

vupnbi2xo

December 4, 2025

```
[1]: # =====  
# 05 - Genetic Algorithm Tuning for SVM (RBF) (GA + SVM)  
# 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 SVC (RBF kernel, probability=True) and GA search space  
# 4) Use GASearchCV to find best hyperparameters on Train set  
# (C, gamma, class_weight)  
# 5) Evaluate best SVM 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.svm import SVC  
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 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 SVM model (RBF kernel)
#     probability=True is required for predict_proba (for ROC/Hybrid fusion)
# -----
svm_model = SVC(
    kernel="rbf",
    probability=True,
    random_state=42
)

# -----
# 5) Define GA hyperparameter search space
#     (C, gamma, class_weight)
#     Using log-uniform ranges for C and gamma is common in SVM tuning.

```

```

# -----
param_grid = {
    "C": Continuous(1e-3, 1e3, distribution="log-uniform"),
    "gamma": Continuous(1e-4, 1e1, distribution="log-uniform"),
    "class_weight": Categorical([None, "balanced"]),
}

# -----
# 6) Configure GA Search (on Train set only)
# -----
ga_search = GASearchCV(
    estimator=svm_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 (SVM - RBF) completed.")

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

print("\n Best Hyperparameters (GA + SVM - RBF):")
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 SVM model on Train and Test sets
# -----
train_scores = evaluate_model(best_svm, X_train, y_train, dataset_name="Train_
↳(70%)")
test_scores = evaluate_model(best_svm, 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("SVM_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("SVM_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("SVM_GA_search_results_70_30.csv", index=False)

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

print("\n Files saved:")
print("  - SVM_best_hyperparams_GA.json")
print("  - SVM_GA_best_scores_70_30.csv")
print("  - SVM_GA_search_results_70_30.csv")
print(f"  - {best_model_filename} (best GA-tuned SVM RBF model)")

# -----
# 12) (Optional) Download files to your local machine
# -----
files.download("SVM_best_hyperparams_GA.json")
files.download("SVM_GA_best_scores_70_30.csv")
files.download("SVM_GA_search_results_70_30.csv")
files.download("SVM_GA_best_model.joblib")

```

136.0/136.0 kB

7.6 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	0.880449	0.301814	0.998834	0
1	40	0.998399	0.000400311	0.998834	0.997287
2	40	0.998655	0.000176813	0.999125	0.998447
3	40	0.998757	0.000213806	0.999222	0.998447
4	40	0.99897	0.000264407	0.999222	0.998447
5	40	0.999125	0.000170617	0.999222	0.998738
6	40	0.999159	0.000150996	0.999222	0.998738
7	40	0.999155	0.000223087	0.999222	0.998254
8	40	0.999222	0	0.999222	0.999222
9	40	0.999164	0.000148443	0.999222	0.99864

10	40	0.999184	0.000169047	0.999222	0.998447
11	40	0.999222	0	0.999222	0.999222
12	40	0.999217	2.11617e-05	0.999222	0.999125

GA Search (SVM - RBF) completed.

Best Hyperparameters (GA + SVM - RBF):

```
{'C': np.float64(18.21400694496058), 'gamma': np.float64(0.004923040304725387),
'class_weight': 'balanced'}
```

Best CV F1 score: 0.999222319366153

Performance on Train (70%):

Accuracy : 0.9992406264831514

Precision: 0.9998054853141413

Recall : 0.9986399844569652

F1-score : 0.9992223950233281

ROC-AUC : 0.9999995672874505

Classification report (4 decimals):

	precision	recall	f1-score	support
0	0.9987	0.9998	0.9993	5388
1	0.9998	0.9986	0.9992	5147
accuracy			0.9992	10535
macro avg	0.9993	0.9992	0.9992	10535
weighted avg	0.9992	0.9992	0.9992	10535

Performance on Test (30%):

Accuracy : 0.9982285208148804

Precision: 0.9972850678733032

Recall : 0.99909338168631

F1-score : 0.9981884057971014

ROC-AUC : 0.9999713492913855

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:

- SVM_best_hyperparams_GA.json
- SVM_GA_best_scores_70_30.csv
- SVM_GA_search_results_70_30.csv
- SVM_GA_best_model.joblib (best GA-tuned SVM RBF 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>