ECG Classification with Machine Learning Models
Comparison of Decision Tree, Random Forest, XGBoost, and CNN

Mohamed Charfeddine

MIT-BIH Arrhythmia Database Analysis

August 26, 2025

1 Data Download and Preprocessing

The first step involves downloading and preprocessing the MIT-BIH Arrhythmia Database. The following code downloads multiple records and extracts ECG segments for classification.

```
import wfdb
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from imblearn.over_sampling import SMOTE
import xgboost as xgb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout
import warnings
warnings.filterwarnings(’ignore’)

# Télécharger et traiter les données MIT-BIH
def download_mitbih_data():
    records = [’100’, ’101’, ’102’, ’103’, ’104’]
    all_signals = []
    all_labels = []

    for record in records:
        try:
            print(f"Téléchargement de l’enregistrement {record}...")

            # Télécharger les données
            signal, fields = wfdb.rdsamp(record, pn_dir=’mitdb’)
            annotation = wfdb.rdann(record, ’atr’, pn_dir=’mitdb’)

            print(f"Signal shape: {signal.shape}")
            print(f"Nombre d’annotations: {len(annotation.sample)}")

            # Utiliser le premier canal
            ecg_signal = signal[:, 0]

            # Créer des segments autour de chaque battement détecté
            for i, (sample_idx, symbol) in enumerate(zip(annotation.sample,
                annotation.symbol)):
                # Prendre une fenêtre de 200 points autour de chaque battement
                start_idx = max(0, sample_idx - 100)
                end_idx = min(len(ecg_signal), sample_idx + 100)

                if end_idx - start_idx >= 200:
                    segment = ecg_signal[start_idx:start_idx + 200]

                    # Mapper les annotations vers 5 classes principales
                    if symbol in [’N’, ’L’, ’R’, ’e’, ’j’, ’.’]:
                        label = ’N’  # Normal
```

```python
                    elif symbol in ['A', 'a', 'J', 'S']:
                        label = 'S'  # Supraventricular
                    elif symbol in ['V', 'E']:
                        label = 'V'  # Ventricular
                    elif symbol in ['F']:
                        label = 'F'  # Fusion
                    else:
                        label = 'Q'  # Unclassifiable

                    all_signals.append(segment)
                    all_labels.append(label)

            print(f"Segments extraits de {record}: {len([l for l in all_labels if l in ['N', 'S', 'V', 'F', 'Q']])}")

        except Exception as e:
            print(f"Erreur avec {record}: {e}")
            continue

    if len(all_signals) == 0:
        print("Aucune donnée téléchargée. Création de données synthétiques pour test...")
        # Créer des données factices pour le test
        np.random.seed(42)
        for i in range(1000):
            # Signal ECG synthétique
            t = np.linspace(0, 1, 200)
            signal = np.sin(2*np.pi*t) + 0.5*np.sin(4*np.pi*t) + np.random.normal(0, 0.1, 200)
            all_signals.append(signal)

            # Labels aléatoires
            labels = ['N', 'S', 'V', 'F', 'Q']
            all_labels.append(np.random.choice(labels))

        print("1000 échantillons synthétiques créés pour le test")

    return np.array(all_signals), np.array(all_labels)

print("Téléchargement des données MIT-BIH...")
X, y = download_mitbih_data()
print(f"Données finales: {len(X)} segments")
print(f"Forme des données: {X.shape}")

# Vérification que les données ne sont pas vides
if len(X) == 0:
    raise ValueError("Aucune donnée n'a été téléchargée!")
```

1.1 Downloading and Preprocessing MIT-BIH ECG Data

This block of code handles loading and preparing the ECG signals for classification:

- **Imports**: Uses `wfdb` to access the MIT-BIH dataset, `numpy` and `pandas` for data manipulation, `sklearn` for splitting and evaluation, `SMOTE` to handle class imbalance, and `XGBoost` / Keras layers for machine learning and deep learning models.
- **Function `download_mitbih_data()`**:
  - Downloads ECG signals from 5 selected records of the MIT-BIH Arrhythmia Database.
  - Extracts the first ECG channel from each recording.
  - Creates segments of 200 data points around each annotated heartbeat.
  - Maps annotations to 5 main classes: Normal (N), Supraventricular (S), Ventricular (V), Fusion (F), and Unclassifiable (Q).
- **Fallback to synthetic data**: If the dataset is not downloaded correctly, the code generates 1000 synthetic ECG segments with random labels for testing purposes.
- **Outputs**:

– `X`: array of ECG segments (samples × 200 points)
– `y`: array of corresponding labels

In simple terms: This step converts raw ECG recordings into labeled segments suitable for training machine learning models, so that each heartbeat can be classified according to its type.

Output:

```
Téléchargement des données MIT-BIH...
Téléchargement de l'enregistrement 100...
Signal shape: (650000, 2)
Nombre d'annotations: 2274
Segments extraits de 100: 2271
Téléchargement de l'enregistrement 101...
Signal shape: (650000, 2)
Nombre d'annotations: 1874
Segments extraits de 101: 4143
Téléchargement de l'enregistrement 102...
Signal shape: (650000, 2)
Nombre d'annotations: 2192
Segments extraits de 102: 6334
Téléchargement de l'enregistrement 103...
Signal shape: (650000, 2)
Nombre d'annotations: 2091
Segments extraits de 103: 8424
Téléchargement de l'enregistrement 104...
Signal shape: (650000, 2)
Nombre d'annotations: 2311
Segments extraits de 104: 10733
Données finales: 10733 segments
Forme des données: (10733, 200)
```

The preprocessing successfully extracted 10,733 ECG segments of 200 samples each from 5 MIT-BIH records.

2 Data Normalization and Train/Test Split

The data is normalized using MinMaxScaler and split into training and testing sets. SMOTE is applied to handle class imbalance.

```python
# Normalisation des signaux
from sklearn.preprocessing import MinMaxScaler, LabelEncoder

scaler = MinMaxScaler()
X_normalized = scaler.fit_transform(X)

# Encodage des labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

print("Classes disponibles:", label_encoder.classes_)
print("Distribution des classes:")
unique, counts = np.unique(y, return_counts=True)
for u, c in zip(unique, counts):
    print(f"{u}: {c}")

# Division train/test
X_train, X_test, y_train, y_test = train_test_split(
    X_normalized, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
)

# Application de SMOTE pour équilibrer les classes
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

print(f"Avant SMOTE: {len(X_train)} échantillons")
print(f"Après SMOTE: {len(X_train_balanced)} échantillons")
```

2.1 Data Normalization, Encoding, and Class Balancing

After extracting the ECG segments, the data is prepared for machine learning models:

- **Normalization**: The ECG signals are normalized using `MinMaxScaler` to scale the values between 0 and 1. This helps the models train faster and avoids giving too much weight to signals with higher amplitudes.
- **Label Encoding**: The heartbeat classes (N, S, V, F, Q) are converted into numeric values using `LabelEncoder`, which is required for machine learning models.
- **Train/Test Split**: The data is split into training (80%) and testing (20%) sets using `train_test_split`. The `stratify` option ensures the class distribution is preserved in both sets.
- **SMOTE (Synthetic Minority Oversampling Technique)**: Some heartbeat classes are very rare, e.g., Ventricular (V) or Supraventricular (S). SMOTE creates synthetic samples of these minority classes in the training set to prevent the models from being biased toward the majority class (Normal).
- **Resulting Sizes**: Before SMOTE, the training set has 8,586 samples. After SMOTE, it increases to 20,608 samples with balanced classes.

In simple terms: This step makes sure the input data is scaled, the labels are numeric, the dataset is divided for evaluation, and all classes are fairly represented so the models can learn properly.

Output:

```
Classes disponibles: ['N' 'Q' 'S' 'V']
Distribution des classes:
N: 6440
Q: 4248
S: 38
V: 7
Avant SMOTE: 8586 échantillons
Après SMOTE: 20608 échantillons
```

The dataset contains 4 ECG beat classes: Normal (N), Atrial premature (Q), Supraventricular premature (S), and Ventricular premature (V). SMOTE significantly increased the training set size to balance the classes.

3 Model Training and Evaluation

Four different machine learning models are trained and evaluated: Decision Tree, Random Forest, XGBoost, and CNN.

```python
# Dictionnaire pour stocker les résultats
results = {}

# 1. Decision Tree
print("Entraînement Decision Tree...")
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train_balanced, y_train_balanced)
y_pred_dt = dt.predict(X_test)

results['Decision Tree'] = {
    'Accuracy': accuracy_score(y_test, y_pred_dt) * 100,
    'Precision': precision_score(y_test, y_pred_dt, average='weighted') * 100,
    'Recall': recall_score(y_test, y_pred_dt, average='weighted') * 100,
    'F1-Score': f1_score(y_test, y_pred_dt, average='weighted') * 100
}

# 2. Random Forest
print("Entraînement Random Forest...")
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train_balanced, y_train_balanced)
y_pred_rf = rf.predict(X_test)

results['Random Forest'] = {
    'Accuracy': accuracy_score(y_test, y_pred_rf) * 100,
```

```
    'Precision': precision_score(y_test, y_pred_rf, average='weighted') * 100,
    'Recall': recall_score(y_test, y_pred_rf, average='weighted') * 100,
    'F1-Score': f1_score(y_test, y_pred_rf, average='weighted') * 100
}

# 3. XGBoost
print("Entraînement XGBoost...")
xgb_model = xgb.XGBClassifier(random_state=42)
xgb_model.fit(X_train_balanced, y_train_balanced)
y_pred_xgb = xgb_model.predict(X_test)

results['XGBoost'] = {
    'Accuracy': accuracy_score(y_test, y_pred_xgb) * 100,
    'Precision': precision_score(y_test, y_pred_xgb, average='weighted') * 100,
    'Recall': recall_score(y_test, y_pred_xgb, average='weighted') * 100,
    'F1-Score': f1_score(y_test, y_pred_xgb, average='weighted') * 100
}

# 4. CNN
print("Entraînement CNN...")
# Reshape pour CNN (samples, timesteps, features)
X_train_cnn = X_train_balanced.reshape(X_train_balanced.shape[0],
    X_train_balanced.shape[1], 1)
X_test_cnn = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

print(f"Shape des données CNN: {X_train_cnn.shape}")

# Convertir les labels en categorical
from tensorflow.keras.utils import to_categorical
y_train_categorical = to_categorical(y_train_balanced)
y_test_categorical = to_categorical(y_test)

# Obtenir la taille d'entrée dynamiquement
input_shape = X_train_cnn.shape[1]
print(f"Taille d'entrée pour CNN: {input_shape}")

# Modèle CNN adapté à la taille des données
model = Sequential([
    Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(input_shape, 1)),
    MaxPooling1D(pool_size=2),
    Conv1D(filters=64, kernel_size=3, activation='relu'),
    MaxPooling1D(pool_size=2),
    Flatten(),
    Dense(100, activation='relu'),
    Dropout(0.5),
    Dense(len(label_encoder.classes_), activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Entraînement (réduit pour le test)
print("Début de l'entraînement CNN...")
history = model.fit(X_train_cnn, y_train_categorical,
                   epochs=10, batch_size=32,
                   validation_split=0.2, verbose=1)

# Prédictions CNN
print("Prédictions CNN...")
y_pred_cnn_prob = model.predict(X_test_cnn, verbose=0)
y_pred_cnn = np.argmax(y_pred_cnn_prob, axis=1)

results['CNN'] = {
    'Accuracy': accuracy_score(y_test, y_pred_cnn) * 100,
    'Precision': precision_score(y_test, y_pred_cnn, average='weighted') * 100,
    'Recall': recall_score(y_test, y_pred_cnn, average='weighted') * 100,
    'F1-Score': f1_score(y_test, y_pred_cnn, average='weighted') * 100
}
```

3.1 Model Training and Evaluation

This block trains and evaluates four different models on the ECG data: Decision Tree, Random Forest, XGBoost, and a Convolutional Neural Network (CNN).

- **Decision Tree**: A simple tree-based model that splits the data based on features to classify heartbeats. It is trained on the balanced training set and tested on the hold-out test set. Accuracy, precision, recall, and F1-score are recorded.
- **Random Forest**: An ensemble of multiple decision trees that vote on the final prediction. It usually performs better than a single Decision Tree because it reduces overfitting.
- **XGBoost**: A gradient boosting model that builds trees sequentially, focusing on correcting errors from previous trees. It is known for high performance on structured data.
- **CNN (Convolutional Neural Network)**:
  - The ECG segments are reshaped into 3D arrays (samples, timesteps, features) suitable for CNN input.
  - Labels are converted to categorical format for multi-class classification.
  - The CNN has convolutional layers to extract patterns, max-pooling layers to reduce dimensionality, a fully connected layer, and a dropout layer to reduce overfitting.
  - The output layer uses softmax activation to predict probabilities for each heartbeat class.
  - The model is trained for 10 epochs with a batch size of 32, and predictions are made on the test set.
- **Results**: For each model, four evaluation metrics are calculated: Accuracy, Precision, Recall, and F1-Score. These are stored in a dictionary for comparison.

In simple terms: This step trains different machine learning and deep learning models on the ECG data and measures how well they can classify heartbeats into the different types.

Output:

```
Entrainement Decision Tree...
Entrainement Random Forest...
Entrainement XGBoost...
Entrainement CNN...
Shape des donnees CNN: (20608, 200, 1)
Taille d'entree pour CNN: 200
Debut de l'entrainement CNN...
Epoch 1/10
516/516 ==================== 8s 11ms/step - accuracy: 0.6299 - loss: 0.7730 -
    ↪ val_accuracy: 1.0000 - val_loss: 0.0286
Epoch 2/10
516/516 ==================== 6s 12ms/step - accuracy: 0.9045 - loss: 0.2612 -
    ↪ val_accuracy: 1.0000 - val_loss: 0.0014
Epoch 3/10
516/516 ==================== 6s 12ms/step - accuracy: 0.9422 - loss: 0.1641 -
    ↪ val_accuracy: 1.0000 - val_loss: 0.0053
Epoch 4/10
516/516 ==================== 6s 11ms/step - accuracy: 0.9537 - loss: 0.1307 -
    ↪ val_accuracy: 1.0000 - val_loss: 7.4543e-04
Epoch 5/10
516/516 ==================== 11s 13ms/step - accuracy: 0.9629 - loss: 0.1088 -
    ↪ val_accuracy: 1.0000 - val_loss: 0.0011
Epoch 6/10
516/516 ==================== 6s 13ms/step - accuracy: 0.9645 - loss: 0.1025 -
    ↪ val_accuracy: 1.0000 - val_loss: 0.0036
Epoch 7/10
516/516 ==================== 5s 10ms/step - accuracy: 0.9689 - loss: 0.0904 -
    ↪ val_accuracy: 1.0000 - val_loss: 2.9113e-04
Epoch 8/10
516/516 ==================== 6s 11ms/step - accuracy: 0.9708 - loss: 0.0837 -
    ↪ val_accuracy: 1.0000 - val_loss: 3.8674e-04
Epoch 9/10
516/516 ==================== 5s 10ms/step - accuracy: 0.9742 - loss: 0.0753 -
    ↪ val_accuracy: 1.0000 - val_loss: 3.5769e-04
```

```
Epoch 10/10
516/516 ==================== 5s 9ms/step - accuracy: 0.9739 - loss: 0.0776 -
    ↪ val_accuracy: 1.0000 - val_loss: 2.3513e-04
Predictions CNN...
```

The CNN training shows excellent convergence with validation accuracy reaching 100% and very low validation loss.

4 Results Comparison

The following section presents the results obtained from our experiments compared to the expected results from the reference article.

```python
# Création du tableau de résultats
import pandas as pd

df_results = pd.DataFrame(results).T
df_results = df_results.round(2)

print("\n=== RÉSULTATS (Tableau 2 de l'article) ===")
print(df_results)

# Comparaison avec les résultats attendus de l'article
expected_results = {
    'CNN': {'Accuracy': 99, 'Precision': 98, 'Recall': 98, 'F1-Score': 98},
    'Decision Tree': {'Accuracy': 78, 'Precision': 73, 'Recall': 76, 'F1-Score': 93},
    'Random Forest': {'Accuracy': 98, 'Precision': 97, 'Recall': 98, 'F1-Score': 98},
    'XGBoost': {'Accuracy': 97, 'Precision': 97, 'Recall': 97, 'F1-Score': 97}
}

print("\n=== RÉSULTATS ATTENDUS (Article) ===")
df_expected = pd.DataFrame(expected_results).T
print(df_expected)

print("\n=== DIFFÉRENCES ===")
for model in results.keys():
    if model in expected_results:
        print(f"\n{model}:")
        for metric in ['Accuracy', 'Precision', 'Recall', 'F1-Score']:
            obtained = results[model][metric]
            expected = expected_results[model][metric]
            diff = obtained - expected
            print(f"  {metric}: {obtained:.1f}% (attendu: {expected}%, diff:
                {diff:+.1f}%)")
```

4.1 Results Comparison

After training all models, we organize and compare their performance metrics:

- **Results Table**: The metrics for each model—Accuracy, Precision, Recall, and F1-Score—are stored in a dictionary and converted into a `pandas` DataFrame for easier visualization.
- **Expected Results**: The reference values from the original article are provided for comparison. This helps evaluate if our models achieve similar performance.
- **Differences**: For each model and metric, we calculate the difference between the obtained result and the expected value. This highlights where our implementation performs better or worse than the published study.

In simple terms: This step summarizes all model performances in a table, compares them to the literature, and shows the differences, making it easy to see which models work best and how our results align with the article.

Output:

```
=== RÉSULTATS (Tableau 2 de l'article) ===
               Accuracy  Precision  Recall  F1-Score
```

```
Decision Tree      98.46       98.59   98.46      98.52
Random Forest      99.49       99.44   99.49      99.38
XGBoost            99.30       99.22   99.30      99.26
CNN                98.98       99.37   98.98      99.12

=== RÉSULTATS ATTENDUS (Article) ===
               Accuracy  Precision  Recall  F1-Score
CNN                  99         98      98        98
Decision Tree        78         73      76        93
Random Forest        98         97      98        98
XGBoost              97         97      97        97

=== DIFFÉRENCES ===

Decision Tree:
  Accuracy: 98.5% (attendu: 78%, diff: +20.5%)
  Precision: 98.6% (attendu: 73%, diff: +25.6%)
  Recall: 98.5% (attendu: 76%, diff: +22.5%)
  F1-Score: 98.5% (attendu: 93%, diff: +5.5%)

Random Forest:
  Accuracy: 99.5% (attendu: 98%, diff: +1.5%)
  Precision: 99.4% (attendu: 97%, diff: +2.4%)
  Recall: 99.5% (attendu: 98%, diff: +1.5%)
  F1-Score: 99.4% (attendu: 98%, diff: +1.4%)

XGBoost:
  Accuracy: 99.3% (attendu: 97%, diff: +2.3%)
  Precision: 99.2% (attendu: 97%, diff: +2.2%)
  Recall: 99.3% (attendu: 97%, diff: +2.3%)
  F1-Score: 99.3% (attendu: 97%, diff: +2.3%)

CNN:
  Accuracy: 99.0% (attendu: 99%, diff: -0.0%)
  Precision: 99.4% (attendu: 98%, diff: +1.4%)
  Recall: 99.0% (attendu: 98%, diff: +1.0%)
  F1-Score: 99.1% (attendu: 98%, diff: +1.1%)
```

5 Summary and Analysis

From the experiments I ran, all the models gave very high results on the MIT-BIH ECG classification task:

- **Random Forest** gave the best accuracy (99.49%).
- **XGBoost** was also very strong (99.30%), with balanced precision and recall.
- **CNN** worked well (98.98%), close to what was reported in the original paper.
- **Decision Tree** was surprisingly good (98.46%), much higher than the 78% mentioned in the literature.

These results are higher than what the article reported. I think this could be because:

1. I only used a small part of the dataset (5 records), which makes the task easier.
2. SMOTE helped balance the classes so the models did not get biased.
3. Normalizing the data with MinMaxScaler also helped the training.
4. The extracted ECG segments I worked with were of good quality.

Overall, all the models did very well in classifying the ECG signals. Ensemble methods like Random Forest and XGBoost were slightly better than the single models, but the big difference in accuracy compared to the literature is probably due to the simplified dataset and preprocessing choices.

6 What I Learned

Working on this project taught me a lot, even though I'm not a machine learning expert. Here's what I got out of it:

- I learned how to download and prepare ECG signals from the MIT-BIH dataset, including cutting them into segments around each heartbeat and labeling them.
- I understood why we need to normalize data and convert labels into numbers before feeding them to models.
- I saw why splitting the data into training and test sets is important to check if the models really learn something.
- I discovered SMOTE, which creates synthetic samples for rare heartbeat types so the models don't ignore them.
- I got hands-on experience with different models: Decision Tree, Random Forest, XGBoost, and CNN, and learned the basic idea of how each one works.
- I learned how to measure model performance using Accuracy, Precision, Recall, and F1-Score, and what these metrics actually mean.
- I practiced comparing my results with the original paper and thinking about why my numbers might be higher or lower.
- I realized that even if I didn't write all the code from scratch, understanding what each step does is what really matters.
- Overall, I learned how to go from raw ECG signals to trained models, check their results, and explain the process clearly.

In short, this project gave me practical experience and made machine learning on real biomedical data much clearer and less intimidating. Thank you for reviewing my work.