Fiche TP N°5

Méthodes de l'IA pour les NIDS

Ce TP est pour construire un modèle prédictif pour classifier les attaques réseau avec le module **Scikitlearn** de Python. Pour ce faire, nous utilisant le jeu de données pour la détection d'intrusion, NSL KDD, disponible sur NSL-KDD

Vous allez implémenter plusieurs classifieurs avec l'utilisation de la bibliothèque sklearn.

1. Importez les bibliothèques requises

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from collections import defaultdict
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import (confusion_matrix, zero_one_loss, accuracy_score,
                             f1_score, precision_score, recall_score)
```

2. Lire les valeurs des caractéristiques et de classe de l'ensemble de données

```
train_file = pd.read_csv(r"C:\Users\LENOVO\datasets\nsl-kdd\KDDTrain+.txt")
test_file = pd.read_csv(r"C:\Users\LENOVO\datasets\nsl-kdd\KDDTest+.txt")
```

3. Génération et analyse des données d'apprentissages et de test

   - Attribuer des noms aux différentes caractéristiques

```
header_names = ['duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes', 'land', 'wrong_fragment',
                'urgent', 'hot', 'num_failed_logins', 'logged_in', 'num_compromised', 'root_shell', 'su_attempted',
                'num_root', 'num_file_creations', 'num_shells', 'num_access_files', 'num_outbound_cmds', 'is_host_login',
                'is_guest_login', 'count', 'srv_count', 'serror_rate', 'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate',
                'same_srv_rate', 'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count', 'dst_host_srv_count',
                'dst_host_same_srv_rate', 'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
                'dst_host_srv_diff_host_rate', 'dst_host_serror_rate', 'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
                'dst_host_srv_rerror_rate', 'attack_type', 'success_pred']
```

```
train_df = pd.read_csv(r"C:\Users\LENOVO\datasets\nsl-kdd\KDDTrain+.txt", names=header_names)
test_df = pd.read_csv(r"C:\Users\LENOVO\datasets\nsl-kdd\KDDTest+.txt", names=header_names)
```

   - Effectuer les mappages entre étiquettes d'attaque et catégories d'attaque

```
category = defaultdict(list)
category['benign'].append('normal')

with open('datasets/training_attack_types.txt', 'r') as f:
    for line in f.readlines():
        attack, cat = line.strip().split(' ')
        category[cat].append(attack)

attack_mapping = dict((v,k) for k in category for v in category[k])
```

```
train_df['attack_category'] = train_df['attack_type'] \
                                .map(lambda x: attack_mapping[x])
test_df['attack_category'] = test_df['attack_type'] \
                                .map(lambda x: attack_mapping[x])
```

```
train_attack_types = train_df['attack_type'].value_counts()
train_attack_cats = train_df['attack_category'].value_counts()
```

```
test_attack_types = test_df['attack_type'].value_counts()
test_attack_cats = test_df['attack_category'].value_counts()
```

4. Préparation de données

- Diviser les dataset en séparant la classe est les caractéristiques

```
train_Y = train_df['attack_category']
train_x= train_df.drop(['attack_category','attack_type'], axis=1)

test_Y = test_df['attack_category']
test_x= test_df.drop(['attack_category','attack_type'], axis=1)
```

- Observer maintenant les distributions attack_type et attack_category

```
train_attack_types.plot(kind='barh', figsize=(20,10), fontsize=20)
```

```
test_attack_types.plot(kind='barh', figsize=(20,10), fontsize=15)
```

```
train_attack_cats.plot(kind='barh', figsize=(20,10), fontsize=30)
```

```
test_attack_cats.plot(kind='barh', figsize=(20,10), fontsize=30)
```

- Comme il existe des caractéristiques de type chaîne, elles sont transférées au type flottant
- Les données d'entrainement :

```
for c in range(0,40):
    train_x.iloc[:,c]=LabelEncoder().fit_transform(train_x.iloc[:,c])
train_x=train_x.astype(np.float)
```

- Les données de test :

```
for c in range(0,40):
    test_x.iloc[:,c]=LabelEncoder().fit_transform(test_x.iloc[:,c])
test_x=test_x.astype(np.float)
```

- Effectuer une sélection des caractéristiques

```
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
model.fit(train_x,train_Y)
print(model.feature_importances_)
feat_importances = pd.Series(model.feature_importances_, index=train_x.columns)
feat_importances.nlargest(12).plot(kind='barh')
plt.show()
```

5. Une fois l'étape de prétraitement est effectuée, vous pouvez implémenter les techniques de ML suivantes :

   – L'arbre de décision,
   – La machine à vecteur de support,
   – KNN,
   – k-means,
   – Méthode ensemble,
   – Le Perceptron.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, zero_one_loss

classifier = DecisionTreeClassifier(random_state=17)
classifier.fit(train_x, train_Y)

pred_y = classifier.predict(test_x)
```

6. Afficher les métriques de performances telles que l'exactitude, le rappel, la précision et le score f1.

```
from sklearn.metrics import confusion_matrix, zero_one_loss
from sklearn.metrics import average_precision_score
from sklearn.metrics import classification_report
results = confusion_matrix(test_Y, pred_y)
error = zero_one_loss(test_Y, pred_y)
accuracy=accuracy_score(test_Y,pred_y)
target_names = ['class 0', 'class 1', 'class 2', 'class 3','class 4']
print(results)
print(error)
print(accuracy)
print(classification_report(test_Y, pred_y, target_names=target_names))
```