

xgepmfq2b

December 4, 2025

1 08 - SHAP Explainability for Best Hybrid Model (Hybrid_RF_DT)

```
[1]: # =====  
# 08 - SHAP Explainability for Best Hybrid Model (Hybrid_RF_DT)  
# Author: Nasser Qanabta  
# =====  
  
!pip install shap -q  
  
# 1) Imports  
import pandas as pd  
import numpy as np  
import json  
import shap  
import matplotlib.pyplot as plt  
  
from sklearn.model_selection import train_test_split  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.linear_model import LogisticRegression  
from google.colab import files  
  
plt.style.use("seaborn-v0_8")  
  
# -----  
# 2) Upload and load required files  
# -----  
  
print("Please upload:")  
print(" - CKD_cleaned_dataset_EN.csv")  
print(" - DT_best_hyperparams_GA.json")  
print(" - RF_best_hyperparams_GA.json")  
  
uploaded = files.upload()  
  
df = pd.read_csv("CKD_cleaned_dataset_EN.csv")  
print("Full cleaned dataset shape:", df.shape)
```

```

target_col = "Chronic_Kidney_Disease"
X_full = df.drop(columns=[target_col])
y_full = df[target_col]

print("Features shape:", X_full.shape, "Target shape:", y_full.shape)

with open("DT_best_hyperparams_GA.json", "r") as f:
    dt_best_params = json.load(f)

with open("RF_best_hyperparams_GA.json", "r") as f:
    rf_best_params = json.load(f)

print("\nBest DT params from GA:", dt_best_params)
print("Best RF params from GA:", rf_best_params)

# -----
# 3) Train/Test split (70% / 30%)
# -----
X_train, X_test, y_train, y_test = train_test_split(
    X_full,
    y_full,
    test_size=0.30,
    stratify=y_full,
    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) Rebuild base models (DT, RF) and train
# -----
best_dt = DecisionTreeClassifier(
    **dt_best_params,
    random_state=42
)

best_rf = RandomForestClassifier(
    **rf_best_params,
    random_state=42,
    n_jobs=-1
)

best_dt.fit(X_train, y_train)
best_rf.fit(X_train, y_train)

```

```

print("\nBase models (DT, RF) trained on Train (70%).")

# -----
# 5) Build Hybrid_RF_DT meta-features (DT_prob, RF_prob)
# -----
def get_prob_feature(model, X, name):
    prob = model.predict_proba(X)[: , 1]
    return pd.Series(prob, index=X.index, name=name)

dt_prob_train = get_prob_feature(best_dt, X_train, "DT_prob")
rf_prob_train = get_prob_feature(best_rf, X_train, "RF_prob")

X_hybrid_train = X_train.copy()
X_hybrid_train["DT_prob"] = dt_prob_train
X_hybrid_train["RF_prob"] = rf_prob_train

print("\nHybrid_RF_DT meta-features created:")
print("  X_hybrid_train:", X_hybrid_train.shape)

# -----
# 6) Train Hybrid_RF_DT meta-model (Logistic Regression)
# -----
hybrid_rf_dt = LogisticRegression(
    max_iter=1000,
    solver="lbfgs",
    random_state=42
)
hybrid_rf_dt.fit(X_hybrid_train, y_train)
print("\nHybrid_RF_DT meta-model trained on Train (70%).")

# -----
# 7) SHAP analysis for Hybrid_RF_DT (KernelExplainer)
# -----
# Background and sample for SHAP
if len(X_hybrid_train) > 2000:
    X_shap_background = X_hybrid_train.sample(n=200, random_state=42)
    X_shap_sample      = X_hybrid_train.sample(n=1000, random_state=42)
else:
    X_shap_background = X_hybrid_train.copy()
    X_shap_sample      = X_hybrid_train.copy()

print("\nSHAP background shape:", X_shap_background.shape)
print("SHAP sample shape      :", X_shap_sample.shape)

def model_predict_proba(X):
    if isinstance(X, np.ndarray):

```

```

        X_df = pd.DataFrame(X, columns=X_hybrid_train.columns)
    else:
        X_df = X
    return hybrid_rf_dt.predict_proba(X_df)[: , 1]

explainer = shap.KernelExplainer(
    model_predict_proba,
    X_shap_background
)

shap_values = explainer.shap_values(X_shap_sample)
print("SHAP values shape:", np.array(shap_values).shape)

# -----
# 8) SHAP summary plots (bar + beeswarm only)
# -----
# Bar plot
plt.figure()
shap.summary_plot(
    shap_values,
    X_shap_sample,
    plot_type="bar",
    show=False
)
plt.tight_layout()
bar_fname = "Hybrid_RF_DT_SHAP_summary_bar.png"
plt.savefig(bar_fname, dpi=300, bbox_inches="tight")
plt.show()
print(f"Saved: {bar_fname}")

# Beeswarm plot
plt.figure()
shap.summary_plot(
    shap_values,
    X_shap_sample,
    show=False
)
plt.tight_layout()
bee_fname = "Hybrid_RF_DT_SHAP_summary_beeswarm.png"
plt.savefig(bee_fname, dpi=300, bbox_inches="tight")
plt.show()
print(f"Saved: {bee_fname}")

# -----
# 9) Download SHAP summary plots only
# -----
files.download(bar_fname)

```

```
files.download(bee_fname)
```

Please upload:

- CKD_cleaned_dataset_EN.csv
- DT_best_hyperparams_GA.json
- RF_best_hyperparams_GA.json

<IPython.core.display.HTML object>

Saving RF_best_hyperparams_GA.json to RF_best_hyperparams_GA.json

Saving DT_best_hyperparams_GA.json to DT_best_hyperparams_GA.json

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,)

Best DT params from GA: {'max_depth': 9, 'min_samples_split': 14, 'min_samples_leaf': 1, 'criterion': 'gini', 'max_features': None}

Best RF params from GA: {'n_estimators': 75, 'max_depth': 4, 'min_samples_split': 12, 'min_samples_leaf': 1, 'criterion': 'gini', 'max_features': None, 'bootstrap': True}

Stratified split (70/30):

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

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

WARNING:shap:Using 200 background data samples could cause slower run times. Consider using shap.sample(data, K) or shap.kmeans(data, K) to summarize the background as K samples.

Base models (DT, RF) trained on Train (70%).

Hybrid_RF_DT meta-features created:

X_hybrid_train: (10535, 22)

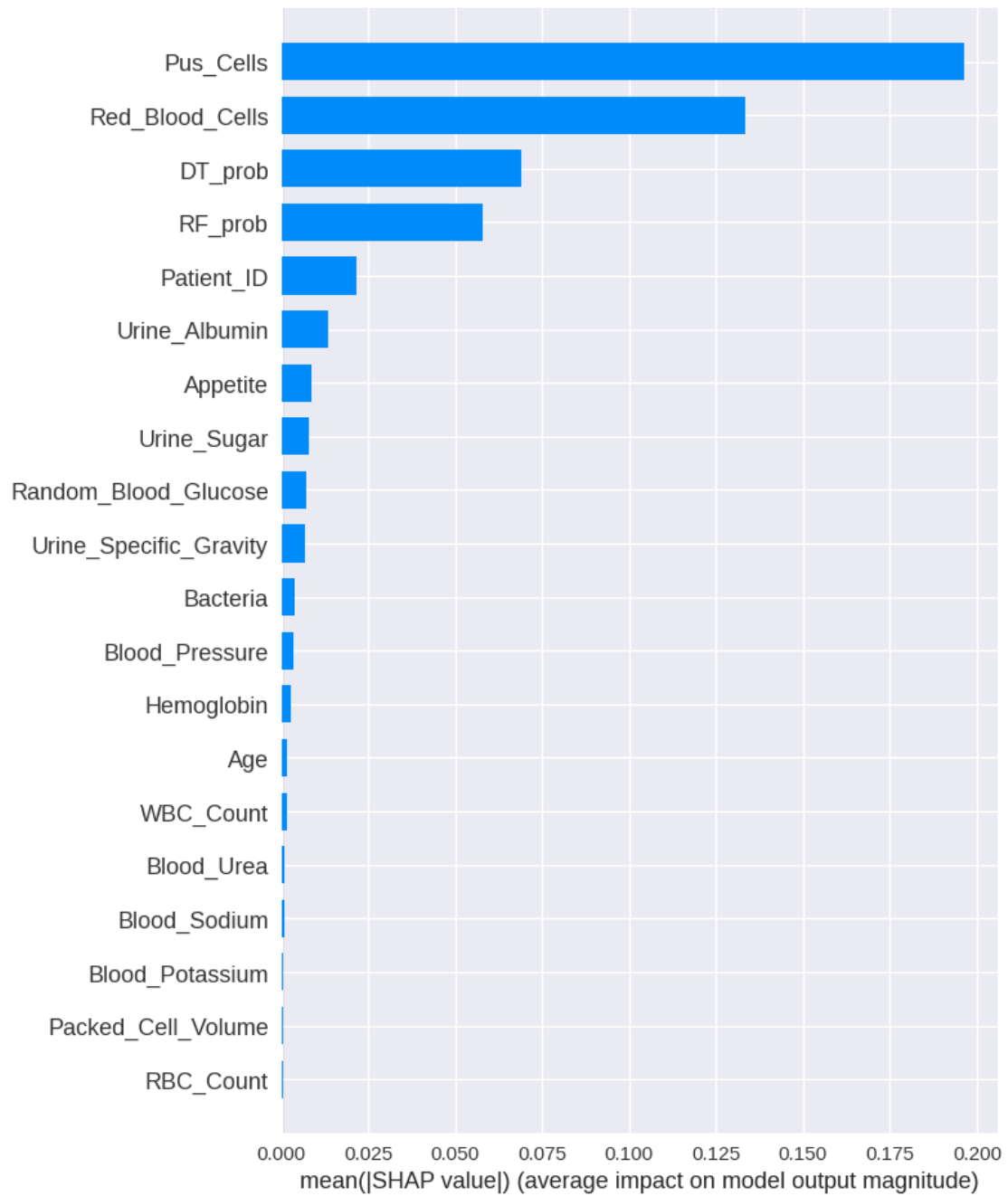
Hybrid_RF_DT meta-model trained on Train (70%).

SHAP background shape: (200, 22)

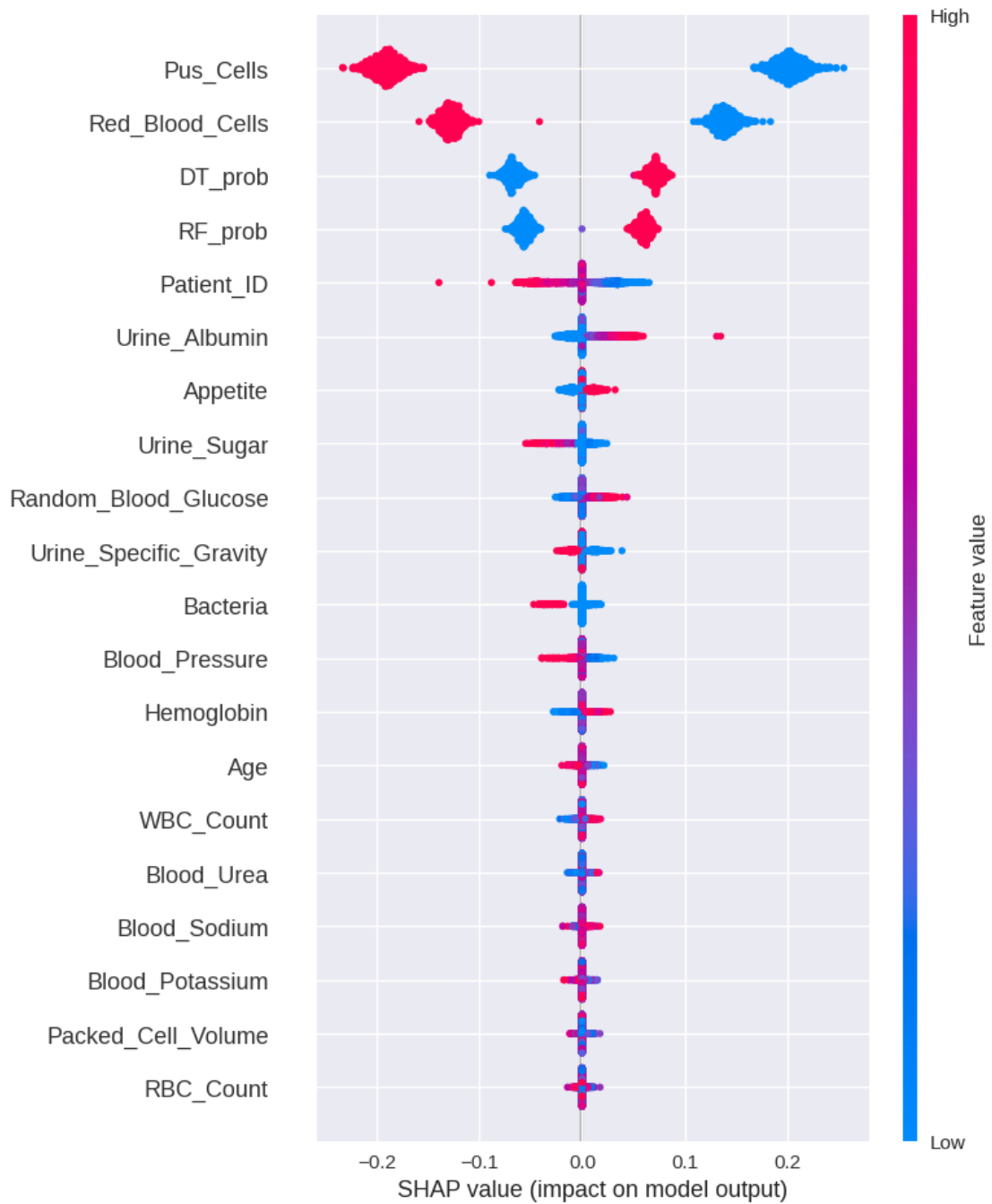
SHAP sample shape : (1000, 22)

0%| | 0/1000 [00:00<?, ?it/s]

SHAP values shape: (1000, 22)



Saved: Hybrid_RF_DT_SHAP_summary_bar.png



Saved: Hybrid_RF_DT_SHAP_summary_beeswarm.png

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>