1 Import Libraries

1.1 Data Processing Libraries

In [1]:

```
# data processing
import numpy as np
import pandas as pd
from scipy.stats.mstats import winsorize
from scipy import stats
```

1.2 Sklearn

In [2]:

```
# sklearn
from sklearn.model_selection import train_test_split, KFold
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn import metrics
```

1.3 GLVQ & PSO manual libs

In [3]:

```
# model classes
from models.GeneralizedLearningVectorQuantization import GeneralizedLearningVectorQuantizat
from models.ParticleSwarmOptimization import ParticleSwarmOptimization as PSO
from models.Utilization import Utilization
```

1.4 Utils

In [4]:

```
import random
import pickle
# random_state = 7
# random_state = 13
# random_state=20
# random_state=24
# random_state=29
# random_state=30
# random_state=31
# random_state=32
# random_state=33
# random_state=34
# random_state=51
# random_state=60


random_state=55
random.seed(random_state)
```

1.5 Visualization

In [5]:

```
#import visualizing libraries
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

2 Load Data Train and Testing

In [6]:

```
wdbc_preprocessed = pickle.load(open('results/dataset_prep.pkl', 'rb'))
```

In [7]:

```
X_test = wdbc_preprocessed['X_test']
y_test = wdbc_preprocessed['y_test']
X_train = wdbc_preprocessed['X_train']
y_train = wdbc_preprocessed['y_train']

testing_set = pickle.load(open('results/dataset.pkl', 'rb'))['testing']['data']
features_name = pickle.load(open('results/dataset.pkl', 'rb'))['features_name']
```

3 Load Optimal Parameter GLVQ dan PSO

3.1 Load Optimal Parameter GLVQ

In [8]:

```
# optimal GLVQ parameter
optimal_parameters_glvq = pickle.load(open('informations/optimal_parameters_glvq.pkl','rb'))
optimal_codebook = optimal_parameters_glvq['optimal_codebook']
optimal_alpha = optimal_parameters_glvq['optimal_alpha']
optimal_max_epoch = optimal_parameters_glvq['optimal_max_epoch']
optimal_min_error = optimal_parameters_glvq['optimal_min_error']
```

In [9]:

```
optimal_parameters_glvq
```

Out[9]:

```
{'optimal_codebook': 5,
 'optimal_alpha': 0.1,
 'optimal_max_epoch': 100,
 'optimal_min_error': 1e-06}
```

3.2 Load Optimal Parameter PSO

In [10]:

```
optimal_parameters_pso_shi = pickle.load(open('informations/optimal_parameters_pso_shi.pkl'

optimal_phi1_shi = optimal_parameters_pso_shi['optimal_phi1']
optimal_phi2_shi = optimal_parameters_pso_shi['optimal_phi2']
optimal_inertia_shi = optimal_parameters_pso_shi['optimal_inertia']
optimal_n_particles_shi = optimal_parameters_pso_shi['optimal_n_particles']
optimal_max_iter_shi = optimal_parameters_pso_shi['optimal_max_iter']
```

In [11]:

```
optimal_parameters_pso_shi
```

Out[11]:

```
{'optimal_phi1': 2.4,
 'optimal_phi2': 2.1,
 'optimal_inertia': 0.6,
 'optimal_n_particles': 30,
 'optimal_max_iter': 100}
```

4 Train and Evaluate Model

4.1 Training Models

4.1.1 Training PSO-GLVQ

In [13]:

```
pso_shi = PSO(
    n_codebooks=optimal_codebook,
    max_iter=optimal_max_iter_shi,
    phi1=optimal_phi1_shi,
    phi2=optimal_phi2_shi,
    inertia=optimal_inertia_shi,
    velocity_update='shi',
    n_particles=optimal_n_particles_shi
)
weight_pso_shi_train = pso_shi.fit(X_train, y_train)
glvq_pso_shi = GLVQ(
    n_codebooks=optimal_codebook,
    initial_weight=weight_pso_shi_train,
    alpha=optimal_alpha,
    max_epoch = optimal_max_epoch,
    min_error=optimal_min_error
)
glvq_pso_shi.fit(X_train, y_train)
```

Out[13]:

```
<models.GeneralizedLearningVectorQuantization.GeneralizedLearningVectorQuant
ization at 0x23fdb6c49d0>
```

In [14]:

```
y_pred_test_glvq_pso_shi = glvq_pso_shi.predict(X_test)
```

4.1.2 Training GLVQ

In [15]:

```
glvq = GLVQ(
    n_codebooks=optimal_codebook,
    alpha=optimal_alpha,
    max_epoch = optimal_max_epoch,
    min_error=optimal_min_error
)
glvq.fit(X_train, y_train)
```

Out[15]:

```
<models.GeneralizedLearningVectorQuantization.GeneralizedLearningVectorQuant
ization at 0x23fdb6ec1c0>
```

In [16]:

```
y_pred_test_glvq = glvq.predict(X_test)
```

4.2 Evaluate Models

4.2.1 Confusion Matrix

4.2.1.1 Confusion Matrix PSOGLVQ

In [17]:

```
print("Akurasi data testing GLVQ-PSO : ",metrics.accuracy_score(y_test, y_pred_test_glvq_ps
print(metrics.confusion_matrix(y_test, y_pred_test_glvq_pso_shi))
print(metrics.classification_report(y_test, y_pred_test_glvq_pso_shi))
```

```
Akurasi data testing GLVQ-PSO :  0.9385964912280702
[[72  0]
 [ 7 35]]
              precision    recall  f1-score   support

         0.0       0.91      1.00      0.95        72
         1.0       1.00      0.83      0.91        42

    accuracy                           0.94       114
   macro avg       0.96      0.92      0.93       114
weighted avg       0.94      0.94      0.94       114
```

4.2.1.2 Confusion Matrix GLVQ

In [18]:

```
print("Akurasi data testing GLVQ : ", metrics.accuracy_score(y_test, y_pred_test_glvq))
print(metrics.confusion_matrix(y_test, y_pred_test_glvq))
print(metrics.classification_report(y_test, y_pred_test_glvq))
```

```
Akurasi data testing GLVQ :  0.9298245614035088
[[70  2]
 [ 6 36]]
              precision    recall  f1-score   support

         0.0       0.92      0.97      0.95        72
         1.0       0.95      0.86      0.90        42

    accuracy                           0.93       114
   macro avg       0.93      0.91      0.92       114
weighted avg       0.93      0.93      0.93       114
```

5 Simpan Hasil Pelatihan dan Evaluasi dari model PSOGLVQ dan GLVQ

5.1 Simpan model PSOGLVQ dan GLVQ hasil pelatihan

In [19]:

```
glvq_pso_shi.save('results/psoglvq_trained.pkl')
glvq.save('results/glvq_trained.pkl')
```

5.2 Simpan hasil evaluasi PSOGLVQ dan GLVQ

In [20]:

```
confusion_matrix_pso_glvq = metrics.confusion_matrix(
    y_pred_test_glvq_pso_shi, y_test).ravel()
confusion_matrix_glvq = metrics.confusion_matrix(
    y_pred_test_glvq, y_test).ravel()

# buat tabel perbandingan hasil klasifikasi PSO-GLVQ dan GLVQ
label_comparison = pd.DataFrame({
    'label': testing_set[:,-1],
    'label_PSO_GLVQ': y_pred_test_glvq_pso_shi,
    'label_GLVQ': y_pred_test_glvq,
})
for i, col in enumerate(features_name):
    label_comparison[col] = testing_set[:, i]

evaluation_results = {
    'metrics': {
        'PSOGLVQ': {
            'akurasi': round((metrics.accuracy_score(y_pred_test_glvq_pso_shi, y_test)*100)
            'error_rate': round(((1-metrics.accuracy_score(y_pred_test_glvq_pso_shi, y_test
            'recall': round((metrics.recall_score(y_pred_test_glvq_pso_shi, y_test)*100), 4
            'precision': round((metrics.precision_score(y_pred_test_glvq_pso_shi, y_test)*1
            'f2score': round((metrics.fbeta_score(y_pred_test_glvq_pso_shi, y_test, beta=2.
            'TN': confusion_matrix_pso_glvq[0],
            'FP': confusion_matrix_pso_glvq[1],
            'FN': confusion_matrix_pso_glvq[2],
            'TP': confusion_matrix_pso_glvq[3],
        },
        'GLVQ': {
            'akurasi': round((metrics.accuracy_score(y_pred_test_glvq, y_test)*100), 4),
            'error_rate': round(((1-metrics.accuracy_score(y_pred_test_glvq, y_test))*100),
            'recall': round((metrics.recall_score(y_pred_test_glvq, y_test)*100), 4),
            'precision': round((metrics.precision_score(y_pred_test_glvq, y_test)*100), 4),
            'f2score': round((metrics.fbeta_score(y_pred_test_glvq, y_test, beta=2.0)*100),
            'TN': confusion_matrix_glvq[0],
            'FP': confusion_matrix_glvq[1],
            'FN': confusion_matrix_glvq[2],
            'TP': confusion_matrix_glvq[3],
        },
    },
    'labelcomparison': {
        'data': label_comparison.to_numpy(),
        'features': label_comparison.columns.values
    }
}
pickle.dump(evaluation_results, open('results/evaluation_results.pkl', 'wb'))
```

In [21]:

```
evaluation_results
```

Out[21]:

```
{'metrics': {'PSOGLVQ': {'akurasi': 93.8596,
   'error_rate': 6.1404,
   'recall': 100.0,
   'precision': 83.3333,
   'f2score': 96.1538,
   'TN': 72,
   'FP': 7,
   'FN': 0,
   'TP': 35},
  'GLVQ': {'akurasi': 92.9825,
   'error_rate': 7.0175,
   'recall': 94.7368,
   'precision': 85.7143,
   'f2score': 92.7835,
   'TN': 70,
   'FP': 6,
   'FN': 2,
   'TP': 36}},
 'labelcomparison': {'data': array([[0.     , 0.     , 0.     , ..., 0.0895
8, 0.3016 , 0.08523],
         [1.     , 1.     , 1.     , ..., 0.2105 , 0.3126 , 0.07849],
         [0.     , 0.     , 0.     , ..., 0.1251 , 0.3153 , 0.0896 ],
         ...,
         [0.     , 0.     , 0.     , ..., 0.09594, 0.2471 , 0.07463],
         [1.     , 0.     , 0.     , ..., 0.1252 , 0.3415 , 0.0974 ],
         [1.     , 1.     , 1.     , ..., 0.152  , 0.265  , 0.06387]]),
  'features': array(['label', 'label_PSO_GLVQ', 'label_GLVQ', 'radius_mean',
         'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean',
         'compactness_mean', 'concavity_mean', 'concave_points_mean',
         'symmetry_mean', 'fractal_dimension_mean', 'radius_se',
         'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
         'compactness_se', 'concavity_se', 'concave_points_se',
         'symmetry_se', 'fractal_dimension_se', 'radius_largest',
         'texture_largest', 'perimeter_largest', 'area_largest',
         'smoothness_largest', 'compactness_largest', 'concavity_largest',
         'concave_points_largest', 'symmetry_largest',
         'fractal_dimension_largest'], dtype=object)}}
```

In [ ]:

```

```