Open in Colab

In [1]:
```
# Import libraries
import numpy as np
import pandas as pd
```

In [2]:
```
# Import dataset
url1 = 'https://raw.githubusercontent.com/andrerizzo/complete_project/dataset/iris_pred
url2 = 'https://raw.githubusercontent.com/andrerizzo/complete_project/dataset/iris_resp
X = pd.read_csv(url1)
y = pd.read_csv(url2)

print(X)
print(y)
```

```
     sepal_width  petal_length_trans
0            3.0           -1.222611
1            3.2           -1.641609
2            3.1           -0.860957
3            3.6           -1.222611
4            3.9           -0.518618
..           ...                 ...
144          3.0            0.767216
145          2.5            0.537961
146          3.0            0.767216
147          3.4            0.860957
148          3.0            0.658733

[149 rows x 2 columns]
             class
0      Iris-setosa
1      Iris-setosa
2      Iris-setosa
3      Iris-setosa
4      Iris-setosa
..             ...
144  Iris-virginica
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica

[149 rows x 1 columns]
```

In [3]:
```
# Handling Categorical Variables

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
y = encoder.fit_transform(y.values.ravel())

print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2]
```

In [4]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0
print(X_train)
print(X_test)
print(y_train)
print(y_test)
```

```
     sepal_width  petal_length_trans
27           3.4           -1.222611
97           2.5           -0.433806
96           2.9           -0.025237
69           3.2            0.369964
18           3.8           -0.860957
..           ...                 ...
9            3.7           -0.860957
103          3.0            1.297001
67           2.2            0.126509
117          2.6            2.710835
47           3.7           -0.860957

[119 rows x 2 columns]
     sepal_width  petal_length_trans
133          2.6            1.046561
109          3.2            0.658733
59           2.0           -0.369964
80           2.4           -0.325284
7            2.9           -1.222611
104          3.0            1.986482
140          3.1            0.658733
95           2.9           -0.075776
118          2.2            0.537961
84           3.4            0.126509
33           3.1           -0.860957
44           3.0           -1.222611
54           2.8            0.126509
24           3.0           -0.627699
37           3.0           -1.641609
132          2.8            0.658733
111          3.0            0.923581
73           2.9           -0.025237
16           3.5           -1.222611
45           3.8           -0.627699
40           2.3           -1.641609
8            3.1           -0.860957
85           3.1            0.290005
22           3.3           -0.518618
62           2.9            0.290005
94           3.0           -0.075776
90           3.0            0.220472
26           3.5           -0.860957
43           3.8           -0.461702
134          3.0            1.641609
[0 1 1 1 0 2 2 1 1 0 2 0 1 2 2 2 1 1 1 2 2 1 2 2 1 2 1 0 1 1 1 1 2 1 2 0 0
 2 1 0 0 1 0 2 1 0 1 2 1 0 2 2 2 2 0 0 2 2 0 2 0 2 2 0 0 2 0 0 0 1 2 2 0 0
 0 1 1 0 0 1 0 2 1 2 1 0 2 0 2 0 0 2 1 2 2 1 1 2 1 2 2 1 1 0 1 2 2 0 1 1 1
 1 0 0 0 2 1 2 0]
[2 2 1 1 0 2 2 1 2 1 0 0 1 0 0 2 2 1 0 0 0 0 1 0 1 1 1 0 0 2]
```

In [5]:
```
# Create dictionary to store accuracy results
results = {}
```

In [6]:
```
# Evaluate Models - LDA

# Import libraries
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

# Define model
classifier = LDA()

# Define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# Evaluate model
scores = 0
scores = cross_val_score(classifier, X_train, y_train, cv=cv, scoring='accuracy', n_job

print("Linear Discriminant Analysis model accuracy: %.3f (%.3f)" % (np.mean(scores),np.

# Update dictionary from accuracy results
# results = {}
results['LDA'] = np.mean(scores)
```

```
Linear Discriminant Analysis model accuracy: 0.913 (0.095)
```

In [7]:
```
# Evaluate Models - kNN

# Import libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

# Define model
classifier = KNeighborsClassifier()

# Define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# Evaluate model
scores= 0
scores = cross_val_score(classifier, X_train, y_train, cv=cv, scoring='accuracy', n_job

print("k-Nearest Neighbors model accuracy: %.3f (%.3f)" % (np.mean(scores),np.std(score

# Update dictionary from accuracy results
# results = {}
results['kNN'] = np.mean(scores)
```

```
k-Nearest Neighbors model accuracy: 0.936 (0.089)
```

In [8]:
```
# Evaluate Models - SVC

# Import libraries
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

# Define model
```

```
classifier = SVC(kernel='linear')

# Define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# Evaluate model
scores = 0
scores = cross_val_score(classifier, X_train, y_train, cv=cv, scoring='accuracy', n_job

print("Support Vector Classifier model accuracy: %.3f (%.3f)" % (np.mean(scores),np.std

# Update dictionary from accuracy results
# results = {}
results['SVC'] = np.mean(scores)
```

```
Support Vector Classifier model accuracy: 0.941 (0.084)
```

In [9]:

```
# Evaluate Models - SGD

# Import libraries
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

# Define model
classifier = SGDClassifier()

# Define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# Evaluate model
scores = 0
scores = cross_val_score(classifier, X_train, y_train, cv=cv, scoring='accuracy', n_job

print("Stochastic Gradient Descend model accuracy: %.3f (%.3f)" % (np.mean(scores),np.s

# Update dictionary from accuracy results
# results = {}
results['SGD'] = np.mean(scores)
```

```
Stochastic Gradient Descend model accuracy: 0.844 (0.093)
```

In [10]:

```
# Evaluate Models - Gaussian Naive Bayes

# Import libraries
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

# Define model
classifier = GaussianNB()

# Define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# Evaluate model
scores = 0
scores = cross_val_score(classifier, X_train, y_train, cv=cv, scoring='accuracy', n_job
```

```
print("Naive Bayes model accuracy: %.3f (%.3f)" % (np.mean(scores),np.std(scores)), "\n

# Update dictionary from accuracy results
# results = {}
results['Gaussian Naive Bayes'] = np.mean(scores)
```

```
Naive Bayes model accuracy: 0.899 (0.095)
```

In [11]:

```
# Evaluate Models - Decision Tree

# Import libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

# Define model
classifier = DecisionTreeClassifier()

# Define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# Evaluate model
scores = 0
scores = cross_val_score(classifier, X_train, y_train, cv=cv, scoring='accuracy', n_job

print("Decision Tree Classifier model accuracy: %.3f (%.3f)" % (np.mean(scores),np.std(

# Update dictionary from accuracy results
# results = {}
results['Decision Trees'] = np.mean(scores)
```

```
Decision Tree Classifier model accuracy: 0.877 (0.111)
```

In [12]:

```
# Evaluate Models - Random Forest

# Import libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

# Define model
classifier = RandomForestClassifier()

# Define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# Evaluate model
scores = 0
scores = cross_val_score(classifier, X_train, y_train, cv=cv, scoring='accuracy', n_job

print("Random Forerst model accuracy: %.3f (%.3f)" % (np.mean(scores),np.std(scores)),

# Update dictionary from accuracy results
# results = {}
results['Random Forest'] = np.mean(scores)
```

```
Random Forerst model accuracy: 0.924 (0.090)
```

In [13]:
```
# Evaluate Models - AdaBoost

# Import libraries
from sklearn.ensemble import AdaBoostClassifier

# Define model
classifier = AdaBoostClassifier()

# Define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# Evaluate model
scores = 0
scores = cross_val_score(classifier, X_train, y_train, cv=cv, scoring='accuracy', n_job

print("AdaBoost model accuracy: %.3f (%.3f)" % (np.mean(scores),np.std(scores)), "\n")

# Update dictionary from accuracy results
# results = {}
results['AdaBoost'] = np.mean(scores)
```

```
AdaBoost model accuracy: 0.727 (0.122)
```

In [14]:
```
# Evaluate Models - MLP Classifier

# Import libraries
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

# Define model
classifier = MLPClassifier()

# Define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# Evaluate model
scores = 0
scores = cross_val_score(classifier, X_train, y_train, cv=cv, scoring='accuracy', n_job

print("Multi-Layer Perceptron Model accuracy: %.3f (%.3f)" % (np.mean(scores),np.std(sc

# Update dictionary from accuracy results
# results = {}
results['MLP Classifier'] = np.mean(scores)
```

```
Multi-Layer Perceptron Model accuracy: 0.936 (0.083)
```

In [15]:
```
# Show results

results = sorted(results.items(), reverse=True, key=lambda item: item[1])
results
```

Out[15]:
```
[('SVC', 0.9411616161616163),
 ('MLP Classifier', 0.9356060606060608),
```

```
('kNN', 0.9356060606060607),
('Random Forest', 0.9244949494949496),
('LDA', 0.9133838383838384),
('Gaussian Naive Bayes', 0.8994949494949495),
('Decision Trees', 0.8770202020202021),
('SGD', 0.8436868686868686),
('AdaBoost', 0.7270202020202021)]
```

In [9]:

```
# Hyperparameter tuning - SVC
from skopt import BayesSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.svm import SVC

# define search space
hparams = dict()
hparams['C'] = (1e-6, 100.0, 'log-uniform')
hparams['gamma'] = (1e-6, 100.0, 'log-uniform')
hparams['degree'] = (1,5)
hparams['kernel'] = ['linear', 'poly', 'rbf', 'sigmoid']

# Define evaluation
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# Define Optimization Parameters
search = BayesSearchCV(estimator=SVC(), search_spaces=hparams, n_jobs=-1, cv=cv)

# Perform the search
search.fit(X_train, y_train)

# Store result
SVC_hyper_result = search.best_params_

# Report best result
print("Best hyperparameters:")
print(SVC_hyper_result, "\n")
print("SVC model accuracy after best hyperparameter definition: %.3f" % (search.best_sc
```

```
Best hyperparameters:
OrderedDict([('C', 1.0872838969479126), ('degree', 1), ('gamma', 100.0), ('kernel', 'lin
ear')])

SVC model accuracy after best hyperparameter definition: 0.944
```

In [17]:

```
# Hyperparameter tuning - kNN
from skopt import BayesSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.neighbors import KNeighborsClassifier as clf

# define search space
hparams = dict()
hparams['n_neighbors'] = (1, 50, 'uniform')
hparams['weights'] = ('uniform', 'distance')
hparams['p'] = (1, 2)

# Define evaluation
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# Define the search
search = BayesSearchCV(estimator=clf(), search_spaces=hparams, n_jobs=-1, cv=cv)
```

```
# Perform the search
search.fit(X_train, y_train)

# Store result
kNN_hyper_result = search.best_params_

# Report best result
print("Best hyperparameters:")
print(kNN_hyper_result, "\n")
print("kNN model accuracy after best hyperparameter definition: %.3f" % (search.best_sc
```

```
C:\Users\andre.rizzo\Anaconda3\lib\site-packages\skopt\optimizer\optimizer.py:449: UserW
arning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
C:\Users\andre.rizzo\Anaconda3\lib\site-packages\skopt\optimizer\optimizer.py:449: UserW
arning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
C:\Users\andre.rizzo\Anaconda3\lib\site-packages\skopt\optimizer\optimizer.py:449: UserW
arning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
C:\Users\andre.rizzo\Anaconda3\lib\site-packages\skopt\optimizer\optimizer.py:449: UserW
arning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
C:\Users\andre.rizzo\Anaconda3\lib\site-packages\skopt\optimizer\optimizer.py:449: UserW
arning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
Best hyperparameters:
OrderedDict([('n_neighbors', 18), ('p', 2), ('weights', 'uniform')])

kNN model accuracy after best hyperparameter definition: 0.944
```

In [22]:

```
# Hyperparameter tuning - Random Forest (using tune_sklearn)

from tune_sklearn import TuneSearchCV
from sklearn.ensemble import RandomForestClassifier as clf

# define search space
hparams = dict()
hparams['n_estimators'] = (1, 1000, 'uniform')
hparams['criterion'] = ['gini', 'entropy']
hparams['max_leaf_nodes'] = (2,100, "uniform")

# Define evaluation
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# Define the search
search = BayesSearchCV(estimator=clf(), search_spaces=hparams, n_jobs=-1, cv=cv)

# Perform the search
search.fit(X_train, y_train)

# Store result
RF_hyper_result = search.best_params_

# Report best result
print("Best hyperparameters:")
print(RF_hyper_result, "\n")
print("Random Forest model accuracy after best hyperparameter definition: %.3f" % (sear
```

```
Best hyperparameters:
OrderedDict([('criterion', 'gini'), ('max_leaf_nodes', 64), ('n_estimators', 407)])
```

```
Random Forest model accuracy after best hyperparameter definition: 0.927
```

In [ ]:

```
# Hyperparameter tuning - LDA (using tune_sklearn)

from tune_sklearn import TuneSearchCV
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA


# define search space
hparams = dict()
hparams['solver'] = ['lsqr', 'eigen']
hparams['shrinkage'] = (None,'auto')
#hparams['n_components'] = (1, 2)
hparams['tol'] = (1e-6, 1e10, "log-uniform")


# Define evaluation
#cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# Define the search
tune_search = TuneSearchCV(LDA(),
                           param_distributions = hparams,
                           use_gpu = False,
                           cv = 10,
                           n_jobs = 1,
                           search_optimization = 'bayesian',
                           refit = True,
                           verbose=0,
                           random_state=1
                          )

# Perform the search
tune_search.fit(X_train, y_train.values.ravel())

# Report the best result
print(tune_search.best_params_)

print("Multi-Layer Perceptron Model accuracy: %.3f (%.3f)" % (np.mean(scores),np.std(sc
```

In [15]:

```
# Train SVC model using computed hyperparameters

from sklearn.svm import SVC
import sklearn.metrics as clf_metrics


# Defining model with hyperparameters
model = SVC(C = SVC_hyper_result["C"],
            kernel = SVC_hyper_result["kernel"],
            degree = SVC_hyper_result["degree"],
            gamma = SVC_hyper_result["gamma"],
           )

# Fitting model to train set
model.fit(X_train, y_train)

# Predicting with testing set
y_pred = model.predict(X_test)
```

```
#clf_metrics.auc(y_test, y_pred)

# Show metrics
print('Confusion matrix')
print(clf_metrics.confusion_matrix(y_test, y_pred),"\n")
print('Accuracy score:', clf_metrics.accuracy_score(y_test, y_pred),"\n")
print('Class weights')
print(model.class_weight_)
```

```
Confusion matrix
[[12  0  0]
 [ 0 10  0]
 [ 0  0  8]]

Accuracy score: 1.0

Class weights
[1. 1. 1.]
```

In [ ]:

```
# Train kNN model using computed hyperparameters
```