

ya4rnrxunm

December 4, 2025

1 02 - Genetic Algorithm Hyperparameter Tuning for Decision Tree (GA + DT)

```
[1]: # =====  
# 02 - Genetic Algorithm Tuning for Decision Tree (GA + DT)  
# 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 DecisionTreeClassifier and GA hyperparameter search space  
# 4) Use GASearchCV to find best hyperparameters on Train set  
# 5) Evaluate best DT 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.tree import DecisionTreeClassifier  
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  
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
# -----
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 Decision Tree model
# -----
dt_model = DecisionTreeClassifier(random_state=42)

# -----
# 5) Define GA hyperparameter search space
# -----
param_grid = {
    "max_depth": Integer(2, 25),
    "min_samples_split": Integer(2, 20),
    "min_samples_leaf": Integer(1, 10),
    "criterion": Categorical(["gini", "entropy", "log_loss"]),
}

```

```

    "max_features": Categorical(["sqrt", "log2", None]),
}

# -----
# 6) Configure GA Search (on Train set only)
# -----
ga_search = GASearchCV(
    estimator=dt_model,
    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 (Decision Tree) completed.")

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

print("\n Best Hyperparameters (GA + DT):")
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 DT model on Train and Test sets
# -----
train_scores = evaluate_model(best_dt, X_train, y_train, dataset_name="Train_
↳(70%)")
test_scores = evaluate_model(best_dt, 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("DT_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("DT_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("DT_GA_search_results_70_30.csv", index=False)

```

```
# 11.4 Save the best GA-tuned DT model for later use (File 06 - Hybrid
↳Probability Fusion)
best_model_filename = "DT_GA_best_model.joblib"
joblib.dump(best_dt, best_model_filename)

print("\n Files saved:")
print("    - DT_best_hyperparams_GA.json")
print("    - DT_GA_best_scores_70_30.csv")
print("    - DT_GA_search_results_70_30.csv")
print(f"    - {best_model_filename} (best GA-tuned Decision Tree model)")

# -----
# 12) (Optional) Download files to your local machine
# -----
files.download("DT_best_hyperparams_GA.json")
files.download("DT_GA_best_scores_70_30.csv")
files.download("DT_GA_search_results_70_30.csv")
files.download("DT_GA_best_model.joblib")
```

136.0/136.0 kB

7.8 MB/s eta 0:00:00

Please upload: CKD_cleaned_dataset_EN.csv

<IPython.core.display.HTML object>

Saving CKD_cleaned_dataset_EN.csv to CKD_cleaned_dataset_EN.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	0.998615	0.000299462	0.999319	0.997862
1	40	0.998791	0.000266693	0.999319	0.998639
2	40	0.999125	0.000282948	0.99932	0.99864
3	40	0.999271	0.000148786	0.99932	0.99864
4	40	0.99932	7.41458e-08	0.99932	0.999319
5	40	0.99932	9.80433e-08	0.99932	0.999319
6	40	0.99932	1.09211e-07	0.99932	0.99932
7	40	0.99932	7.60399e-08	0.99932	0.99932
8	40	0.99932	3.39796e-08	0.99932	0.99932
9	40	0.99932	5.40754e-08	0.99932	0.99932
10	40	0.99932	5.78009e-08	0.99932	0.99932
11	40	0.99932	2.22045e-16	0.99932	0.99932
12	40	0.99932	2.22045e-16	0.99932	0.99932

GA Search (Decision Tree) completed.

Best Hyperparameters (GA + DT):

```
{'max_depth': 9, 'min_samples_split': 14, 'min_samples_leaf': 1, 'criterion':  
'gini', 'max_features': None}
```

Best CV F1 score: 0.9993198128154472

Performance on Train (70%):

Accuracy : 0.9996203132415757

Precision: 0.9996114241305615

Recall : 0.9996114241305615

F1-score : 0.9996114241305615

ROC-AUC : 0.999995853171402

Classification report (4 decimals):

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

Performance on Test (30%):

Accuracy : 0.9991142604074402

Precision: 0.99909338168631

Recall : 0.99909338168631

F1-score : 0.99909338168631

ROC-AUC : 0.9993300443889744

Classification report (4 decimals):

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

Files saved:

- DT_best_hyperparams_GA.json
- DT_GA_best_scores_70_30.csv
- DT_GA_search_results_70_30.csv
- DT_GA_best_model.joblib (best GA-tuned Decision Tree 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>