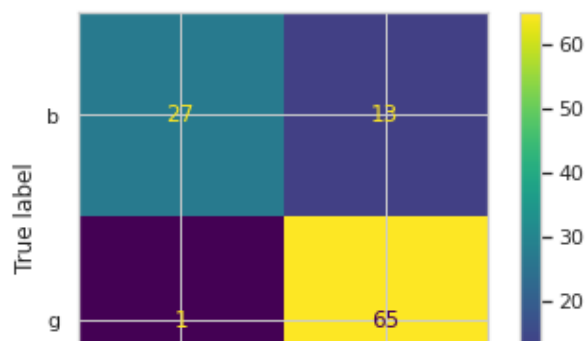
Performance Evaluation

	precision	recall	f1-score	support
b	0.96	0.68	0.79	40
g	0.83	0.98	0.90	66
accuracy			0.87	106
macro avg	0.90	0.83	0.85	106
weighted avg	0.88	0.87	0.86	106

Accuracy:

0.8679245283018868

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)



IONOSPHERE DATASET

Multi Layer Perceptron(With Tuning)[70-30 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("ionosphere.data",header=None)

col name = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '1

```
col_name = ['f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9', 'f10', 'f11', 'f12', 'f13', 'f14', 'f15', 'f16', 'f17', 'f18', 'f19', 'f20', 'f21', 'f22', 'f23', 'f24', 'f25', 'f26', 'f27', 'f28', 'f29', 'f30', 'f31', 'f32', 'f33', 'f34', 'Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Finding the important parameters that contribute to most of the variance in the data.
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
```

```
pca_test = PCA(n_components=34)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())
```

```
# So we can see that we have 10 important parameters
```

```
pca = PCA(n_components=15)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
```

```
# Classification
```

```
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
```

```

print(classifier.get_params())

#####
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant', 'adaptive'],
}
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

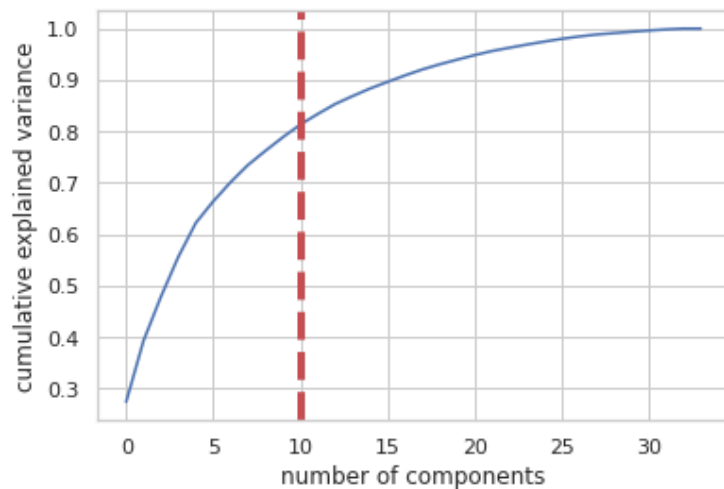
print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()

```

None

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
```

Confusion Matrix:

```
[[30 10]
 [ 1 65]]
```

Performance Evaluation

	precision	recall	f1-score	support
b	0.97	0.75	0.85	40
g	0.87	0.98	0.92	66

IONOSPHERE DATASET

SVM(Without Tuning)[70-30 split]

```

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("ionosphere.data", header=None)

col_name = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', 'Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=0.3, random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=34)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle='--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=15)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification
from sklearn.svm import SVC

```

```
from sklearn.svm import SVC
```

```
classifier = SVC()  
classifier.fit(X_train,y_train)
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")  
print(confusion_matrix(y_test, y_pred))
```

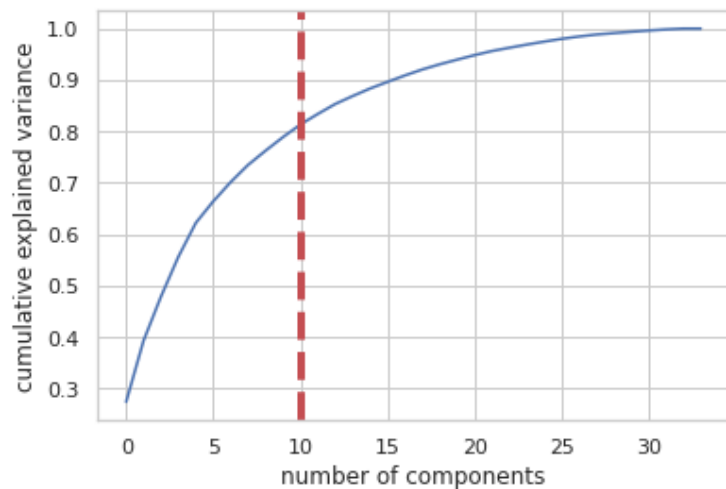
```
print("-----")  
print("-----")
```

```
print("Performance Evaluation")  
print(classification_report(y_test, y_pred))
```

```
print("-----")  
print("-----")
```

```
print("Accuracy:")  
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt  
from sklearn.metrics import plot_confusion_matrix  
plot_confusion_matrix(classifier, X_test, y_test)  
plt.show()
```



None

Confusion Matrix:

```
[[33  7]
 [ 0 66]]
```

Performance Evaluation

	precision	recall	f1-score	support
b	1.00	0.82	0.90	40
g	0.90	1.00	0.95	66
accuracy			0.93	106

IONOSPHERE DATASET

SVM(With Tuning)[70-30 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("ionosphere.data",header=None)
```

```
col_name = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
            '19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34','C1']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=34)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=15)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001], 'kernel': ['rbf', 'poly'],

pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()

```

```
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

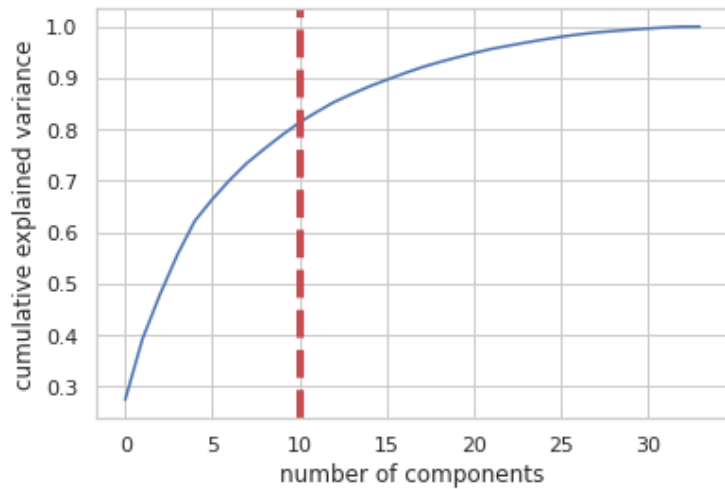
print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```



None

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

```
'kernel': ['rbf', 'poly', 'sigmoid']}]
```

Fitting 5 folds for each of 48 candidates, totalling 240 fits

```
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
```




```
# IRIS PLANT DATASET
# Random Forest Classifier(Without Tuning)[70-30 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

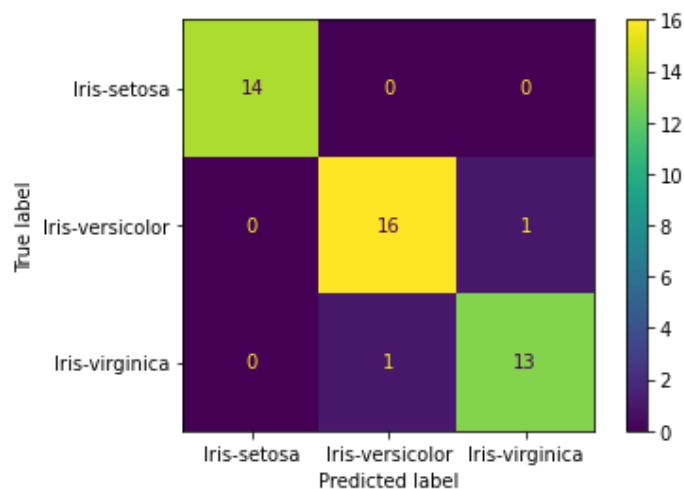
```
[[14  0  0]
 [ 0 16  1]
 [ 0  1 13]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	14
Iris-versicolor	0.94	0.94	0.94	17
Iris-virginica	0.93	0.93	0.93	14
accuracy			0.96	45
macro avg	0.96	0.96	0.96	45
weighted avg	0.96	0.96	0.96	45

Accuracy:

0.9555555555555556



```
# IRIS PLANT DATASET
# Random Forest Classifier(Without Tuning)[60-40 split]
```

```
import pandas as pd
import numpy as np
```

```
# Dataset Preparation
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,randome
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

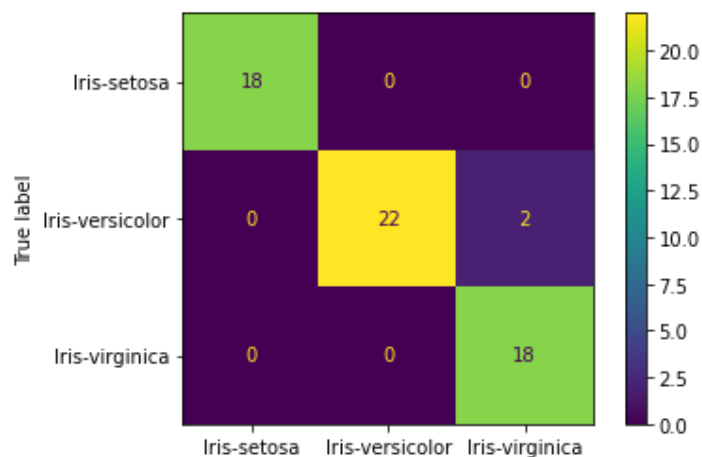
```
[[18  0  0]
 [ 0 22  2]
 [ 0  0 18]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	18
Iris-versicolor	1.00	0.92	0.96	24
Iris-virginica	0.90	1.00	0.95	18
accuracy			0.97	60
macro avg	0.97	0.97	0.97	60
weighted avg	0.97	0.97	0.97	60

Accuracy:

0.9666666666666667



IRIS PLANT DATASET

Random Forest Classifier(Without Tuning)[50-50 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando
```

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

```
[[23  0  0]
 [ 0 23  4]
 [ 0  1 24]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	23
Iris-versicolor	0.96	0.85	0.90	27
Iris-virginica	0.86	0.96	0.91	25
accuracy			0.93	75
macro avg	0.94	0.94	0.94	75
weighted avg	0.94	0.93	0.93	75

Accuracy:

0.9333333333333333



IRIS PLANT DATASET

Random Forest Classifier(Without Tuning)[40-60 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,rando
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

Classification

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

```
[[28  0  0]
 [ 0 28  4]
 [ 0  1 29]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	28
Iris-versicolor	0.97	0.88	0.92	32
Iris-virginica	0.88	0.97	0.92	30

IRIS PLANT DATASET

Random Forest Classifier(Without Tuning)[30-70 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Classification

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```



```
print("-----")
print("-----")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

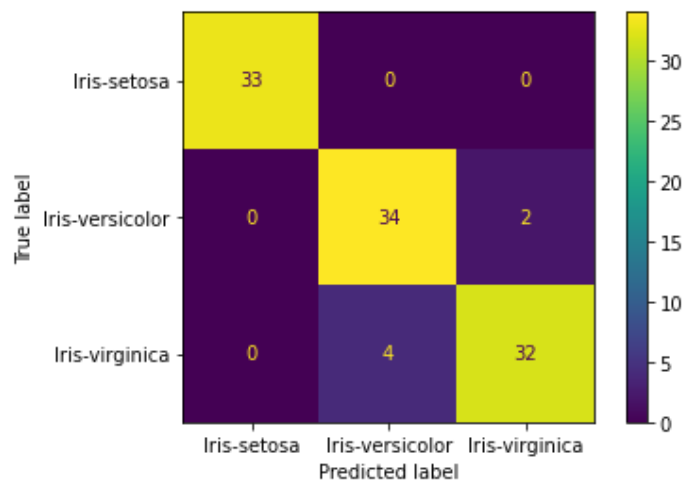
```
[[33  0  0]
 [ 0 34  2]
 [ 0  4 32]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	33
Iris-versicolor	0.89	0.94	0.92	36
Iris-virginica	0.94	0.89	0.91	36
accuracy			0.94	105
macro avg	0.95	0.94	0.94	105
weighted avg	0.94	0.94	0.94	105

Accuracy:

0.9428571428571428



```
#####
```

```
# IRIS PLANT DATASET
```

```
# Random Forest Classifier(With Tuning)[70-30 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier()
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
```

```
# Look at parameters used by our current forest
```

```
print('Parameters currently in use:\n')
```

```
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```
from sklearn.model_selection import RandomizedSearchCV
```

```
# Number of trees in random forest
```

```
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
```

```

# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}
pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)

```

```
plt.show()
```

```

# IRIS PLANT DATASET
# Random Forest Classifier(With Tuning)[60-40 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}
pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()

```

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 43.7s

[Parallel(n_jobs=-1)]: Done 158 tasks | elapsed: 3.1min

[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 5.9min finished

Confusion Matrix:

```
[[18  0  0]
 [ 0 22  2]
 [ 0  0 18]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	18
Iris-versicolor	1.00	0.92	0.96	24
Iris-virginica	0.90	1.00	0.95	18
accuracy			0.97	60
macro avg	0.97	0.97	0.97	60
weighted avg	0.97	0.97	0.97	60

Accuracy:

0.9666666666666667



IRIS PLANT DATASET

Random Forest Classifier(With Tuning)[50-50 split]

import pandas as pd

```

import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}
pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 44.2s

[Parallel(n_jobs=-1)]: Done 158 tasks | elapsed: 3.1min

[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 5.9min finished

Confusion Matrix:

```
[[23  0  0]
 [ 0 23  4]
 [ 0  1 24]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	23
Iris-versicolor	0.96	0.85	0.90	27
Iris-virginica	0.86	0.96	0.91	25
accuracy			0.93	75
macro avg	0.94	0.94	0.94	75
weighted avg	0.94	0.93	0.93	75

IRIS PLANT DATASET

Random Forest Classifier(With Tuning)[40-60 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("iris.data",header=None)
```

```

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}

```

```

        max_depth : max_depth,
        'min_samples_split': min_samples_split,
        'min_samples_leaf': min_samples_leaf,
        'bootstrap': bootstrap}
pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()

```

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 44.2s

[Parallel(n_jobs=-1)]: Done 158 tasks | elapsed: 3.0min

[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 5.9min finished

Confusion Matrix:

```
[[28  0  0]
 [ 0 28  4]
 [ 0  1 29]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	28
Iris-versicolor	0.97	0.88	0.92	32
Iris-virginica	0.88	0.97	0.92	30

IRIS PLANT DATASET

Random Forest Classifier(With Tuning)[30-70 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}

pprint(random_grid)

```

```
#####
```

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 43.5s

[Parallel(n_jobs=-1)]: Done 158 tasks | elapsed: 3.0min

[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 5.9min finished

Confusion Matrix:

```
[[33  0  0]
 [ 0 33  3]
 [ 0  3 33]]
```

Performance Evaluation

```
#####
```

```
iris-setosa    1.00    1.00    1.00    55
```

IRIS PLANT DATASET

Multi Layer Perceptron(Without Tuning)[70-30 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name

```
X = df.drop('Class',axis=1)
```



```
x = df.drop([ 'Class' ], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
```

```
% self.max_iter, ConvergenceWarning)
```

```
Confusion Matrix:
```

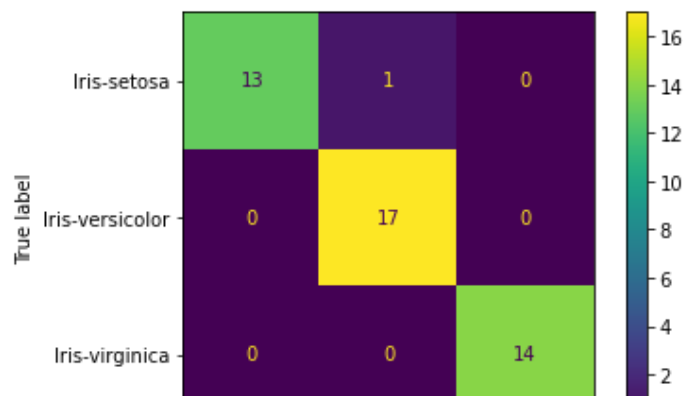
```
[[13  1  0]
 [ 0 17  0]
 [ 0  0 14]]
```

```
Performance Evaluation
```

	precision	recall	f1-score	support
Iris-setosa	1.00	0.93	0.96	14
Iris-versicolor	0.94	1.00	0.97	17
Iris-virginica	1.00	1.00	1.00	14
accuracy			0.98	45
macro avg	0.98	0.98	0.98	45
weighted avg	0.98	0.98	0.98	45

```
Accuracy:
```

```
0.9777777777777777
```



```
# IRIS PLANT DATASET
```

```
# Multi Layer Perceptron(Without Tuning)[60-40 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=42)
```

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
```

```
% self.max_iter, ConvergenceWarning)
```

```
Confusion Matrix:
```

```
[[17  1  0]
 [ 0 20  4]
 [ 0  0 18]]
```

```
-----
Performance Evaluation
```

	precision	recall	f1-score	support
Iris-setosa	1.00	0.94	0.97	18
Iris-versicolor	0.95	0.83	0.89	24
Iris-virginica	0.82	1.00	0.90	18
accuracy			0.92	60
macro avg	0.92	0.93	0.92	60
weighted avg	0.93	0.92	0.92	60

```
-----
Accuracy:
```

```
0.9166666666666666
```



```
# IRIS PLANT DATASET
```

```
# Multi Layer Perceptron(Without Tuning)[50-50 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

```
[[22  1  0]
 [ 0 23  4]
 [ 0  0 25]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	0.96	0.98	23
Iris-versicolor	0.96	0.85	0.90	27
Iris-virginica	0.86	1.00	0.93	25
accuracy			0.93	75
macro avg	0.94	0.94	0.94	75
weighted avg	0.94	0.93	0.93	75

Accuracy:

0.9333333333333333

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)

IRIS PLANT DATASET

Multi Layer Perceptron(Without Tuning)[40-60 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)

y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random

Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

Classification using MLP

from sklearn.neural_network import MLPClassifier

```
classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

```
[[28  0  0]
 [ 0 28  4]
 [ 0  0 30]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	28
Iris-versicolor	1.00	0.88	0.93	32
Iris-virginica	0.88	1.00	0.94	30
accuracy			0.96	90
macro avg	0.96	0.96	0.96	90
weighted avg	0.96	0.96	0.96	90

IRIS PLANT DATASET

Multi Layer Perceptron(Without Tuning)[30-70 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

Classification using MLP

```
from sklearn.neural_network import MLPClassifier
```

```
classifier = MLPClassifier()
```

```
classifier.fit(X_train,y_train)
```

```
y_pred = classifier.predict(X_test)
```



```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

```
[[33  0  0]
 [ 0 31  5]
 [ 0  1 35]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	33

IRIS PLANT DATASET

Multi Layer Perceptron(With Tuning)[70-30 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)

y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)

Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

Classification

from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier(max_iter=100)

#####

Showing all the parameters

from pprint import pprint

Look at parameters used by our current forest

print('Parameters currently in use:\n')

pprint(classifier.get_params())

#####

Creating a set of important sample features

```

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()

```

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)
```

Confusion Matrix:

```
[[14  0  0]
 [ 0 17  0]
 [ 0  0 14]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	14
Iris-versicolor	1.00	1.00	1.00	17
Iris-virginica	1.00	1.00	1.00	14
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

Accuracy:

1.0

IRIS PLANT DATASET

Multi Layer Perceptron(With Tuning)[60-40 split]

```
import pandas as pd
import numpy as np
```

```

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)

```

```
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[18  0  0]
```

IRIS PLANT DATASET

Multi Layer Perceptron(With Tuning)[50-50 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando
```

```

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

```



```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
```

```
# IRIS PLANT DATASET
```

```
# Multi Layer Perceptron(With Tuning)[40-60 split]
```

```
import pandas as pd
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
# Look at parameters used by our current forest
```

```

print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant', 'adaptive'],
}
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()

```

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
 % self.max_iter, ConvergenceWarning)
```

Confusion Matrix:

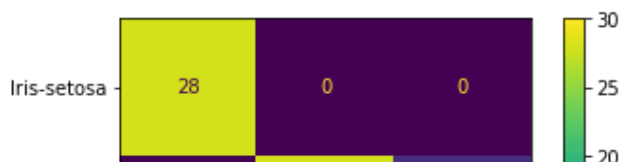
```
[[28  0  0]
 [ 0 28  4]
 [ 0  0 30]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	28
Iris-versicolor	1.00	0.88	0.93	32
Iris-virginica	0.88	1.00	0.94	30
accuracy			0.96	90
macro avg	0.96	0.96	0.96	90
weighted avg	0.96	0.96	0.96	90

Accuracy:

0.9555555555555556



IRIS PLANT DATASET

Multi Layer Perceptron(With Tuning)[30-70 split]

```

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

#####

```

```
from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
 % self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[33  0  0]
 [ 0 32  4]
 [ 0  1 35]]
```

Performance Evaluation

IRIS PLANT DATASET

SVM(Without Tuning)[70-30 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```


Confusion Matrix:

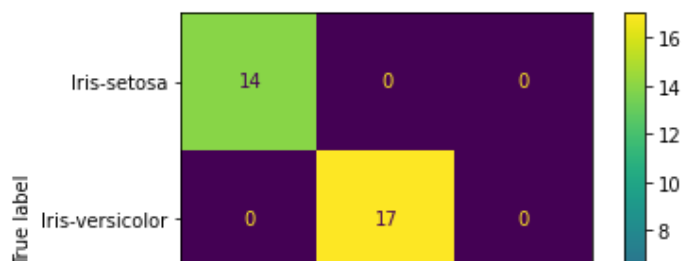
```
[[14  0  0]
 [ 0 17  0]
 [ 0  0 14]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	14
Iris-versicolor	1.00	1.00	1.00	17
Iris-virginica	1.00	1.00	1.00	14
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

Accuracy:

1.0



IRIS PLANT DATASET

SVM(Without Tuning)[60-40 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

```
[[18  0  0]
 [ 0 23  1]
 [ 0  0 18]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	18
Iris-versicolor	1.00	0.96	0.98	24
Iris-virginica	0.95	1.00	0.97	18
accuracy			0.98	60
macro avg	0.98	0.99	0.98	60
weighted avg	0.98	0.98	0.98	60

IRIS PLANT DATASET

SVM(Without Tuning)[50-50 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Classification

```
from sklearn.svm import SVC
```

```
classifier = SVC()
classifier.fit(X_train,y_train)
```

```
y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

```
[[23  0  0]
 [ 0 23  4]
 [ 0  1 24]]
```

```
-----
-----
```

Performance Evaluation

```
precision    recall  f1-score   support
```

IRIS PLANT DATASET

SVM(Without Tuning)[40-60 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

Classification

```
from sklearn.svm import SVC
```

```
classifier = SVC()
```

```
classifier.fit(X_train,y_train)
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

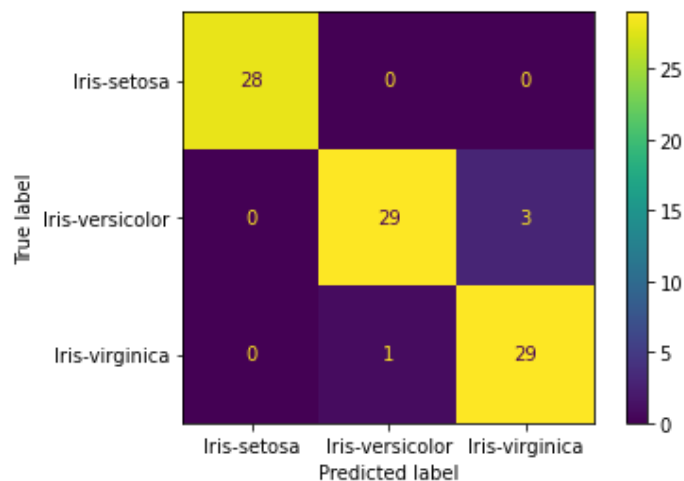
```
[[28  0  0]
 [ 0 29  3]
 [ 0  1 29]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	28
Iris-versicolor	0.97	0.91	0.94	32
Iris-virginica	0.91	0.97	0.94	30
accuracy			0.96	90
macro avg	0.96	0.96	0.96	90
weighted avg	0.96	0.96	0.96	90

Accuracy:

0.9555555555555556



```
# IRIS PLANT DATASET
# SVM(Without Tuning)[30-70 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,rando

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

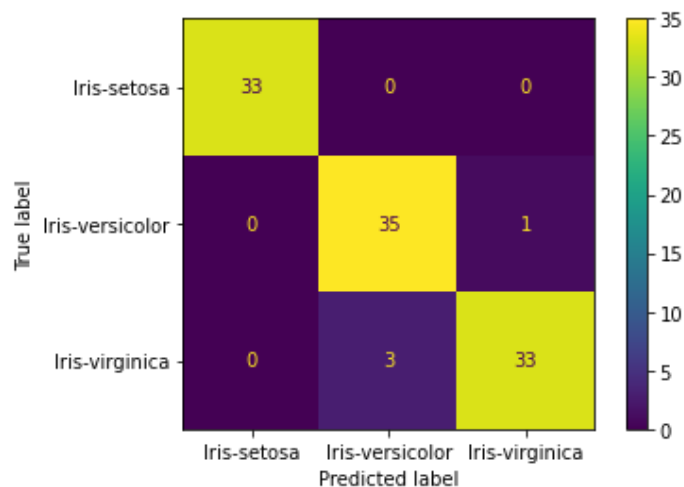
```
[[33  0  0]
 [ 0 35  1]
 [ 0  3 33]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	33
Iris-versicolor	0.92	0.97	0.95	36
Iris-virginica	0.97	0.92	0.94	36
accuracy			0.96	105
macro avg	0.96	0.96	0.96	105
weighted avg	0.96	0.96	0.96	105

Accuracy:

0.9619047619047619



```
# IRIS PLANT DATASET
# SVM(With Tuning)[70-30 split]
```

```
import pandas as pd
import numpy as np
```

```
# Dataset Preparation
df = pd.read_csv("iris.data",header=None)
```



```

col_name = ['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width', 'Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001], 'kernel': ['rbf', 'poly'],

pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

```

```
y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
```

Fitting 5 folds for each of 48 candidates, totalling 240 fits

```
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
```

IRIS PLANT DATASET

SVM(With Tuning)[60-40 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
```

```

X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001], 'kernel': ['rbf', 'poly'],

pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")

```

```
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
```

```
# IRIS PLANT DATASET
```

```
# SVM(With Tuning)[50-50 split]
```

```
import pandas as pd
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.svm import SVC
```

```
classifier = SVC()
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1.0, 0.1, 0.01, 0.001], 'kernel': ['rbf', 'poly'],
```

```
pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
```

Fitting 5 folds for each of 48 candidates, totalling 240 fits

```
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
```

```
# IRIS PLANT DATASET
```

```
# SVM(With Tuning)[40-60 split]
```

```
import pandas as pd
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```



```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.svm import SVC
```

```
classifier = SVC()
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001], 'kernel': ['rbf', 'poly',
```

```
pprint(param_grid)
```

```
#####
```

```
from sklearn.model_selection import GridSearchCV
```

```
# Use the random grid to search for best hyperparameters
```

```
# First create the base model to tune
```

```
classifier = SVC()
```

```
# Random search of parameters, using 3 fold cross validation,
```

```
# search across 100 different combinations, and use all available cores
```

```
rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)
```

```
y_pred = rf_random.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
```

Fitting 5 folds for each of 48 candidates, totalling 240 fits

```
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
```

IRIS PLANT DATASET

SVM(With Tuning)[30-70 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)

y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=42)

Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

Classification

```
from sklearn.svm import SVC

classifier = SVC()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel': ['rbf', 'poly'],

pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import plot_confusion_matrix  
plot_confusion_matrix(rf_random, X_test, y_test)  
plt.show()
```

Parameters currently in use:

```
#####
```

```
    # Sepal Length, Sepal Width, Petal Length, Petal Width, Class
```

```
# IRIS PLANT DATASET
```

```
# Random Forest Classifier(Without Tuning)[70-30 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Finding the important parameters that contribute to most of the variance in the data.
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
from sklearn.decomposition import PCA
```

```
pca_test = PCA(n_components=4)
```

```
pca_test.fit(X_train)
```

```
sns.set(style='whitegrid')
```

```
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
```

```
plt.xlabel('number of components')
```

```
plt.ylabel('cumulative explained variance')
```

```
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
```

```
display(plt.show())
```

```
# So we can see that we have 10 important parameters
```

```
pca = PCA(n_components=2)
```

```
pca.fit(X_train)
```

```
X_train = pca.transfrom(X_train)
X_test = pca.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

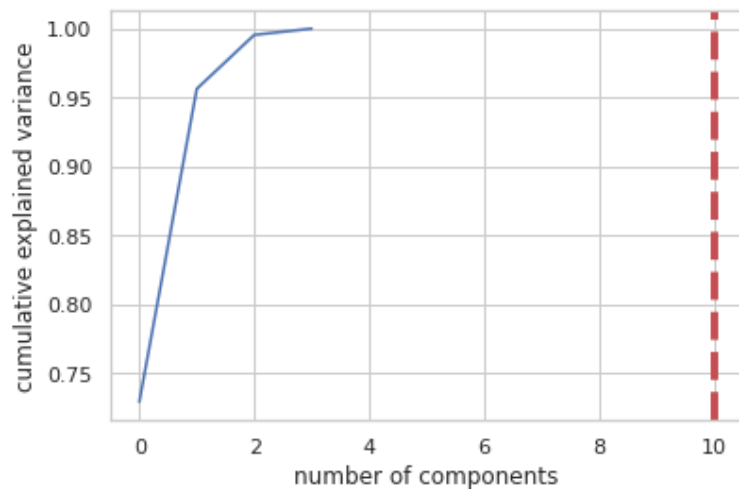
print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



None

Confusion Matrix:

```
[[14  0  0]
 [ 0 14  3]
 [ 0  0 14]]
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	14
Iris-versicolor	1.00	0.82	0.90	17
Iris-virginica	0.82	1.00	0.90	14
accuracy			0.93	45
macro avg	0.94	0.94	0.94	45
weighted avg	0.95	0.93	0.93	45

Accuracy:

IRIS PLANT DATASET

Random Forest Classifier(With Tuning)[70-30 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```



```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Finding the important parameters that contribute to most of the variance in the data.
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
from sklearn.decomposition import PCA
```

```
pca_test = PCA(n_components=4)
```

```
pca_test.fit(X_train)
```

```
sns.set(style='whitegrid')
```

```
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
```

```
plt.xlabel('number of components')
```

```
plt.ylabel('cumulative explained variance')
```

```
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
```

```
display(plt.show())
```

```
# So we can see that we have 10 important parameters
```

```
pca = PCA(n_components=2)
```

```
pca.fit(X_train)
```

```
X_train = pca.transform(X_train)
```

```
X_test = pca.transform(X_test)
```

```
# Classification
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier()
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
```

```
# Look at parameters used by our current forest
```

```
print('Parameters currently in use:\n')
```

```
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```
from sklearn.model_selection import RandomizedSearchCV
```

```
# Number of trees in random forest
```

```
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
```

```
# Number of features to consider at every split
```

```
max_features = ['auto', 'sqrt']
```

```

# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}
pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

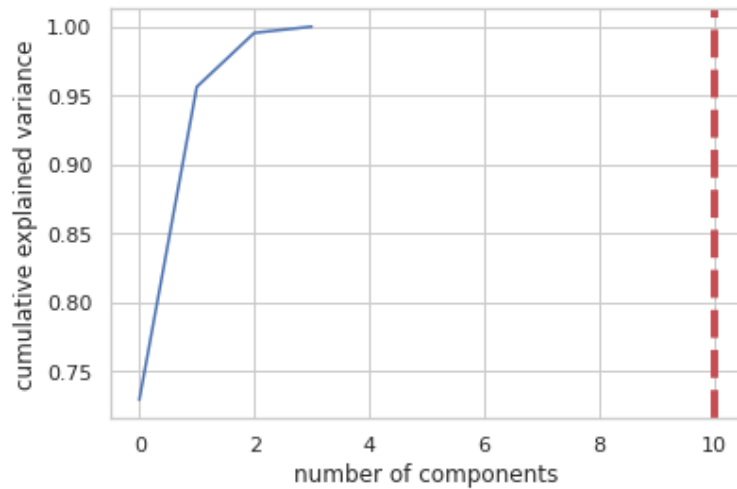
print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()

```



None

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 44.7s

[Parallel(n_jobs=-1)]: Done 158 tasks | elapsed: 3.0min

[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 5.8min finished

Confusion Matrix:

```
[[14  0  0]
 [ 0 14  3]
 [ 0  0 14]]
```

Performance Evaluation

	precision	recall	f1-score	support
Tris-setosa	1.00	1.00	1.00	14

IRIS PLANT DATASET

Multi Layer Perceptron(Without Tuning)[70-30 split]

```
import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=4)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=2)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier
```

```
classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

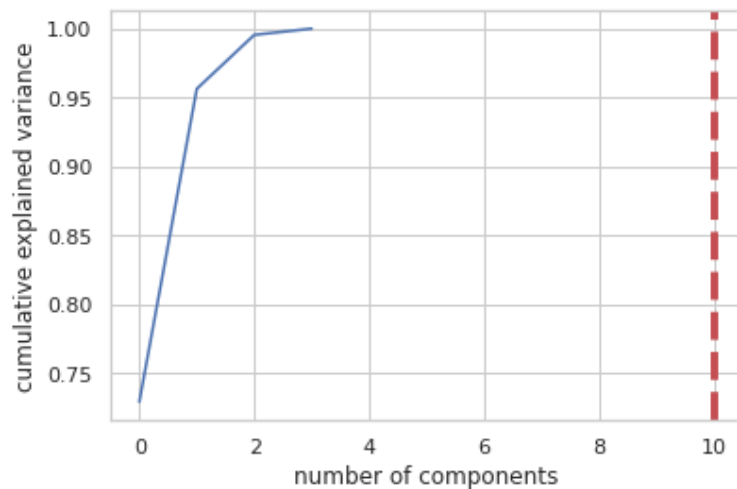
print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



None

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)
```

Confusion Matrix:

```
[[13  1  0]
 [ 0 14  3]
 [ 0  0 14]]
```

```
-----
-----
```

Performance Evaluation

	precision	recall	f1-score	support
Iris-setosa	1.00	0.93	0.96	14
Iris-versicolor	0.93	0.82	0.87	17

IRIS PLANT DATASET

Multi Layer Perceptron(With Tuning)[70-30 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

```

Finding the important parameters that contribute to most of the variance in the data.

```

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

```

```

pca_test = PCA(n_components=4)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

```

So we can see that we have 10 important parameters

```

pca = PCA(n_components=2)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

```

Classification

```

from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

```

```

#####
# Showing all the parameters

```

```

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

```

```

#####
# Creating a set of important sample features

```

```

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

```

```

#####

```

```

from sklearn.model_selection import GridSearchCV

```

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

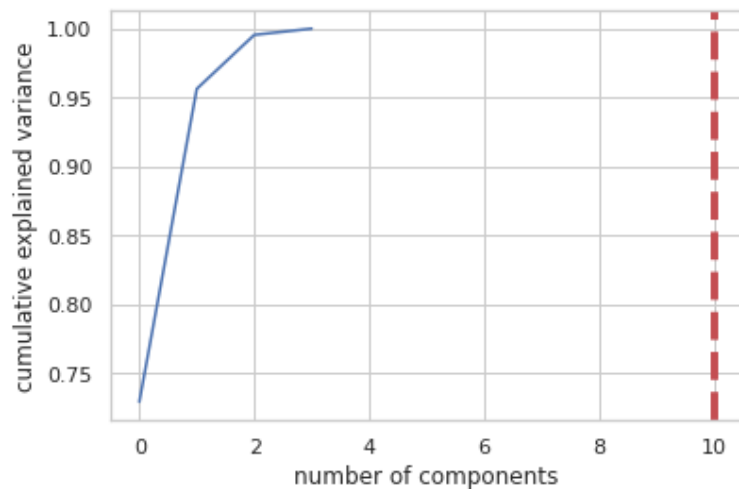
print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

None

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam'.
```

```
# IRIS PLANT DATASET
```

```
# SVM(Without Tuning)[70-30 split]
```

```
import pandas as pd
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("iris.data",header=None)
```

```
col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Finding the important parameters that contribute to most of the variance in the data.
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
from sklearn.decomposition import PCA
```

```
pca_test = PCA(n_components=4)
```

```
pca_test.fit(X_train)
```

```
sns.set(style='whitegrid')
```

```
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
```

```
plt.xlabel('number of components')
```

```
plt.ylabel('cumulative explained variance')
```

```
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
```

```
display(plt.show())
```

```
# So we can see that we have 10 important parameters
```

```
pca = PCA(n_components=2)
```

```
pca.fit(X_train)
```

```
X_train = pca.transform(X_train)
```

```
X_test = pca.transform(X_test)
```

```
# Classification
```

```
from sklearn.svm import SVC
```

```
classifier = SVC()
```

```
classifier.fit(X_train,y_train)
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Performance Evaluation")
```

```
print(classification_report(y_test, y_pred))
```

```
print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



```
# IRIS PLANT DATASET
# SVM(With Tuning)[70-30 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("iris.data",header=None)

col_name = ['Sepal Length','Sepal Width','Petal Length','Petal Width','Class']

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=4)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters
```

```
pca = PCA(n_components=2)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel': ['rbf', 'poly'],

pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

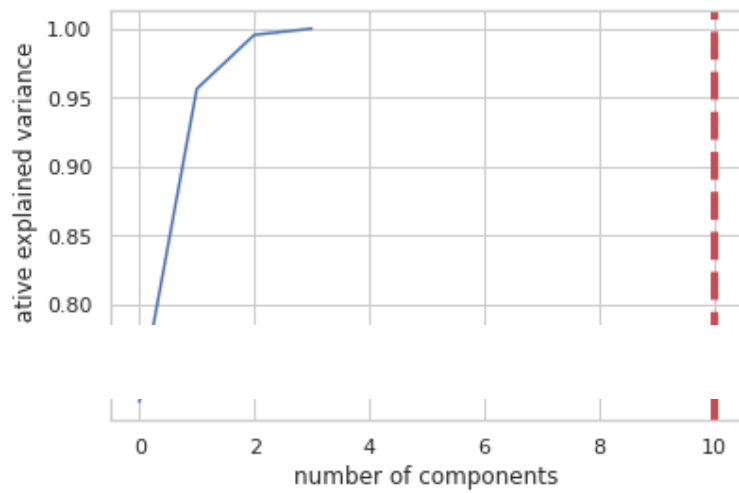
print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-----")  
print("-----")
```

```
print("Accuracy:")  
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt  
from sklearn.metrics import plot_confusion_matrix  
plot_confusion_matrix(rf_random, X_test, y_test)  
plt.show()
```



None

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
```

● ×

```

# WINE DATASET
# Random Forest Classifier(Without Tuning)[70-30 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,randome

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")

```



```
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



Confusion Matrix:

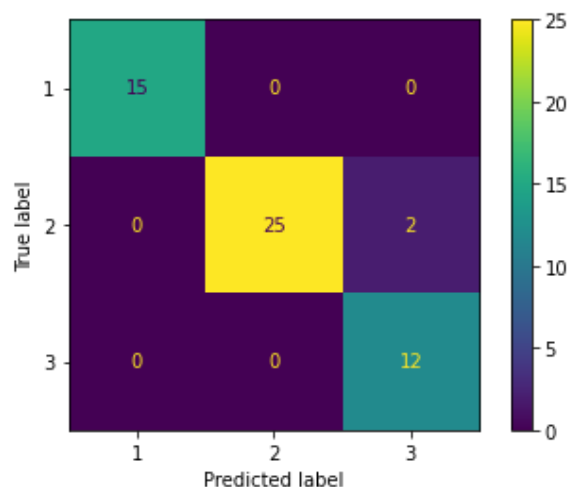
```
[[15  0  0]
 [ 0 25  2]
 [ 0  0 12]]
```

Performance Evaluation

	precision	recall	f1-score	support
1	1.00	1.00	1.00	15
2	1.00	0.93	0.96	27
3	0.86	1.00	0.92	12
accuracy			0.96	54
macro avg	0.95	0.98	0.96	54
weighted avg	0.97	0.96	0.96	54

Accuracy:

0.9629629629629629



WINE DATASET

Random Forest Classifier(Without Tuning)[60-40 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class', 'Alcohol', 'Malic acid', 'Ash', 'Alkalinity of ash', 'Magnesium', 'Total ph
```

```
col_name = [ 'Class' , 'Alcohol' , 'Malic acid' , 'Ash' , 'Alkalinity of ash' , 'Magnesium' , 'Total phenols' ,
              'Nonflavanoid phenols' , 'Proanthocyanins' , 'Color intensity' , 'Hue' , 'OD280/OD315' ]
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=0)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier(n_estimators=20, random_state=0)
```

```
classifier.fit(X_train,y_train)
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Performance Evaluation")
```

```
print(classification_report(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Accuracy:")
```

```
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import plot_confusion_matrix
```

```
plot_confusion_matrix(classifier, X_test, y_test)
```

```
plt.show()
```

Confusion Matrix:

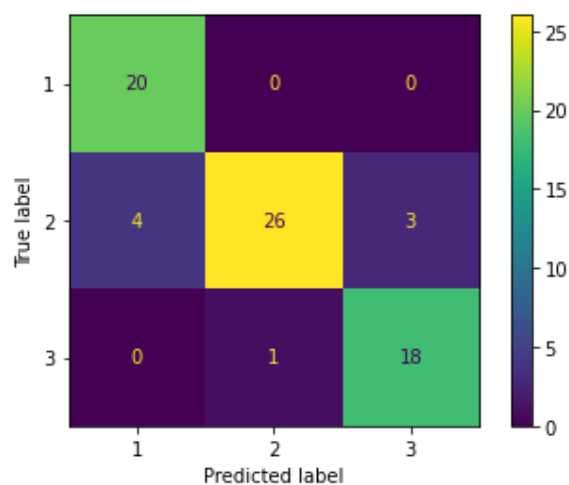
```
[[20  0  0]
 [ 4 26  3]
 [ 0  1 18]]
```

Performance Evaluation

	precision	recall	f1-score	support
1	0.83	1.00	0.91	20
2	0.96	0.79	0.87	33
3	0.86	0.95	0.90	19
accuracy			0.89	72
macro avg	0.88	0.91	0.89	72
weighted avg	0.90	0.89	0.89	72

Accuracy:

0.8888888888888888



WINE DATASET

Random Forest Classifier(Without Tuning)[50-50 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
y = y_train
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,random_state=0)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier(n_estimators=20, random_state=0)
```

```
classifier.fit(X_train,y_train)
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Performance Evaluation")
```

```
print(classification_report(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Accuracy:")
```

```
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import plot_confusion_matrix
```

```
plot_confusion_matrix(classifier, X_test, y_test)
```

```
plt.show()
```

Confusion Matrix:

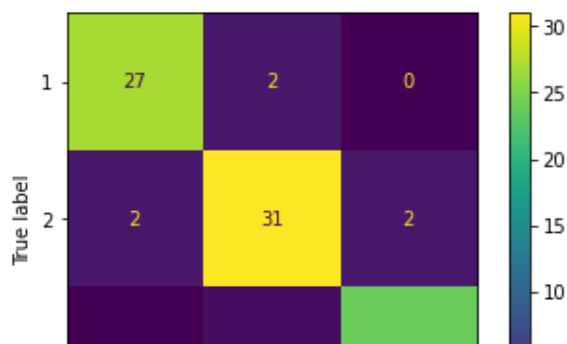
```
[[27  2  0]
 [ 2 31  2]
 [ 0  1 24]]
```

Performance Evaluation

	precision	recall	f1-score	support
1	0.93	0.93	0.93	29
2	0.91	0.89	0.90	35
3	0.92	0.96	0.94	25
accuracy			0.92	89
macro avg	0.92	0.93	0.92	89
weighted avg	0.92	0.92	0.92	89

Accuracy:

0.9213483146067416



WINE DATASET

Random Forest Classifier(Without Tuning)[40-60 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

```
[[30  4  0]
 [ 6 34  2]
 [ 0  1 30]]
```

Performance Evaluation

	precision	recall	f1-score	support
1	0.83	0.88	0.86	34
2	0.87	0.81	0.84	42
3	0.94	0.97	0.95	31
accuracy			0.88	107
macro avg	0.88	0.89	0.88	107
weighted avg	0.88	0.88	0.88	107

Accuracy:

0.8785046728971962



WINE DATASET

Random Forest Classifier(Without Tuning)[30-70 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,randome
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

Classification

```
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```


Confusion Matrix:

```
[[35  6  0]
 [ 1 47  1]
 [ 0  0 35]]
```

Performance Evaluation

	precision	recall	f1-score	support
1	0.97	0.85	0.91	41
2	0.89	0.96	0.92	49
3	0.97	1.00	0.99	35
accuracy			0.94	125
macro avg	0.94	0.94	0.94	125

WINE DATASET

Random Forest Classifier(With Tuning)[70-30 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Classification

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier()
```

```
#####
```

Showing all the parameters

```
from pprint import pprint
```

```

# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}
pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

```

```
print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0}
```

WINE DATASET

Random Forest Classifier(With Tuning)[60-40 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Classification

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier()
```

```
#####
```

Showing all the parameters

```
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}
pprint(random_grid)
```

```
#####
```

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)
```

```
y_pred = rf_random.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Accuracy:")  
print(accuracy_score(y_test, y_pred))  
  
import matplotlib.pyplot as plt  
from sklearn.metrics import plot_confusion_matrix  
plot_confusion_matrix(rf_random, X_test, y_test)  
plt.show()
```

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 ... ..}
```

WINE DATASET

Random Forest Classifier(With Tuning)[50-50 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,randome
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Classification

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier()
```

```
#####
```

Showing all the parameters

```
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())
```

```
#####
```

Creating a set of important sample features

```
from sklearn.model_selection import RandomizedSearchCV
```

```

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}

pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

```



```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'bootstrap': True,
```

```
# WINE DATASET
```

```
# Random Forest Classifier(With Tuning)[40-60 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph  
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=42)
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier()
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
```

```
# Look at parameters used by our current forest
```

```
print('Parameters currently in use:\n')
```

```
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```
from sklearn.model_selection import RandomizedSearchCV
```

```
# Number of trees in random forest
```

```
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
```

```

# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}
pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

```

```
plot_confusion_matrix(rf_random, X_test, y_test)  
plt.show()
```

```

# WINE DATASET
# Random Forest Classifier(With Tuning)[30-70 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.,random

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree

```

```

max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}
pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()

```

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 51.3s

[Parallel(n_jobs=-1)]: Done 158 tasks | elapsed: 3.5min

[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 6.8min finished

Confusion Matrix:

```
[[35  6  0]
 [ 1 46  2]
 [ 0  1 34]]
```

Performance Evaluation

	precision	recall	f1-score	support
1	0.97	0.85	0.91	41
2	0.87	0.94	0.90	49
3	0.94	0.97	0.96	35
accuracy			0.92	125
macro avg	0.93	0.92	0.92	125
weighted avg	0.92	0.92	0.92	125

Accuracy:

0.92



#####

```
# WINE DATASET
# Multi Layer Perceptron(Without Tuning)[70-30 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
```



```
print("-----")
```

```
print("Accuracy:")
```

```
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import plot_confusion_matrix
```

```
plot_confusion_matrix(classifier, X_test, y_test)
```

```
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
```

```
% self.max_iter, ConvergenceWarning)
```

```
Confusion Matrix:
```

```
[[15  0  0]
```

```
 [ 1 24  2]
```

```
 [ 0  0 12]]
```

```
-----
```

```
-----
```

```
Performance Evaluation
```

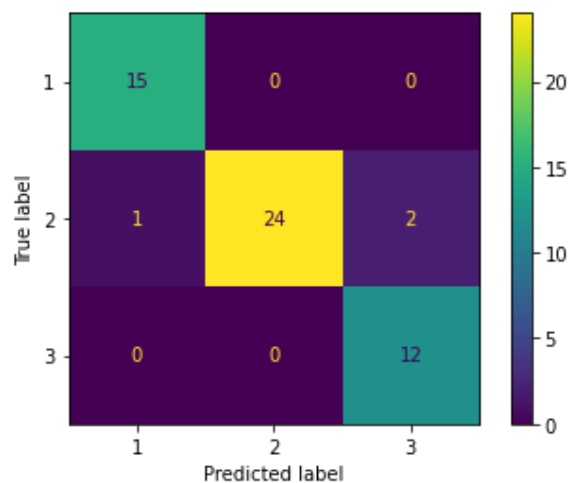
	precision	recall	f1-score	support
1	0.94	1.00	0.97	15
2	1.00	0.89	0.94	27
3	0.86	1.00	0.92	12
accuracy			0.94	54
macro avg	0.93	0.96	0.94	54
weighted avg	0.95	0.94	0.94	54

```
-----
```

```
-----
```

```
Accuracy:
```

```
0.9444444444444444
```



```
# WINE DATASET
```

```
# Multi Layer Perceptron(Without Tuning)[60-40 split]
```

```
import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,randome

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)
```

Confusion Matrix:

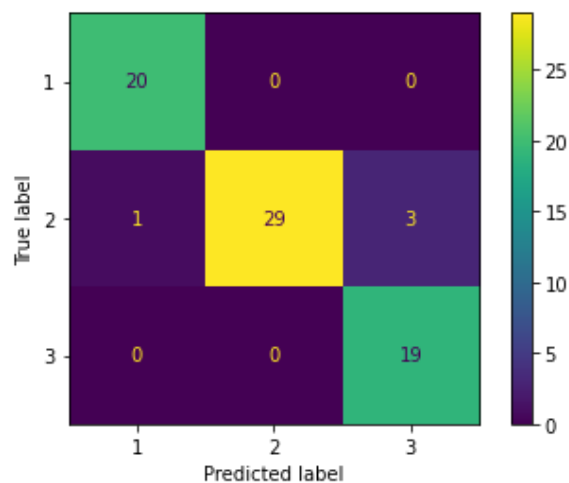
```
[[20  0  0]
 [ 1 29  3]
 [ 0  0 19]]
```

Performance Evaluation

	precision	recall	f1-score	support
1	0.95	1.00	0.98	20
2	1.00	0.88	0.94	33
3	0.86	1.00	0.93	19
accuracy			0.94	72
macro avg	0.94	0.96	0.95	72
weighted avg	0.95	0.94	0.94	72

Accuracy:

0.9444444444444444



WINE DATASET

Multi Layer Perceptron(Without Tuning)[50-50 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
```

```
% self.max_iter, ConvergenceWarning)
```

```
Confusion Matrix:
```

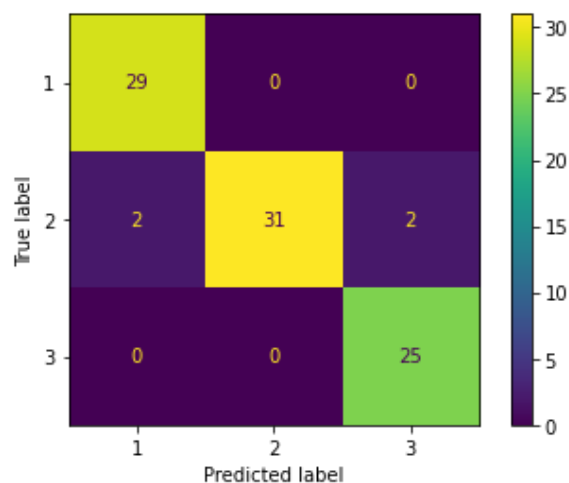
```
[[29  0  0]
 [ 2 31  2]
 [ 0  0 25]]
```

```
Performance Evaluation
```

	precision	recall	f1-score	support
1	0.94	1.00	0.97	29
2	1.00	0.89	0.94	35
3	0.93	1.00	0.96	25
accuracy			0.96	89
macro avg	0.95	0.96	0.96	89
weighted avg	0.96	0.96	0.95	89

```
Accuracy:
```

```
0.9550561797752809
```



```
# WINE DATASET
```

```
# Multi Layer Perceptron(Without Tuning)[40-60 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']
```

```
df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
```

```
% self.max_iter, ConvergenceWarning)
```

```
Confusion Matrix:
```

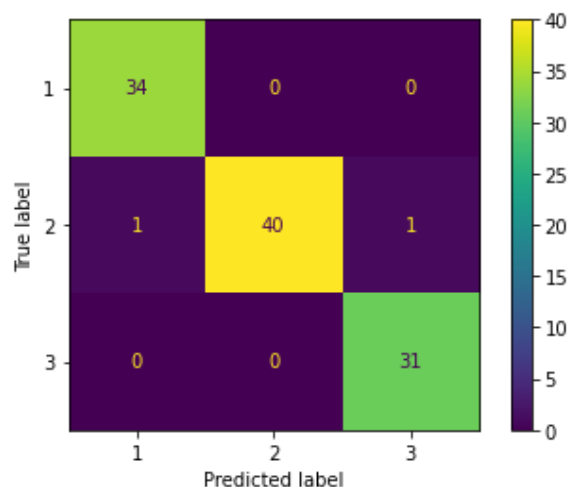
```
[[34  0  0]
 [ 1 40  1]
 [ 0  0 31]]
```

```
Performance Evaluation
```

	precision	recall	f1-score	support
1	0.97	1.00	0.99	34
2	1.00	0.95	0.98	42
3	0.97	1.00	0.98	31
accuracy			0.98	107
macro avg	0.98	0.98	0.98	107
weighted avg	0.98	0.98	0.98	107

```
Accuracy:
```

```
0.9813084112149533
```



```
# WINE DATASET
```

```
# Multi Layer Perceptron(Without Tuning)[30-70 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph  
'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
y = utl class ]

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```


Confusion Matrix:

```
[[41  0  0]
 [ 1 46  2]
 [ 0  1 34]]
```

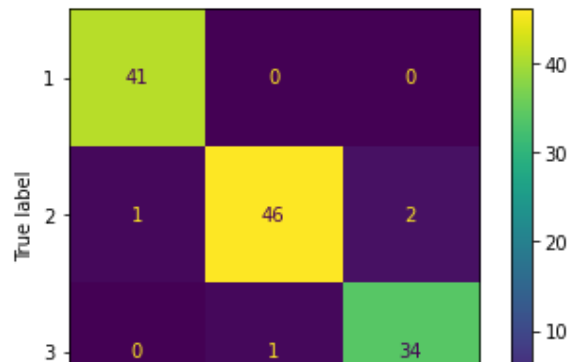
Performance Evaluation

	precision	recall	f1-score	support
1	0.98	1.00	0.99	41
2	0.98	0.94	0.96	49
3	0.94	0.97	0.96	35
accuracy			0.97	125
macro avg	0.97	0.97	0.97	125
weighted avg	0.97	0.97	0.97	125

Accuracy:

0.968

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)



WINE DATASET

Multi Layer Perceptron(With Tuning)[70-30 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph  
'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)
```

```

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True.}
```

WINE DATASET

Multi Layer Perceptron(With Tuning)[60-40 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Classification

```
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)
```

```
#####
# Showing all the parameters
```

```

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant', 'adaptive'],
}
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)

```

```
plt.show()
```

```

# WINE DATASET
# Multi Layer Perceptron(With Tuning)[50-50 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,rando

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

```

```
#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```


Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
{'activation': ['tanh', 'relu'],
 'alpha': [0.0001, 0.05],
 'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)],
 'learning_rate': ['constant', 'adaptive'],
 'solver': ['sgd', 'adam']}
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
% self.max_iter, ConvergenceWarning)
Confusion Matrix:
[[27  2  0]
 [ 3 29  3]
 [ 0  0 25]]
```

Performance Evaluation

	precision	recall	f1-score	support
1	0.90	0.93	0.92	29
2	0.94	0.83	0.88	35

WINE DATASET

Multi Layer Perceptron(With Tuning)[40-60 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
```

WINE DATASET

Multi Layer Perceptron(With Tuning)[30-70 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']

df.columns = col_name

X = df.drop(['Class'], axis=1)

y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=42)

Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

```

# Classification
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

```

```
print("Accuracy:")  
print(accuracy_score(y_test, y_pred))  
  
import matplotlib.pyplot as plt  
from sklearn.metrics import plot_confusion_matrix  
plot_confusion_matrix(rf_random, X_test, y_test)  
plt.show()
```

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False.
```

```
#####
```

```
    'learning_rate': 'constant'.
```

```
# WINE DATASET
```

```
# SVM(Without Tuning)[70-30 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,rando
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.svm import SVC
```

```
classifier = SVC()
```

```
classifier.fit(X_train,y_train)
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

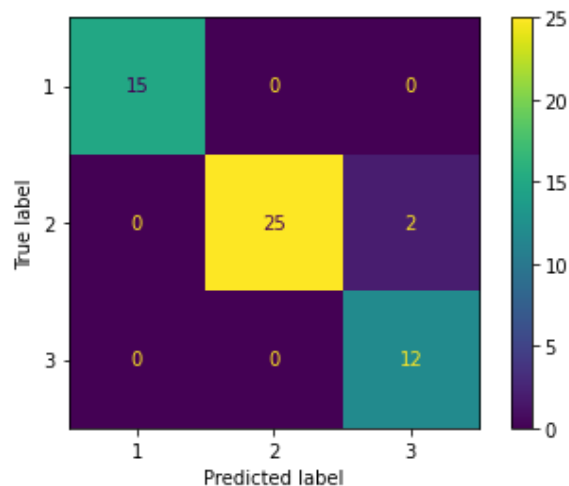
```
[[15  0  0]
 [ 0 25  2]
 [ 0  0 12]]
```

Performance Evaluation

	precision	recall	f1-score	support
1	1.00	1.00	1.00	15
2	1.00	0.93	0.96	27
3	0.86	1.00	0.92	12
accuracy			0.96	54
macro avg	0.95	0.98	0.96	54
weighted avg	0.97	0.96	0.96	54

Accuracy:

0.9629629629629629




```
# WINE DATASET
# SVM(Without Tuning)[60-40 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,randome

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

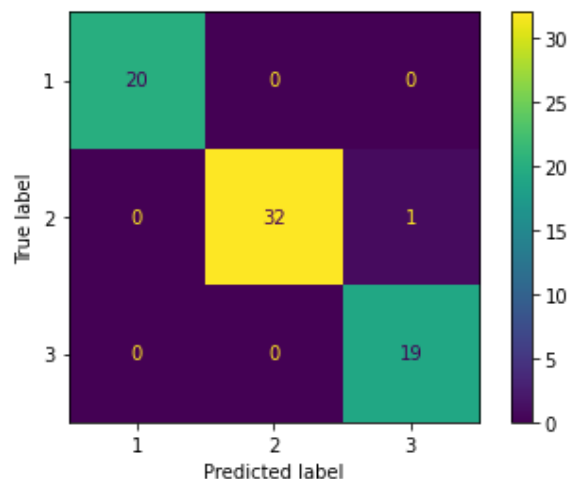
```
[[20  0  0]
 [ 0 32  1]
 [ 0  0 19]]
```

Performance Evaluation

	precision	recall	f1-score	support
1	1.00	1.00	1.00	20
2	1.00	0.97	0.98	33
3	0.95	1.00	0.97	19
accuracy			0.99	72
macro avg	0.98	0.99	0.99	72
weighted avg	0.99	0.99	0.99	72

Accuracy:

0.9861111111111112



```
# WINE DATASET
# SVM(Without Tuning)[50-50 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph  
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,random
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.svm import SVC
```

```
classifier = SVC()
```

```
classifier.fit(X_train,y_train)
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Performance Evaluation")
```

```
print(classification_report(y_test, y_pred))
```

```
print("-----")
```

```
print("-----")
```

```
print("Accuracy:")
```

```
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import plot_confusion_matrix
```

```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

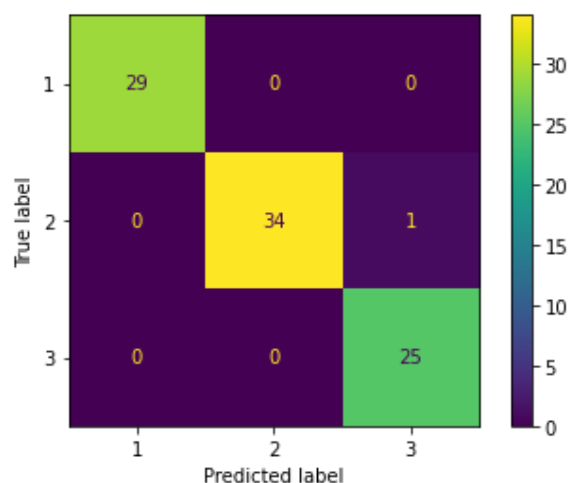
```
[[29  0  0]
 [ 0 34  1]
 [ 0  0 25]]
```

Performance Evaluation

	precision	recall	f1-score	support
1	1.00	1.00	1.00	29
2	1.00	0.97	0.99	35
3	0.96	1.00	0.98	25
accuracy			0.99	89
macro avg	0.99	0.99	0.99	89
weighted avg	0.99	0.99	0.99	89

Accuracy:

0.9887640449438202



WINE DATASET

SVM(Without Tuning)[40-60 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data", header=None)
```

```
col_name = ['Class', 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium', 'Total ph',
            'Nonflavanoid phenols', 'Proanthocyanins', 'Color intensity', 'Hue', 'OD280/OD315']
```

```
df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

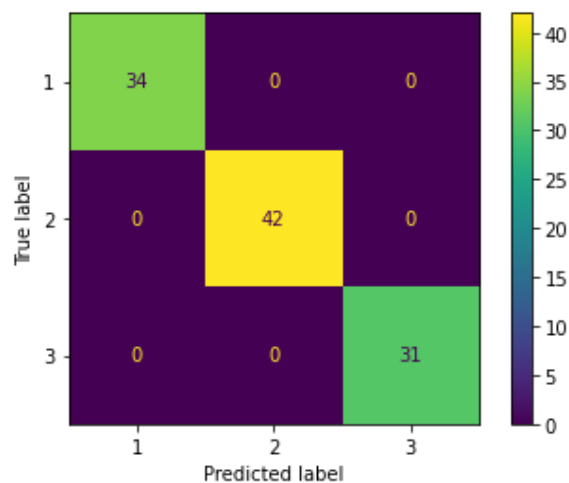
```
[[34  0  0]
 [ 0 42  0]
 [ 0  0 31]]
```

Performance Evaluation

	precision	recall	f1-score	support
1	1.00	1.00	1.00	34
2	1.00	1.00	1.00	42
3	1.00	1.00	1.00	31
accuracy			1.00	107
macro avg	1.00	1.00	1.00	107
weighted avg	1.00	1.00	1.00	107

Accuracy:

1.0



WINE DATASET

SVM(Without Tuning)[30-70 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

Confusion Matrix:

```
[[38  3  0]
 [ 0 49  0]
 [ 0  2 33]]
```

Performance Evaluation

	precision	recall	f1-score	support
1	1.00	0.93	0.96	41
2	0.91	1.00	0.95	49
3	1.00	0.94	0.97	35
accuracy			0.96	125
macro avg	0.97	0.96	0.96	125
weighted avg	0.96	0.96	0.96	125

Accuracy:

0.96



WINE DATASET

SVM(With Tuning)[70-30 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,randome
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```



```
X_test = sc.transform(X_test)
```

```
# Classification
from sklearn.svm import SVC
```

```
classifier = SVC()
```

```
#####
# Showing all the parameters
```

```
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())
```

```
#####
# Creating a set of important sample features
```

```
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001], 'kernel': ['rbf', 'poly'],

pprint(param_grid)
```

```
#####
from sklearn.model_selection import GridSearchCV
```

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
```

```
rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)
```

```
y_pred = rf_random.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Accuracy:")
```

```
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
```

WINE DATASET

SVM(With Tuning)[60-40 split]

```
import pandas as pd
```

```
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.6,test_size=0.4,rando
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

Classification

```
from sklearn.svm import SVC
```

```
classifier = SVC()
```

```
#####
# Showing all the parameters
```

```
from pprint import pprint
```

```
# Look at parameters used by our current forest
```

```
print('Parameters currently in use:\n')
```

```
pprint(classifier.get_params())
```

```
#####
# Creating a set of important sample features
```

```
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly'],

pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
```

Fitting 5 folds for each of 48 candidates, totalling 240 fits

```
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
```

WINE DATASET

SVM(With Tuning)[50-50 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315
```

```
df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.5,test_size=0.5,random_state=42)

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly'],

pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
```

Fitting 5 folds for each of 48 candidates, totalling 240 fits

```
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
```

WINE DATASET

SVM(With Tuning)[40-60 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.4,test_size=0.6,randome
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```



```

# Classification
from sklearn.svm import SVC

classifier = SVC()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel': ['rbf', 'poly'],

pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200.
```

```
# WINE DATASET
```

```
# SVM(With Tuning)[30-70 split]
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Dataset Preparation
```

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
```

```
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.3,test_size=0.7,randome
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
# Classification
```

```
from sklearn.svm import SVC
```

```
classifier = SVC()
```

```
#####
```

```
# Showing all the parameters
```

```
from pprint import pprint
```

```
# Look at parameters used by our current forest
```

```
print('Parameters currently in use:\n')
```

```
pprint(classifier.get_params())
```

```
#####
```

```
# Creating a set of important sample features
```

```
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly',
```

```
pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
{'C': [0.1, 1, 10, 100],
 'gamma': [1, 0.1, 0.01, 0.001],
 'kernel': ['rbf', 'poly', 'sigmoid']}
```

Fitting 5 folds for each of 48 candidates, totalling 240 fits

```
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....
[CV] ..... C=0.1, gamma=1, kernel=rbf, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=poly .....
[CV] ..... C=0.1, gamma=1, kernel=poly, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
[CV] C=0.1, gamma=1, kernel=sigmoid .....
[CV] ..... C=0.1, gamma=1, kernel=sigmoid, total= 0.0s
```

#####

```
[CV] C=0.1, gamma=1, kernel=sigmoid .....
```

WINE DATASET

Random Forest Classifier(Without Tuning)[70-30 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph

```
'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,randome
```

```
# Feature Scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Finding the important parameters that contribute to most of the variance in the data.
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
```

```
pca_test = PCA(n_components=13)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())
```

```
# So we can see that we have 10 important parameters
```

```
pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
```

```
# Classification
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier()
classifier.fit(X_train, y_train)
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

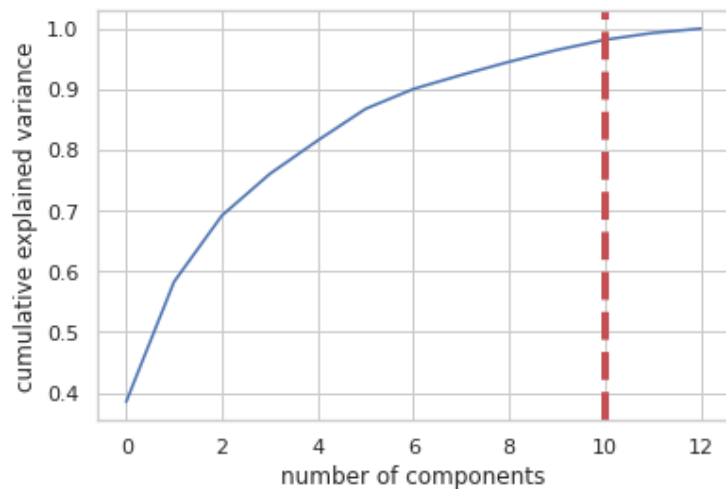
print("-----")
print("-----")

print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



None

Confusion Matrix:

```
[[15  0  0]
 [ 2 23  2]
 [ 0  0 12]]
```


 Performance Evaluation

WINE DATASET

Random Forest Classifier(With Tuning)[70-30 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```



```
# Finding the important parameters that contribute to most of the variance in the data.
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=13)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())
```

```
# So we can see that we have 10 important parameters
```

```
pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
```

```
# Classification
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier()
```

```
#####
# Showing all the parameters
```

```
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())
```

```
#####
# Creating a set of important sample features
```

```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
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
random_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}
pprint(random_grid)

#####

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = classifier, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

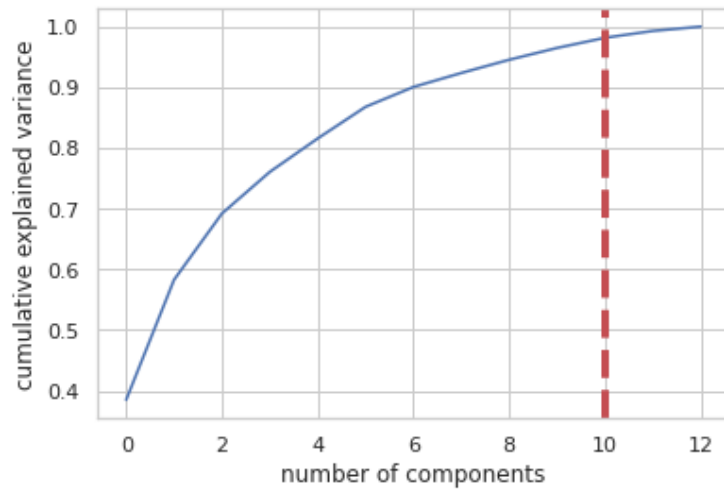
print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()

```



None

Parameters currently in use:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers
```

WINE DATASET

Multi Layer Perceptron(Without Tuning)[70-30 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']
```

```

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.30,rand

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=13)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification using MLP
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\n-----")

```

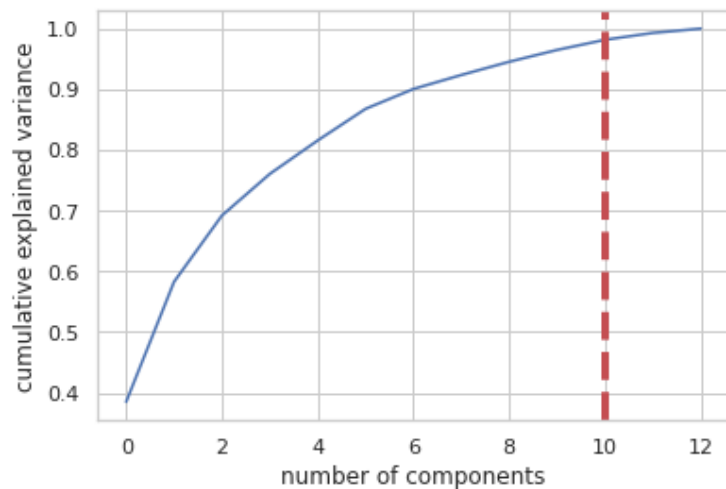
```
print("\n-----")
```

```
print("Performance Evaluation")
print(classification_report(y_test, y_pred))
```

```
print("-----")
print("-----")
```

```
print("Accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



None

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron

WINE DATASET

Multi Layer Perceptron(With Tuning)[70-30 split]

import pandas as pd

import numpy as np

Dataset Preparation

df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']

df.columns = col_name

X = df.drop(['Class'], axis=1)

y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,random_state=42)

Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.decomposition import PCA

```
pca_test = PCA(n_components=13)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())
```

So we can see that we have 10 important parameters

```
pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
```

Classification

```
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(max_iter=100)
```

```
#####
# Showing all the parameters
```

```
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())
```

```
#####
# Creating a set of important sample features
```

```
parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
pprint(parameter_space)
```

```
#####
```

```
from sklearn.model_selection import GridSearchCV
```

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = MLPClassifier(max_iter=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
```

```
rf_random = GridSearchCV(classifier, parameter_space, n_jobs=-1, cv=3)
rf_random.fit(X_train, y_train)
```

```
rf_random.predict(X_test)
```

```
y_pred = rf_random.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
print("Confusion Matrix:")  
print(confusion_matrix(y_test, y_pred))
```

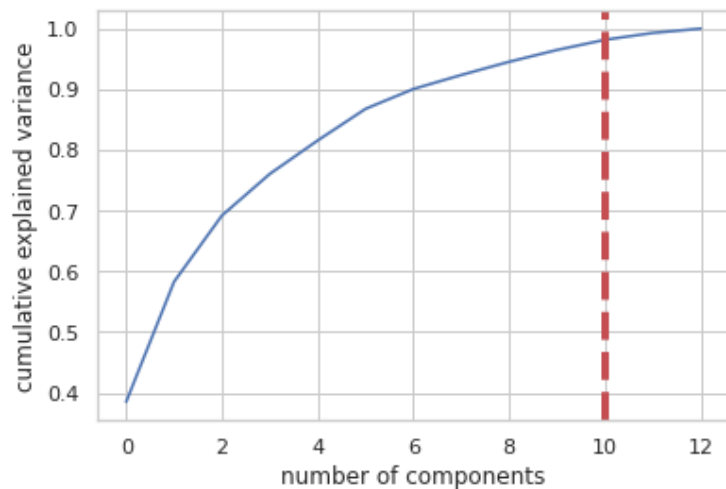
```
print("-----")  
print("-----")
```

```
print("Performance Evaluation")  
print(classification_report(y_test, y_pred))
```

```
print("-----")  
print("-----")
```

```
print("Accuracy:")  
print(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt  
from sklearn.metrics import plot_confusion_matrix  
plot_confusion_matrix(rf_random, X_test, y_test)  
plt.show()
```

None

Parameters currently in use:

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,,)
```

WINE DATASET

SVM(Without Tuning)[70-30 split]

```
import pandas as pd
import numpy as np
```

Dataset Preparation

```
df = pd.read_csv("wine.data",header=None)
```

```
col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315']
```

```
df.columns = col_name
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,randome
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)

# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=13)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Classification
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

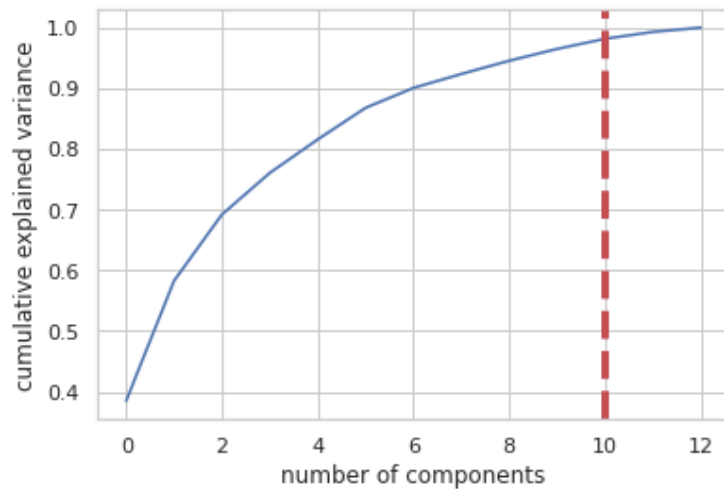
print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
```

```
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



None

Confusion Matrix:

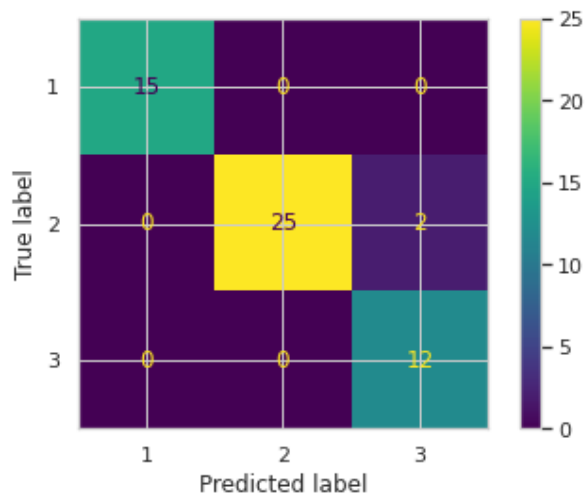
```
[[15  0  0]
 [ 0 25  2]
 [ 0  0 12]]
```

Performance Evaluation

	precision	recall	f1-score	support
1	1.00	1.00	1.00	15
2	1.00	0.93	0.96	27
3	0.86	1.00	0.92	12
accuracy			0.96	54
macro avg	0.95	0.98	0.96	54
weighted avg	0.97	0.96	0.96	54

Accuracy:

0.9629629629629629



```
# WINE DATASET
# SVM(With Tuning)[70-30 split]

import pandas as pd
import numpy as np

# Dataset Preparation
df = pd.read_csv("wine.data",header=None)

col_name = ['Class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total ph
            'Nonflavanoid phenols','Proanthocyanins','Color intensity','Hue','OD280/OD315

df.columns = col_name

X = df.drop(['Class'], axis=1)
y = df['Class']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7,test_size=0.3,randome

# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Finding the important parameters that contribute to most of the variance in the data.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

pca_test = PCA(n_components=13)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())

# So we can see that we have 10 important parameters

pca = PCA(n_components=10)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
```

```

# Classification
from sklearn.svm import SVC

classifier = SVC()

#####
# Showing all the parameters

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(classifier.get_params())

#####
# Creating a set of important sample features

param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel': ['rbf', 'poly'],

pprint(param_grid)

#####

from sklearn.model_selection import GridSearchCV

# Use the random grid to search for best hyperparameters
# First create the base model to tune
classifier = SVC()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores

rf_random = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
rf_random.fit(X_train, y_train)

y_pred = rf_random.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("-----")
print("-----")

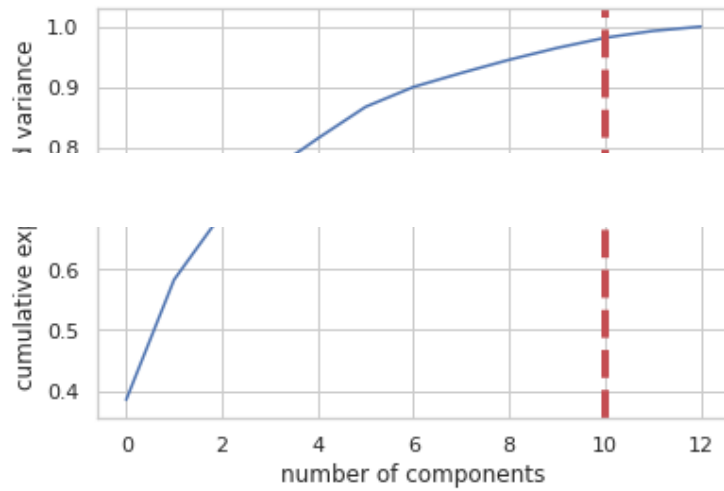
print("Performance Evaluation")
print(classification_report(y_test, y_pred))

print("-----")
print("-----")

print("Accuracy:")
print(accuracy_score(y_test, y_pred))

```

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_random, X_test, y_test)
plt.show()
```



None

Parameters currently in use:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True.
```

