# Fiche Technique - Attaques Adversariales sur Modeles ML

Equipe de Cybersecurite

18 novembre 2025

## Introduction

Cette fiche explique comment tester la robustesse de nos modeles ML (Random Forest, KNN, Decision Tree) face aux attaques adversariales dans le contexte de detection d'intrusion.

## 1 Concepts Fondamentaux

### 1.1 Attaque Adversariale : Qu'est-ce que c'est ?

**Analogie :** Un intrus qui modifie legerement son apparence pour tromper un systeme de securite.
**Exemple concret :** Notre modele detecte une attaque DDoS normalement. On modifie 2-3 features (duree, nombre de packets) de facon subtile. Le modele pense maintenant que c'est du trafic normal.

### 1.2 Pourquoi nos modeles sont vulnerables ?

— **Random Forest :** Sensible aux perturbations sur les features importantes
— **KNN :** Peut etre trompe en deplacant les points pres des frontieres
— **Decision Tree :** Une petite modification peut changer la feuille de decision

### 1.3 Choix des methodes d'attaque - par Copilot

J'ai choisi ces deux exemples (HopSkipJump et surrogate+FGSM) car ils sont les plus adaptes aux modeles scikit-learn (arbres, KNN, random forest) pour les raisons suivantes :

**HopSkipJump (decision-based, black-box)**

- Fonctionne sans acces au gradient, donc compatible avec les modeles non-differentiables (arbres, KNN)
- C'est une attaque generique, applicable a tout classificateur qui donne une prediction (score ou classe)
- Represente le scenario "attaquant externe" qui ne connait pas l'architecture du modele

**Surrogate + FGSM (transfert)**

- Les attaques par gradient (FGSM, PGD) ne marchent que sur des modeles differentiables (reseaux de neurones)
- Pour attaquer un modele non-differentiable, on entraine un "surrogate" (reseau Keras) sur les memes donnees ou sur les predictions du modele cible, puis on genere des adversariaux sur le surrogate et on teste leur effet sur le modele cible
- Cette approche est tres utilisee en recherche pour tester la robustesse des modeles non-differentiables

**Pourquoi pas d'autres attaques ?**

- Les attaques directes par gradient (FGSM, PGD, Carlini-Wagner) ne fonctionnent pas sur scikit-learn trees/KNN
- Les attaques par heuristique (Square, Boundary, NES) sont aussi possibles, mais HopSkipJump est la plus connue et la plus efficace pour le "decision-based"
- Les attaques par perturbation aleatoire ou optimisation evolutive sont moins efficaces ou plus lentes

## 2 Strategies d'Attaque Implementees

### 2.1 1. Attaque HopSkipJump (Black-Box)

**Idee :** Methode generique qui fonctionne avec n'importe quel modele sans connaitre ses internes.
**Avantage :** Simple a mettre en œuvre, universelle
**Comment ca marche :** Teste plein de petites modifications jusqu'a trouver celles qui trompent le modele.

Listing 1 – Exemple minimal HopSkipJump

```
# Exemple minimal - HopSkipJump (black-box / decision-based)
import numpy as np
from art.estimators.classification import SklearnClassifier
from art.attacks.evasion import HopSkipJump
# from sklearn.ensemble import RandomForestClassifier  # votre modele deja entraine

# Supposons : model = RandomForestClassifier(...); model.fit(X_train, y_train)
# et X_test, y_test disponibles

# Definir clip_values selon vos features (ex: (0., 1.) ou (X_min, X_max))
clip_values = (X_train.min(axis=0).astype(float), X_train.max(axis=0).astype(float))
# ART attend un tuple (min, max). Si vous preferez uniformiser : (0., 1.)

art_clf = SklearnClassifier(model=model, clip_values=(0.0, 1.0))  # ajuster clip_values
```

```
attack = HopSkipJump(classifier=art_clf, max_iter=50, max_eval=1000,
    init_eval=10)
# Pour un petit test, utiliser un sous-ensemble
X_sub = X_test[:100].astype(np.float32)

# Generer adversariaux (peut prendre du temps selon max_iter)
X_adv = attack.generate(x=X_sub)

# Evaluer transfert/effet sur le modele
preds_clean = model.predict(X_sub)
preds_adv = model.predict(X_adv)

from sklearn.metrics import accuracy_score
print("Accuracy clean:", accuracy_score(y_test[:100], preds_clean))
print("Accuracy adv :", accuracy_score(y_test[:100], preds_adv))
```

### 2.2 2. Attaque par Transfert (Surrogate)

**Idee :**

1. Entrainer un reseau neuronal qui imite notre modele cible
2. Generer des attaques sur ce reseau (plus facile)
3. Tester si ces attaques fonctionnent sur le modele original

**Avantage :** Plus efficace, permet d'attaquer des modeles non-differentiables

Listing 2 – Exemple minimal Surrogate + FGSM

```
# Exemple minimal - surrogate (Keras) + FGSM
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, InputLayer
from art.estimators.classification import KerasClassifier
from art.attacks.evasion import FastGradientMethod
from sklearn.metrics import accuracy_score

# 1) creer les labels "surrogate" comme predictions du target
y_surrogate = model.predict(X_train)              # si classification multi-
    classe, shape (n,)
# si model.predict renvoie classes, transformer en one-hot si besoin pour
    Keras
from tensorflow.keras.utils import to_categorical
num_classes = len(np.unique(y_surrogate))
y_surrogate_oh = to_categorical(y_surrogate, num_classes)

# 2) petit reseau Keras comme surrogate
surrogate = Sequential([
    InputLayer(input_shape=(X_train.shape[1],)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')
])
surrogate.compile(optimizer='adam', loss='categorical_crossentropy', metrics
    =['accuracy'])

# 3) Normaliser / clip_values correctement (ici on suppose features deja
    dans [0,1])
```

```
surrogate.fit(X_train.astype(np.float32), y_surrogate_oh, epochs=10,
    batch_size=64, verbose=1)

# 4) wrapper ART
keras_clf = KerasClassifier(model=surrogate, clip_values=(0.0, 1.0))

# 5) creer adversariaux via FGSM
attack = FastGradientMethod(estimator=keras_clf, eps=0.05)  # eps =
    magnitude
X_sub = X_test[:500].astype(np.float32)
X_adv = attack.generate(x=X_sub)

# 6) tester l'effet sur vos modeles scikit-learn
preds_clean = model.predict(X_sub)
preds_adv = model.predict(X_adv)
print("Acc clean:", accuracy_score(y_test[:len(X_sub)], preds_clean))
print("Acc adv :", accuracy_score(y_test[:len(X_sub)], preds_adv))
```

# 3 Guide d'Implementation

## 3.1 Code Complet - Demo d'Attaques

Listing 3 – Script complet demo_attack.py

```
"""Demo script: entrainer vos modeles et effectuer deux attaques
    adversariales

- Attack 1: HopSkipJump (decision-based / black-box) via ART
- Attack 2: Surrogate (Keras) + FGSM puis test de transferabilite

Usage:
    python scripts/attacks/demo_attack.py

Pre-requis:
    pip install adversarial-robustness-toolbox tensorflow scikit-learn numpy
"""
import warnings
warnings.filterwarnings('ignore')

import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

from scripts.models.random_forest import train_random_forest
from scripts.models.knn import train_knn
from scripts.models.decision_tree import train_decision_tree

try:
    from art.estimators.classification import SklearnClassifier,
        KerasClassifier
    from art.attacks.evasion import HopSkipJump, FastGradientMethod
except Exception as e:
    raise ImportError(
        "This demo requires 'adversarial-robustness-toolbox'.\n"
        "Install with: pip install adversarial-robustness-toolbox tensorflow
            "
    ) from e

```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, InputLayer
from tensorflow.keras.utils import to_categorical


def make_scaled_dataset(n_samples=2000, n_features=20, n_classes=2, random_state=0):
    X, y = make_classification(
        n_samples=n_samples,
        n_features=n_features,
        n_informative=int(n_features / 2),
        n_redundant=0,
        n_classes=n_classes,
        random_state=random_state,
    )
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=random_state)
    scaler = MinMaxScaler(feature_range=(0.0, 1.0))
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    return X_train.astype(np.float32), X_test.astype(np.float32), y_train.astype(np.int64), y_test.astype(np.int64)


def train_models(X_train, y_train):
    # Utilise vos fonctions d'entrainement
    rf_model, rf_cv = train_random_forest(X_train, y_train, n_estimators=50, max_depth=8)
    knn_model, knn_cv = train_knn(X_train, y_train, n_neighbors=5)
    dt_model, dt_cv = train_decision_tree(X_train, y_train, max_depth=8)
    return rf_model, knn_model, dt_model


def hopskip_attack_and_eval(model, X_sub, y_sub, clip=(0.0, 1.0), max_iter=20):
    art_clf = SklearnClassifier(model=model, clip_values=clip)
    attack = HopSkipJump(classifier=art_clf, max_iter=max_iter)
    print("Generating HopSkipJump adversarials (this may take a while)...")
    X_adv = attack.generate(x=X_sub)
    preds_clean = model.predict(X_sub)
    preds_adv = model.predict(X_adv)
    acc_clean = (preds_clean == y_sub).mean()
    acc_adv = (preds_adv == y_sub).mean()
    return acc_clean, acc_adv


def surrogate_fgsm_transfer(rf_model, knn_model, dt_model, X_train, y_train, X_sub, y_sub, clip=(0.0, 1.0), eps=0.05):
    # Construire labels a partir du modele cible (ici on s'appuie sur rf_model comme source des etiquettes)
    surrogate_labels = rf_model.predict(X_train)
    num_classes = len(np.unique(surrogate_labels))
    surrogate_labels_oh = to_categorical(surrogate_labels, num_classes)

    # Petit reseau Keras
    surrogate = Sequential([
        InputLayer(input_shape=(X_train.shape[1],)),
        Dense(128, activation='relu'),
        Dense(64, activation='relu'),
        Dense(num_classes, activation='softmax')
    ])
```

```
    surrogate.compile(optimizer='adam', loss='categorical_crossentropy',
        metrics=['accuracy'])
    surrogate.fit(X_train, surrogate_labels_oh, epochs=5, batch_size=64,
        verbose=0)

    keras_clf = KerasClassifier(model=surrogate, clip_values=clip)
    fgsm = FastGradientMethod(estimator=keras_clf, eps=eps)
    print("Generating FGSM adversarials on surrogate...")
    X_adv = fgsm.generate(x=X_sub)

    # Evaluer la transferabilite sur les 3 modeles
    results = {}
    for name, model in (('RandomForest', rf_model), ('KNN', knn_model), ('
        DecisionTree', dt_model)):
        preds_clean = model.predict(X_sub)
        preds_adv = model.predict(X_adv)
        acc_clean = (preds_clean == y_sub).mean()
        acc_adv = (preds_adv == y_sub).mean()
        results[name] = (acc_clean, acc_adv)
    return results


def main():
    print("Preparing dataset...")
    X_train, X_test, y_train, y_test = make_scaled_dataset(n_samples=3000,
        n_features=20, n_classes=2)

    print("Training models (this is fast with small data)...")
    rf_model, knn_model, dt_model = train_models(X_train, y_train)

    # Sous-ensemble pour attaques (plus rapide)
    X_sub = X_test[:100]
    y_sub = y_test[:100]

    print('\n=== HopSkipJump attack on RandomForest ===')
    acc_clean, acc_adv = hopskip_attack_and_eval(rf_model, X_sub, y_sub,
        clip=(0.0, 1.0), max_iter=20)
    print(f"RandomForest - clean acc: {acc_clean:.3f}, adv acc: {acc_adv:.3f
        }")

    print('\n=== Surrogate (FGSM) transfer attack ===')
    transfer_results = surrogate_fgsm_transfer(rf_model, knn_model, dt_model
        , X_train, y_train, X_sub, y_sub, clip=(0.0, 1.0), eps=0.05)
    for model_name, (acc_clean, acc_adv) in transfer_results.items():
        print(f"{model_name} - clean acc: {acc_clean:.3f}, adv acc (
            transferred): {acc_adv:.3f}")


if __name__ == '__main__':
    main()
```

### 3.2 Pre-requis Techniques

```
pip install adversarial-robustness-toolbox tensorflow scikit-learn numpy
```

### 3.3 Structure du Projet

scripts/

```
|-- attacks/
|   |-- demo_attack.py          # Script principal
|-- models/
    |-- random_forest.py        # Modeles existants
    |-- knn.py
    |-- decision_tree.py
```

### 3.4 Execution

```
python scripts/attacks/demo_attack.py
```

## 4 Metriques d'Evaluation

### 4.1 Calcul de robustesse

```
Success Rate = (Nb attaques reussies / Nb total) * 100
Accuracy Drop = Accuracy_avant - Accuracy_apres
```

### 4.2 Template de rapport

**Test [Modele] - [Methode]**
- Modele : Random Forest
- Methode : HopSkipJump
- Accuracy avant : 95% -> apres : 72%
- Taux de succes : 23%
- Features sensibles : [Src_bytes, Duration]
- Temps execution : 15 min

## Conclusion

Ces deux methodes (HopSkipJump et Surrogate+FGSM) sont les standards pour tester la robustesse de vos modeles scikit-learn. Commencer par HopSkipJump pour une evaluation rapide, puis utiliser la methode surrogate pour des analyses plus poussees.