Source Code : Cuckoo search with runge kutta   -Arnab Choudhury(21BHI10028)

```
import numpy as np
from scipy.integrate import solve_ivp
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import pandas as pd
from scipy.io import arff
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
from sklearn.preprocessing import LabelEncoder
```

```
df = pd.read_csv("data.csv")
```

```
df.head(5)
```

```
df.tail(5)
```

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

```
df.diagnosis.value_counts()
```

```
df.drop(['Unnamed: 32','id'],axis=1,inplace=True)
```

```
corr=df.corr()
corr.shape
```

```
plt.figure(figsize=(20,20))
sns.heatmap(corr,cbar=True,square=True,fmt='.1f',annot=True)
```

```
plt.figure(figsize=(15,10))
sns.boxplot(x="diagnosis",y="radius_mean",data=df)
plt.show()
```

```
plt.figure(figsize=(15,10))
sns.boxplot(df)
plt.show()
```

Source Code : Cuckoo search with runge kutta   -Arnab Choudhury(21BHI10028)

```
sns.FacetGrid(df,hue='diagnosis',height=10).map(sns.kdeplot,"radius_mea
n").add_legend()
plt.show()
```

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

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size =
0.3,random_state = 0)
```

```
from scipy.integrate import solve_ivp
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardize the features (you can use other preprocessing methods as
needed)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Cuckoo Search Algorithm for Feature Selection
def cuckoo_search(objective_function, num_dimensions, num_cuckoos,
max_generations):
    # Initialize cuckoos randomly
    cuckoos = np.random.rand(num_cuckoos, num_dimensions)
    best_solution = None
    best_fitness = float('inf')

    for generation in range(max_generations):
        # Evaluate fitness for each cuckoo
        fitness_values = [objective_function(c) for c in cuckoos]

        # Update the best solution
        min_fitness_idx = np.argmin(fitness_values)
        if fitness_values[min_fitness_idx] < best_fitness:
            best_fitness = fitness_values[min_fitness_idx]
            best_solution = cuckoos[min_fitness_idx]

        # Perform a random walk (you can modify this step for RK)
        step_size = np.random.rand(num_cuckoos, num_dimensions)
        cuckoos += step_size

        # Abandon a fraction of the worst solutions and replace them 
with new ones
        # You can replace this step with RK optimization if needed
        # For simplicity, we just generate new random solutions here
        replace_idx = np.argsort(fitness_values)[-int(0.2 * 
num_cuckoos):]
        cuckoos[replace_idx] = np.random.rand(len(replace_idx), 
num_dimensions)

    return best_solution, best_fitness

# Define the objective function for CS (e.g., classification accuracy)
def cs_objective_function(selected_features):
    # Train a classifier on the selected features
    classifier = SVC()
    classifier.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = classifier.predict(X_test)

    # Calculate classification accuracy
    accuracy = accuracy_score(y_test, y_pred)
    return -accuracy  # Negative accuracy because CS seeks to minimize

# Use Cuckoo Search to select a subset of features
num_dimensions = X_train.shape[1]
num_cuckoos = 20
max_generations = 100

best_solution, best_fitness = cuckoo_search(cs_objective_function, 
num_dimensions, num_cuckoos, max_generations)
selected_features = np.where(best_solution > 0.5)[0]

# Runge-Kutta Optimization for Feature Selection (you will need to 
customize this part)
# Note: RK is not typically used for feature selection, so you may need 
to adapt it accordingly

def rk_objective_function(selected_features):
    # Train a classifier on the selected features
    classifier = SVC()
    classifier.fit(X_train[:, selected_features], y_train)

    # Make predictions on the test set
    y_pred = classifier.predict(X_test[:, selected_features])
    # Calculate classification accuracy
    accuracy = accuracy_score(y_test, y_pred)
    return -accuracy  # Negative accuracy because RK seeks to minimize

# Define the differential equation for RK optimization
def differential_equation(t, features):
    # You can implement your own differential equation
    # Here, we use a simple linear equation as an example
    return -features

# Set up RK optimization
t_span = (0, 1)  # Time span
initial_features = np.ones(num_dimensions)  # Initial feature values

# Solve the differential equation using RK
solution = solve_ivp(differential_equation, t_span, initial_features,
method='RK45')

# Get the optimized features from the solution
optimized_features = solution.y[:, -1]

# Use the selected and optimized features for classification
final_selected_features = selected_features
final_optimized_features = np.where(optimized_features > 0.5)[0]
final_features = np.union1d(final_selected_features,
final_optimized_features)

# Train a classifier on the final feature set
final_classifier = SVC()
final_classifier.fit(X_train[:, final_features], y_train)

# Make predictions on the test set
y_final_pred = final_classifier.predict(X_test[:, final_features])

# Evaluate the final classifier's performance
final_accuracy = accuracy_score(y_test, y_final_pred)
print("Final Classification Accuracy:", final_accuracy)
```

```
plt.figure(figsize=(15,10))
sns.histplot(df)
plt.show()
```

```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(y_test, y_final_pred)
```

Source Code : Cuckoo search with runge kutta   -Arnab Choudhury(21BHI10028)

```
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['malignant', 'benign'],
            yticklabels=['malignant', 'benign'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()
```

```
from sklearn.metrics import precision_recall_curve
```

```
from sklearn.linear_model import LogisticRegression
```

```
LogReg=LogisticRegression()
```

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

```
LogReg_score = LogReg.predict_proba(X_test)[:, 1]
```

```
precision, recall, thresholds = precision_recall_curve(y_test,
LogReg_score, pos_label=1)
```

```
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')
```