

zmqpe2hh5

December 4, 2025

```
[3]: # =====
# 04 - Genetic Algorithm Tuning for Logistic Regression (GA + LR)
# Author: Nasser Qanabta
# Description:
# 1) Load cleaned CKD dataset (CKD_cleaned_dataset_EN.csv)
# 2) Split into Train (70%) and Test (30%) using stratified sampling
# 3) Define LogisticRegression and GA hyperparameter search space
# 4) Use GASearchCV to find best hyperparameters on Train set
# 5) Evaluate best LR model on Train and Test sets
# 6) Save best hyperparameters, scores, GA history, and best model
# (for later use in File 06 - Hybrid Probability Fusion)
# =====

!pip install sklearn-genetic-opt -q

# 1) Imports
import pandas as pd
import numpy as np
import json
import joblib # for saving the best GA-tuned model

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    roc_auc_score,
    classification_report
)
from sklearn_genetic import GASearchCV
from sklearn_genetic.space import Integer, Categorical, Continuous
from google.colab import files

# -----
# 2) Upload and load cleaned CKD dataset
```

```

# (generated in 01_preprocessing.ipynb)
# -----
print(" Please upload: CKD_cleaned_dataset_EN.csv")
uploaded = files.upload() # select CKD_cleaned_dataset_EN.csv

df = pd.read_csv("CKD_cleaned_dataset_EN.csv")

print("Full cleaned dataset shape:", df.shape)

# Separate features and target
target_col = "Chronic_Kidney_Disease"
X = df.drop(columns=[target_col])
y = df[target_col]

print("Features shape:", X.shape, "Target shape:", y.shape)

# -----
# 3) Train/Test split (70% / 30%) with stratification
# (same settings as other GA files)
# -----
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.30,
    stratify=y,
    random_state=42
)

print("\nStratified split (70/30):")
print(" X_train:", X_train.shape, "y_train:", y_train.shape)
print(" X_test :", X_test.shape, "y_test :", y_test.shape)

# -----
# 4) Define base Logistic Regression model
# (solver and penalty will be controlled by GA search space)
# -----
base_lr = LogisticRegression(
    random_state=42,
    max_iter=2000
)

# -----
# 5) Define GA hyperparameter search space
# Note:
# - We restrict to valid (penalty, solver) combinations
# supported by current scikit-learn version.
# - Here we use only L2 regularization with lbfgs / saga.

```

```

# -----
param_grid = {
    "C": Continuous(1e-3, 100.0, distribution="log-uniform"),
    "penalty": Categorical(["l2"]), # <= remove "none"
    "solver": Categorical(["lbfgs", "saga"]), # both support l2
    "class_weight": Categorical([None, "balanced"]),
}

# -----
# 6) Configure GA Search (on Train set only)
# -----
ga_search = GASearchCV(
    estimator=base_lr,
    cv=3, # 3-fold CV on TRAIN data
    scoring="f1", # can be changed to "roc_auc"
    param_grid=param_grid,
    n_jobs=-1, # use all cores
    population_size=20, # can increase for deeper search
    generations=12, # can increase for deeper search
    verbose=True,
)

# -----
# 7) Run GA Search on TRAINING data only
# -----
ga_search.fit(X_train, y_train)

print("\n GA Search (Logistic Regression) completed.")

# -----
# 8) Extract best estimator and hyperparameters
# -----
best_lr = ga_search.best_estimator_
best_params = ga_search.best_params_
best_cv_score = ga_search.best_score_

print("\n Best Hyperparameters (GA + LR):")
print(best_params)
print("Best CV F1 score:", best_cv_score)

# -----
# 9) Evaluation function (Train / Test)
# -----
def evaluate_model(model, X, y, dataset_name=""):
    y_pred = model.predict(X)

```

```

    y_prob = model.predict_proba(X)[: , 1] if hasattr(model, "predict_proba")
    ↪ else None

    acc = accuracy_score(y, y_pred)
    prec = precision_score(y, y_pred)
    rec = recall_score(y, y_pred)
    f1 = f1_score(y, y_pred)
    try:
        auc = roc_auc_score(y, y_prob) if y_prob is not None else None
    except ValueError:
        auc = None

    print(f"\n Performance on {dataset_name}:")
    print("Accuracy :", acc)
    print("Precision:", prec)
    print("Recall    :", rec)
    print("F1-score :", f1)
    if auc is not None:
        print("ROC-AUC :", auc)
    print("\nClassification report (4 decimals):")
    print(classification_report(y, y_pred, digits=4))

    return {
        "dataset": dataset_name,
        "accuracy": acc,
        "precision": prec,
        "recall": rec,
        "f1": f1,
        "roc_auc": auc,
    }

# -----
# 10) Evaluate best LR model on Train and Test sets
# -----
train_scores = evaluate_model(best_lr, X_train, y_train, dataset_name="Train",
    ↪ (70%))
test_scores = evaluate_model(best_lr, X_test, y_test, dataset_name="Test",
    ↪ (30%))

# -----
# 11) Save best hyperparameters, scores, GA history, and model
# -----

# 11.1 Save best hyperparameters as JSON
with open("LR_best_hyperparams_GA.json", "w") as f:
    json.dump(best_params, f, indent=4)

```

```

# 11.2 Save best scores for train/test as CSV
scores_df = pd.DataFrame([train_scores, test_scores])
scores_df.to_csv("LR_GA_best_scores_70_30.csv", index=False)

# 11.3 Save full GA search history (all tried combinations)
ga_results_df = pd.DataFrame(ga_search.cv_results_)
ga_results_df.to_csv("LR_GA_search_results_70_30.csv", index=False)

# 11.4 Save the best GA-tuned LR model for later use (File 06 - Hybrid_
↳Probability Fusion)
best_model_filename = "LR_GA_best_model.joblib"
joblib.dump(best_lr, best_model_filename)

print("\n Files saved:")
print("    - LR_best_hyperparams_GA.json")
print("    - LR_GA_best_scores_70_30.csv")
print("    - LR_GA_search_results_70_30.csv")
print(f"    - {best_model_filename} (best GA-tuned Logistic Regression model)")

# -----
# 12) (Optional) Download files to your local machine
# -----
files.download("LR_best_hyperparams_GA.json")
files.download("LR_GA_best_scores_70_30.csv")
files.download("LR_GA_search_results_70_30.csv")
files.download("LR_GA_best_model.joblib")

```

Please upload: CKD_cleaned_dataset_EN.csv

<IPython.core.display.HTML object>

Saving CKD_cleaned_dataset_EN.csv to CKD_cleaned_dataset_EN (2).csv

Full cleaned dataset shape: (15051, 21)

Features shape: (15051, 20) Target shape: (15051,)

Stratified split (70/30):

X_train: (10535, 20) y_train: (10535,)

X_test : (4516, 20) y_test : (4516,)

gen	nevals	fitness	fitness_std	fitness_max	fitness_min
0	20	0.998578	0.000182153	0.998932	0.99835
1	40	0.998762	0.000114223	0.998932	0.998447
2	40	0.99883	8.92537e-05	0.998932	0.998641
3	40	0.998888	5.71583e-05	0.998932	0.998738
4	40	0.998936	4.82649e-05	0.999029	0.998835
5	40	0.998956	5.20254e-05	0.999029	0.998835
6	40	0.999	4.44694e-05	0.999029	0.998932
7	40	0.999019	2.91121e-05	0.999029	0.998932
8	40	0.999029	4.44089e-16	0.999029	0.999029

9	40	0.999014	4.62742e-05	0.999029	0.998835
10	40	0.999024	2.11494e-05	0.999029	0.998932
11	40	0.999029	4.44089e-16	0.999029	0.999029
12	40	0.999019	4.22742e-05	0.999029	0.998835

GA Search (Logistic Regression) completed.

Best Hyperparameters (GA + LR):

```
{'C': np.float64(7.6292026924782705), 'penalty': 'l2', 'solver': 'lbfgs',
'class_weight': 'balanced'}
```

Best CV F1 score: 0.9990286357963419

Performance on Train (70%):

Accuracy : 0.9995253915519696
Precision: 0.9996113486202876
Recall : 0.9994171361958423
F1-score : 0.9995142329738658
ROC-AUC : 0.9999993869905549

Classification report (4 decimals):

	precision	recall	f1-score	support
0	0.9994	0.9996	0.9995	5388
1	0.9996	0.9994	0.9995	5147
accuracy			0.9995	10535
macro avg	0.9995	0.9995	0.9995	10535
weighted avg	0.9995	0.9995	0.9995	10535

Performance on Test (30%):

Accuracy : 0.9982285208148804
Precision: 0.9972850678733032
Recall : 0.99909338168631
F1-score : 0.9981884057971014
ROC-AUC : 0.9997743266102287

Classification report (4 decimals):

	precision	recall	f1-score	support
0	0.9991	0.9974	0.9983	2310
1	0.9973	0.9991	0.9982	2206
accuracy			0.9982	4516
macro avg	0.9982	0.9982	0.9982	4516
weighted avg	0.9982	0.9982	0.9982	4516

Files saved:

- LR_best_hyperparams_GA.json
- LR_GA_best_scores_70_30.csv
- LR_GA_search_results_70_30.csv
- LR_GA_best_model.joblib (best GA-tuned Logistic Regression model)

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>