LAMPIRAN

L.1 *Simple Source Code* kNN+PSO

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder
from docx import Document

class Particle:
    def __init__(self, dimensions, bounds):
        self.position = np.random.uniform(bounds[0],
bounds[1], dimensions)
        self.velocity = np.random.uniform(-0.1, 0.1,
dimensions)
        self.best_position = self.position.copy()
        self.best_score = float('-inf')

def knn_accuracy(X_train, X_test, y_train, y_test, k):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    return accuracy_score(y_test, y_pred)

def objective_function(particle, X_train, X_test, y_train,
y_test, k):
    accuracy = knn_accuracy(X_train, X_test, y_train, y_test,
k)
    if accuracy > particle.best_score:
        particle.best_score = accuracy
        particle.best_position = particle.position.copy()
    return accuracy

def pso_knn(X_train, X_test, y_train, y_test, k,
n_particles=30, max_iter=100, w=0.5, c1=1, c2=2):
    dimensions = X_train.shape[1]
    bounds = (0, 1)
    particles = [Particle(dimensions, bounds) for _ in
range(n_particles)]
    global_best_position = np.zeros(dimensions)
    global_best_score = float('-inf')

    import sys
    np.set_printoptions(threshold=sys.maxsize)
```

```
    pd.set_option('display.max_rows', None)

    output = []  # Untuk menyimpan output

    for iteration in range(max_iter):
        iteration_output = []  # Untuk menyimpan output pada
setiap iterasi
        iteration_output.append(f"Iteration:
{iteration+1}\n")
        for i, particle in enumerate(particles):
            iteration_output.append(f"Particle {i+1} Position:
{particle.position}\n")
            iteration_output.append(f"Particle {i+1} Velocity:
{particle.velocity}\n")
            fitness = objective_function(particle, X_train,
X_test, y_train, y_test, k)
            iteration_output.append(f"Particle {i+1} Best
Position: {particle.best_position}\n")

            if fitness > global_best_score:
                global_best_score = fitness
                global_best_position =
particle.position.copy()

            w *= 0.99  # decay inertia weight
            r1, r2 = np.random.rand(dimensions),
np.random.rand(dimensions)
            particle.velocity = (w * particle.velocity +
                c1 * r1 * (particle.best_position -
particle.position) +
                c2 * r2 * (global_best_position -
particle.position))
            particle.position += particle.velocity

        iteration_output.append(f"Global Best Position:
{global_best_position}\n")  # Tambahkan Global best position
        output.extend(iteration_output)

    return global_best_position, global_best_score, output

# Download dataset dari Google Drive
!gdown --id 1xhdLfZzv2RkXJE2ZfaDDCmtJQJuJlUpT -O dataset.csv

# Membaca dataset
df = pd.read_csv('dataset.csv', sep=';')
```

```
# Mendapatkan target_column, X, dan y
target_column = 'Diabetic'  # Kolom target ada pada kolom ke-
18
X = df.drop(target_column, axis=1)
y = df[target_column]

# Melakukan one-hot encoding untuk fitur-fitur kategorikal
categorical_features =
X.select_dtypes(include=['object']).columns.tolist()
if categorical_features:
    encoder = OneHotEncoder(drop='first',
sparse_output=False)  # Perubahan disini
    X_encoded =
pd.DataFrame(encoder.fit_transform(X[categorical_features]))
    X_encoded.columns =
encoder.get_feature_names_out(categorical_features)
    X = pd.concat([X.drop(columns=categorical_features),
X_encoded], axis=1)

# Membagi dataset menjadi data latih dan data uji dengan
metode random-stratified
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, random_state=42, stratify=y)

# Menggunakan PSO untuk mengoptimalkan KNN dengan k=5
best_position, best_score, output = pso_knn(X_train, X_test,
y_train, y_test, k=5)

print("Best position:", best_position)
print("Accuracy score:", best_score)

# Menyimpan output ke dalam file .docx
doc = Document()
doc.add_heading('PSO Optimization for KNN Results', level=1)
doc.add_paragraph(f"Best position: {best_position}")
doc.add_paragraph(f"Accuracy score: {best_score}")

for line in output:
    doc.add_paragraph(line)

doc.save('pso_knn_results.docx')
```

L.2 *Streaming Output*

Berikut adalah potongan *output* dari optimasi PSO pada algoritma kNN yang di-*running* menggunakan *code* Python. Di sini penulis mengambil 3 posisi partikel terakhir dari iterasi ke-2.

. . .

Particle 28 Position: [0.55433236 0.31549455 0.28027973 0.10285564 0.49661065 0.78685693
0.47597327 1.1341218 0.54557732 0.56478085 0.13851841 0.28010264
0.36912695 0.81169575 0.54209872 0.90405304 0.19827801 0.32831278
0.77875702 0.30265006 -0.27201313 0.56874643 0.80805863 0.24823435
0.67934405 0.37582711]

Particle 28 Velocity: [-0.05879723 -0.27295467 -0.23321226 -0.54119708 -0.3527829 -0.1992613
0.25814593 0.88020159 0.33504606 -0.19287333 0.07218852 -0.06939519
-0.53276313 -0.16749581 0.03888888 0.60114377 -0.04017492 -0.0305457
0.04049313 0.25294009 -1.06028523 0.26164554 0.04369994 0.23107289
-0.22445674 -0.1395144]

Particle 28 Best Position: [0.61312958 0.58844922 0.51349199 0.64405272 0.84939355 0.98611824
0.21782734 0.25392022 0.21053126 0.75765418 0.06632988 0.34949783
0.90189009 0.97919155 0.50320983 0.30290926 0.23845293 0.35885849
0.73826389 0.04970997 0.7882721 0.30710088 0.76435869 0.01716146
0.90380079 0.5153415]

Particle 29 Position: [0.78439702 -0.1004691 0.35652791 0.2687645 0.61241943 0.76941481
0.36788204 0.79278137 0.51644923 0.75764537 0.14611282 0.41293304
-0.25514744 0.79592431 0.45836749 1.15696939 0.18852125 0.16369167
1.06534638 0.46850454 0.33780759 0.44032038 0.55746761 0.42952829
-0.30050623 0.36536842]

Particle 29 Velocity: [0.73498387 -1.00555279 -0.03581197 -0.14497562 0.26484339 0.75276923
0.03420533 0.0441196 -0.04171369 0.6785455 0.03919623 -0.19446217
-0.90584527 0.78063094 0.43357105 0.93757355 -0.06572782 -0.27259723
0.95062543 0.07158903 -0.30110945 0.01139177 0.05680196 0.09458172
-1.21582117 -0.32657674]

Particle 29 Best Position: [0.04941316 0.9050837 0.39233988 0.41374012 0.34757604 0.01664558
 0.33367671 0.74866178 0.55816291 0.07909987 0.10691659 0.60739521
 0.65069783 0.01529337 0.02479645 0.21939585 0.25424908 0.4362889
 0.11472095 0.39691551 0.63891704 0.42892861 0.50066564 0.33494657
 0.91531494 0.69194516]

Particle 30 Position: [0.75943102 0.28106025 0.19496262 0.40573559 0.47380579 0.71233372
  0.34323414 1.41172624 0.50872814 0.37804607 0.2646594   0.12805241
 -0.55130721 1.0995461   0.64599723 0.54399274 0.18338545 0.28583529
  0.73404424 0.27400423 -0.34881959 0.54396482 0.14024493 0.51262637
  0.22310896 0.1812596]

Particle 30 Velocity: [-0.10416342 0.21740164 -0.43196744 -0.007283   -0.3602982 0.21470321
 -0.5990621   1.37982458 0.01144463 -0.20318617 0.06506698 -0.06273568
 -1.31177632 0.97953987 0.45963048 0.26136093 -0.30281606 0.01601596
  0.02720621 -0.53745624 -0.86839284 0.29144617 0.13505487 -0.20640404
  0.00391104 -0.45727268]

Particle 30 Best Position: [0.86359444 0.06365862 0.62693006 0.41301859 0.83410399 0.4976305
 0.94229624 0.03190166 0.49728351 0.58123224 0.19959243 0.19078809
 0.76046911 0.12000623 0.18636675 0.28263181 0.48620151 0.26981934
 0.70683803 0.81146047 0.51957325 0.25251865 0.00519006 0.71903041
 0.21919793 0.63853228]

Global Best Position: [0.53147083 0.20529575 0.38980887 0.16065598 0.65065202 0.71344236
 0.57065707 0.80906625 0.51459175 0.48808064 0.27166801 0.06053902
 0.08271363 0.8863753 0.77093791 0.87375208 0.16812673 0.29513939
 0.7969456 0.46561045 0.01136601 0.42995411 0.77688507 0.47800729
 0.24363085 0.18656826]

Iteration: 3

. . .