iris

April 30, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.linear_model import LogisticRegression as LR
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

[2]:
```
iris = datasets.load_iris()
data = iris.data
labels = iris.target
num_samples,num_features = data.shape[0],data.shape[1]
print(f'num_samples,num_features = {(num_samples,num_features)}')
indices = np.arange(num_samples)
X_train, X_test, y_train, y_test,train_indices,test_indices =␣
 ↪train_test_split(data, labels,indices, test_size=0.6, random_state=42)
train_indices = np.arange(1,num_samples,4)
test_indices = np.arange(0,num_samples,4)
print(f'train_indices ={train_indices.shape},test_indices ={test_indices.
 ↪shape}')
```

```
num_samples,num_features = (150, 4)
train_indices =(38,),test_indices =(38,)
```

[3]:
```
data
```

[3]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
```

```
[4.8, 3. , 1.4, 0.1],
[4.3, 3. , 1.1, 0.1],
[5.8, 4. , 1.2, 0.2],
[5.7, 4.4, 1.5, 0.4],
[5.4, 3.9, 1.3, 0.4],
[5.1, 3.5, 1.4, 0.3],
[5.7, 3.8, 1.7, 0.3],
[5.1, 3.8, 1.5, 0.3],
[5.4, 3.4, 1.7, 0.2],
[5.1, 3.7, 1.5, 0.4],
[4.6, 3.6, 1. , 0.2],
[5.1, 3.3, 1.7, 0.5],
[4.8, 3.4, 1.9, 0.2],
[5. , 3. , 1.6, 0.2],
[5. , 3.4, 1.6, 0.4],
[5.2, 3.5, 1.5, 0.2],
[5.2, 3.4, 1.4, 0.2],
[4.7, 3.2, 1.6, 0.2],
[4.8, 3.1, 1.6, 0.2],
[5.4, 3.4, 1.5, 0.4],
[5.2, 4.1, 1.5, 0.1],
[5.5, 4.2, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.2],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.6, 1.4, 0.1],
[4.4, 3. , 1.3, 0.2],
[5.1, 3.4, 1.5, 0.2],
[5. , 3.5, 1.3, 0.3],
[4.5, 2.3, 1.3, 0.3],
[4.4, 3.2, 1.3, 0.2],
[5. , 3.5, 1.6, 0.6],
[5.1, 3.8, 1.9, 0.4],
[4.8, 3. , 1.4, 0.3],
[5.1, 3.8, 1.6, 0.2],
[4.6, 3.2, 1.4, 0.2],
[5.3, 3.7, 1.5, 0.2],
[5. , 3.3, 1.4, 0.2],
[7. , 3.2, 4.7, 1.4],
[6.4, 3.2, 4.5, 1.5],
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
```

```
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
```

```
       [4.9, 2.5, 4.5, 1.7],
       [7.3, 2.9, 6.3, 1.8],
       [6.7, 2.5, 5.8, 1.8],
       [7.2, 3.6, 6.1, 2.5],
       [6.5, 3.2, 5.1, 2. ],
       [6.4, 2.7, 5.3, 1.9],
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

[4]:
```
labels
```

[4]: array([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, 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, 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])

[5]:
```
train_indices
```

[5]:
```
array([  1,   5,   9,  13,  17,  21,  25,  29,  33,  37,  41,  45,  49,
        53,  57,  61,  65,  69,  73,  77,  81,  85,  89,  93,  97, 101,
       105, 109, 113, 117, 121, 125, 129, 133, 137, 141, 145, 149])
```

[6]:
```
test_indices
```

[6]:
```
array([  0,   4,   8,  12,  16,  20,  24,  28,  32,  36,  40,  44,  48,
        52,  56,  60,  64,  68,  72,  76,  80,  84,  88,  92,  96, 100,
       104, 108, 112, 116, 120, 124, 128, 132, 136, 140, 144, 148])
```

[7]:
```
from numpy.random import randint
from numpy.random import rand


def select_features(elem,features):
    selected_elem = np.where(elem==1)[0]
    selected_features = features[:,selected_elem]
    return selected_features

def classification_accuracy(labels,preds):
    correct = np.where(labels == preds)[0]
    accuracy = correct.shape[0]/labels.shape[0]
    return accuracy

def objective(pop,data,labels,train_ind,test_ind):
    accuracies = np.zeros(pop.shape[0])
    idx= 0
    for elem in pop:
        selected_features = select_features(elem,data)
        train_data = selected_features[train_ind,:]
        test_data = selected_features[test_ind,:]
        if train_data.shape[0]==0 or train_data.shape[1]==0 or test_data.
 ↪shape[0]==0 or test_data.shape[1]==0:
            idx=idx+1
            continue
        train_labels = labels[train_indices]
        test_labels = labels[test_indices]
```

```python
        LR_classifier = LR(random_state=0)
        LR_classifier.fit(X=train_data, y=train_labels)
        predictions = LR_classifier.predict(test_data)
        accuracies[idx] = classification_accuracy(test_labels, predictions)
        idx = idx + 1
    return accuracies

def parent_selection(pop,n_pop,scores,k=3):
    selected = []
    for _ in range(n_pop):
        idx = randint(len(pop))
        for ix in randint(0, len(pop),k-1):
            # check if better (e.g. perform a tournament)
            if scores[ix] < scores[idx]:
                idx = ix
        selected.append(pop[idx])
    return selected

def crossover(p1,p2,r_cross):
    c1 = p1.copy()
    c2 = p2.copy()
    if rand() < r_cross:
        pt = randint(1, len(p1)-2)
        c1 = list(p1[:pt])+list(p2[pt:])
        c2 = list(p2[:pt])+list(p1[pt:])
    return [np.array(c1), np.array(c2)]

def mutation(bitstring, r_mut):
    for i in range(len(bitstring)):
        # check for a mutation
        if rand() < r_mut:
            # flip the bit
            bitstring[i] = 1 - bitstring[i]
    return bitstring

def get_children(selected_parents,n_pop,r_cross,r_mut):
    children = []
    for i in range(0, n_pop, 2):
        p1, p2 = selected_parents[i], selected_parents[i+1]
        for c in crossover(p1, p2, r_cross):
            mutation(c, r_mut)
            children.append(c)
    return np.array(children)

def genetic_algorithm(epochs,data,labels,train_indices,test_indices):
    pop_size = 10
    k = 4
```

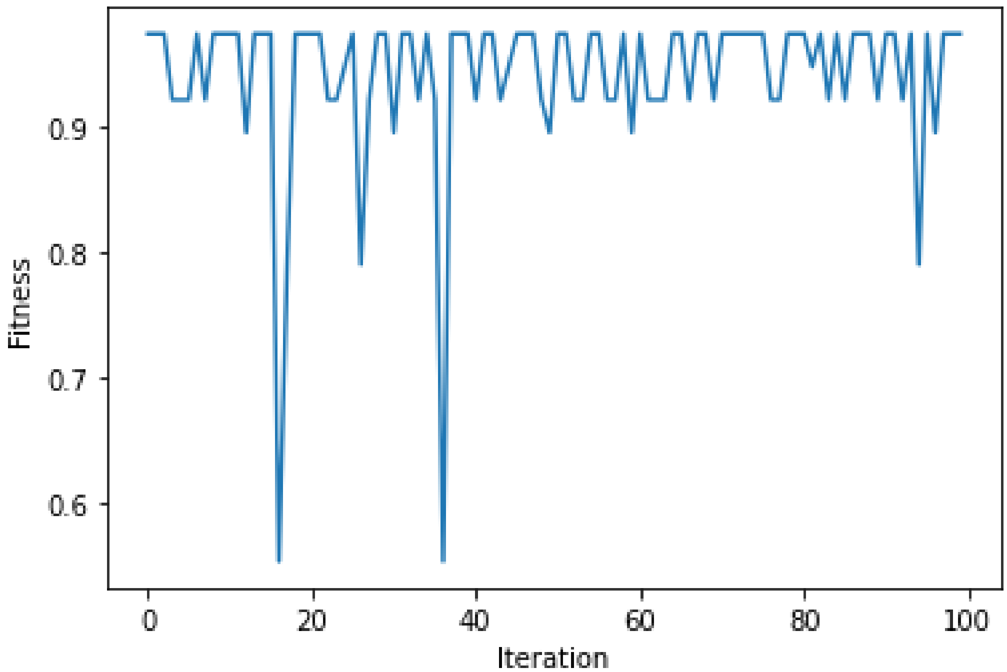
```
    r_cross = 0.9
    r_mut = 1/pop_size
    pop_shape = (pop_size, num_features)
    #initial population
    new_population = np.random.randint(low=0, high=2, size=pop_shape)
    print(f"new_population: {new_population} ")

    best_outputs = []
    num_generations = epochs
    for gen in range(num_generations):
        #measure fitness of each member in population
        scores = objective(new_population, data, labels, train_indices,␣
↪test_indices)

        #print current best in population
        best_outputs.append(np.max(scores))
        print(f"Gen: {gen} => Best result : {best_outputs[-1]}")

        #Select parent in current population to generate children for next␣
↪generation
        selected = parent_selection(new_population,pop_size,scores)

        #Get children of parents
        children = get_children(selected,pop_size,r_cross,r_mut)

        #replace old population
        new_population = children

#     best_outputs.append(np.max(scores))
#    print(f"Gen: {gen} => Best result : {best_outputs[-1]}")

    # Getting the best solution after iterating finishing all generations.
    # At first, the fitness is calculated for each solution in the final␣
↪generation.
    scores = objective(new_population, data, labels, train_indices,␣
↪test_indices)
    # Then return the index of that solution corresponding to the best fitness.
    best_match_idx = np.where(scores == np.max(scores))[0]
    best_match_idx = best_match_idx[0]
    print(f'np.max(scores) ={np.max(scores)}')
    best_solution = new_population[best_match_idx, :]
    best_solution_indices = np.where(best_solution == 1)[0]
    best_solution_num_elements = best_solution_indices.shape[0]
    best_solution_fitness = scores[best_match_idx]

    print("best_match_idx : ", best_match_idx)
    print("best_solution : ", best_solution)
```

```python
    print("Selected indices : ", best_solution_indices)
    print("Number of selected elements : ", best_solution_num_elements)
    print("Best solution fitness : ", best_solution_fitness)

    plt.plot(best_outputs)
    plt.xlabel("Iteration")
    plt.ylabel("Fitness")
    plt.show()

genetic_algorithm(100,data,labels,train_indices,test_indices)
```

```
new_population: [[0 1 0 0]
 [1 0 1 0]
 [1 1 1 0]
 [1 0 0 1]
 [1 1 0 0]
 [1 1 0 0]
 [1 1 1 1]
 [1 1 1 0]
 [1 1 1 1]
 [0 1 0 1]]
Gen: 0 => Best result : 0.9736842105263158
Gen: 1 => Best result : 0.9736842105263158
Gen: 2 => Best result : 0.9736842105263158
Gen: 3 => Best result : 0.9210526315789473
Gen: 4 => Best result : 0.9210526315789473
Gen: 5 => Best result : 0.9210526315789473
Gen: 6 => Best result : 0.9736842105263158
Gen: 7 => Best result : 0.9210526315789473
Gen: 8 => Best result : 0.9736842105263158
Gen: 9 => Best result : 0.9736842105263158
Gen: 10 => Best result : 0.9736842105263158
Gen: 11 => Best result : 0.9736842105263158
Gen: 12 => Best result : 0.8947368421052632
Gen: 13 => Best result : 0.9736842105263158
Gen: 14 => Best result : 0.9736842105263158
Gen: 15 => Best result : 0.9736842105263158
Gen: 16 => Best result : 0.5526315789473685
Gen: 17 => Best result : 0.7894736842105263
Gen: 18 => Best result : 0.9736842105263158
Gen: 19 => Best result : 0.9736842105263158
Gen: 20 => Best result : 0.9736842105263158
Gen: 21 => Best result : 0.9736842105263158
Gen: 22 => Best result : 0.9210526315789473
Gen: 23 => Best result : 0.9210526315789473
Gen: 24 => Best result : 0.9473684210526315
Gen: 25 => Best result : 0.9736842105263158
Gen: 26 => Best result : 0.7894736842105263
```

```
Gen: 27 => Best result : 0.9210526315789473
Gen: 28 => Best result : 0.9736842105263158
Gen: 29 => Best result : 0.9736842105263158
Gen: 30 => Best result : 0.8947368421052632
Gen: 31 => Best result : 0.9736842105263158
Gen: 32 => Best result : 0.9736842105263158
Gen: 33 => Best result : 0.9210526315789473
Gen: 34 => Best result : 0.9736842105263158
Gen: 35 => Best result : 0.9210526315789473
Gen: 36 => Best result : 0.5526315789473685
Gen: 37 => Best result : 0.9736842105263158
Gen: 38 => Best result : 0.9736842105263158
Gen: 39 => Best result : 0.9736842105263158
Gen: 40 => Best result : 0.9210526315789473
Gen: 41 => Best result : 0.9736842105263158
Gen: 42 => Best result : 0.9736842105263158
Gen: 43 => Best result : 0.9210526315789473
Gen: 44 => Best result : 0.9473684210526315
Gen: 45 => Best result : 0.9736842105263158
Gen: 46 => Best result : 0.9736842105263158
Gen: 47 => Best result : 0.9736842105263158
Gen: 48 => Best result : 0.9210526315789473
Gen: 49 => Best result : 0.8947368421052632
Gen: 50 => Best result : 0.9736842105263158
Gen: 51 => Best result : 0.9736842105263158
Gen: 52 => Best result : 0.9210526315789473
Gen: 53 => Best result : 0.9210526315789473
Gen: 54 => Best result : 0.9736842105263158
Gen: 55 => Best result : 0.9736842105263158
Gen: 56 => Best result : 0.9210526315789473
Gen: 57 => Best result : 0.9210526315789473
Gen: 58 => Best result : 0.9736842105263158
Gen: 59 => Best result : 0.8947368421052632
Gen: 60 => Best result : 0.9736842105263158
Gen: 61 => Best result : 0.9210526315789473
Gen: 62 => Best result : 0.9210526315789473
Gen: 63 => Best result : 0.9210526315789473
Gen: 64 => Best result : 0.9736842105263158
Gen: 65 => Best result : 0.9736842105263158
Gen: 66 => Best result : 0.9210526315789473
Gen: 67 => Best result : 0.9736842105263158
Gen: 68 => Best result : 0.9736842105263158
Gen: 69 => Best result : 0.9210526315789473
Gen: 70 => Best result : 0.9736842105263158
Gen: 71 => Best result : 0.9736842105263158
Gen: 72 => Best result : 0.9736842105263158
Gen: 73 => Best result : 0.9736842105263158
Gen: 74 => Best result : 0.9736842105263158
```

```
Gen: 75 => Best result : 0.9736842105263158
Gen: 76 => Best result : 0.9210526315789473
Gen: 77 => Best result : 0.9210526315789473
Gen: 78 => Best result : 0.9736842105263158
Gen: 79 => Best result : 0.9736842105263158
Gen: 80 => Best result : 0.9736842105263158
Gen: 81 => Best result : 0.9473684210526315
Gen: 82 => Best result : 0.9736842105263158
Gen: 83 => Best result : 0.9210526315789473
Gen: 84 => Best result : 0.9736842105263158
Gen: 85 => Best result : 0.9210526315789473
Gen: 86 => Best result : 0.9736842105263158
Gen: 87 => Best result : 0.9736842105263158
Gen: 88 => Best result : 0.9736842105263158
Gen: 89 => Best result : 0.9210526315789473
Gen: 90 => Best result : 0.9736842105263158
Gen: 91 => Best result : 0.9736842105263158
Gen: 92 => Best result : 0.9210526315789473
Gen: 93 => Best result : 0.9736842105263158
Gen: 94 => Best result : 0.7894736842105263
Gen: 95 => Best result : 0.9736842105263158
Gen: 96 => Best result : 0.8947368421052632
Gen: 97 => Best result : 0.9736842105263158
Gen: 98 => Best result : 0.9736842105263158
Gen: 99 => Best result : 0.9736842105263158
np.max(scores) =0.9210526315789473
best_match_idx :  9
best_solution :  [0 0 1 0]
Selected indices :  [2]
Number of selected elements :  1
Best solution fitness :  0.9210526315789473
```


Fitness
0.9
0.8
0.7
0.6
0
20
40
60
80
100
Iteration