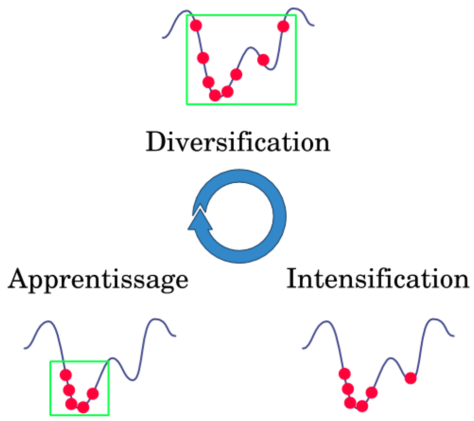
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Ecole Supérieure en Informatique 8 Mai 1945, Sidi Bel-Abbés
3ème Année Second Cycle
Système d'information et web

Rapport

Systèmes multi-agents :Métaheuristiques à base de jeu

Etudiante :
SARA TOUAHRI

Enseignant :
NADIR MAHAMMED



2024 - 2025

Table des matières

Table des figures

1 Introduction aux SMAs et les Métaheuristiques à base de jeu

— Les systèmes multi-agents (SMA), composés d'agents autonomes interagissant pour résoudre des problèmes complexes, servent de base à de nombreuses **métaheuristiques à base de jeu.** Ces méthodes, inspirées des dynamiques de compétition et de coopération entre agents, visent à explorer efficacement l'espace de recherche pour trouver des solutions optimales. En intégrant des concepts de diversification, intensification et apprentissage, elles permettent de résoudre des problèmes difficiles tout en s'adaptant aux environnements dynamiques et complexes

— J'ai travaillé sur un dataset contenant des données de profils Twitter (dataset2)(e.g., nombre de followers, amis, publications, et autres caractéristiques). Après avoir nettoyé les données, traité les valeurs manquantes, encodé les variables catégorielles et normalisé les données numériques, j'ai divisé le dataset en ensembles d'entraînement, de validation et de test. Ensuite, j'ai utilisé une métaheuristique pour optimiser les hyperparamètres d'un modèle RandomForestClassifier.

— Enfin, j'ai évalué les performances du modèle à l'aide de métriques comme l'accuracy, la précision, le rappel, le F1-score et le ROC-AUC. Ce TP m'a permis de comprendre l'application pratique des métaheuristiques à base de jeu sur un problème complexe et pertinent.

2 Présentation des outils

2.1 Python

Python est l'un des langages les plus utilisés en Machine Learning, reconnu pour sa simplicité, sa vaste communauté et son écosystème riche en bibliothèques. Pour ce projet, les outils suivants ont été essentiels :

— **NumPy** : Fournit des outils avancés pour la manipulation de tableaux multidimensionnels et le calcul numérique.

— **Pandas** : Facilite la manipulation et l'analyse des données structurées, notamment les fichiers CSV.

— **Matplotlib** : Utilisé pour visualiser les données et les résultats d'analyse.

— **Scikit-learn** : Une bibliothèque incontournable pour le Machine Learning, offrant des algorithmes de classification, régression, et clustering, ainsi que des outils pour le prétraitement des données et l'évaluation des modèles.

2.2 Jupyter Notebook

L'environnement **Jupyter Notebook** a été utilisé pour le développement interactif de ce projet. Il permet d'écrire, d'exécuter et de documenter le code dans un format pratique, tout en facilitant la visualisation des résultats et l'expérimentation.

2.3 Métaheuristique à base de jeu : Mixed Leader-Based Optimizer (MLBO)

Le **MLBO** est une métaheuristique basée sur les jeux, conçue pour résoudre des problèmes d'optimisation efficacement.

2.3.1 Principe de base

— **Objectif** : Trouver une solution quasi-optimale en combinant :
  — Le meilleur membre de la population actuelle.
  — Un membre aléatoire pour maintenir la diversité.
  Cette combinaison forme un "leader mixte" pour équilibrer exploration et exploitation dans l'espace de recherche.

2.3.2 Étapes principales

1. **Initialisation** : Génération d'une population initiale aléatoire et évaluation de ses membres.
2. **Création du leader mixte** : Association du meilleur membre avec un membre aléatoire.
3. **Mise à jour de la population** : Ajustement des membres en fonction du leader mixte.
4. **Répétitions** : Le processus est répété jusqu'à atteindre une solution satisfaisante ou une condition d'arrêt.

2.3.3 Avantages

— **Simplicité** : Pas besoin de paramètres de contrôle complexes.
— **Équilibre** : Combine exploration et exploitation pour une meilleure recherche de solutions.
— **Performance** : Compétitif avec d'autres algorithmes d'optimisation comme PSO ou GA.

2.3.4 Application dans ce projet

Le **MLBO** a été utilisé pour optimiser les hyperparamètres du modèle de classification, en exploitant les données issues des réseaux sociaux pour détecter les faux profils.

3 Prétraitement des données et entraînement du modèle

Explication du Code étape par étape

Voici une explication détaillée du code utilisé pour l'optimisation d'un modèle de classification des profils Twitter. Le but est de détecter des profils fictifs en utilisant un modèle de forêt aléatoire et une optimisation des hyperparamètres.

3.1 Chargement et nettoyage des données

Dans cette étape, le code charge le fichier Excel contenant les données Twitter et nettoie les noms des colonnes pour éliminer les espaces et les guillemets. Ensuite, il supprime les colonnes inutiles telles que `id` et `description`.

```
# Charger les donn es depuis un fichier Excel
file_path = r'C:\Users\pc\Downloads\fake␣profile␣detection␣datasets
    \csv_result-Twitter_dataset.xlsx'
data = pd.ExcelFile(file_path).parse('csv_result-Twitter_dataset')
print(data.columns)

# Nettoyage des noms de colonnes
```

```
data.columns = data.columns.str.replace("'", '', regex=True).str.strip()
print(data.columns)

# Suppression des colonnes inutiles
columns_to_drop = ['id', 'description']
columns_to_drop = [col for col in columns_to_drop if col in data.columns]
data_cleaned = data.drop(columns=columns_to_drop)
```

3.2 Traitement des valeurs manquantes

Les valeurs manquantes dans les colonnes numériques sont remplacées par zéro, et celles des colonnes catégorielles sont remplacées par `'Unknown'`.

```
# Remplir les valeurs manquantes dans les colonnes num riques
data_cleaned['followers_count'] = data_cleaned['followers_count'].fillna(0)
data_cleaned['friends_count'] = data_cleaned['friends_count'].fillna(0)
data_cleaned['favourites_count'] = data_cleaned['favourites_count'].fillna(0)

# Remplir les valeurs manquantes pour les colonnes cat gorielles
categorical_columns = ['protected', 'profile_use_background_image', 'verified', 'contributors_enabled',
                       'default_profile', 'default_profile_image', 'is_translator', 'is_fake']
for col in categorical_columns:
    data_cleaned[col] = data_cleaned[col].fillna('Unknown')
```

3.3 Encodage des variables catégorielles et normalisation des données

Les variables catégorielles sont encodées sous forme de variables binaires à l'aide de `pd.get_dummies`, et les variables numériques sont standardisées avec `StandardScaler`.

```
# Encodage des variables cat gorielles
data_processed = pd.get_dummies(data_cleaned, columns=categorical_columns, drop_first=True)

# Standardisation des colonnes num riques
scaler = StandardScaler()
numeric_columns = ['followers_count', 'friends_count', 'favourites_count', 'hashtags_average',
                   'mentions_average', 'urls_average', 'statuses_count', 'listed_count']
data_processed[numeric_columns] = scaler.fit_transform(data_processed[numeric_columns])
```

3.4 Séparation des données en ensembles d'entraînement, validation et test

Les données sont séparées en variables indépendantes (`X`) et cible (`y`). Ensuite, elles sont divisées en ensembles d'entraînement, de validation et de test à l'aide de `train_test_split`.

```
# S parer les variables ind pendantes (X) et la cible (y)
X = data_processed.drop(columns=['is_fake_True'])
y = data_processed['is_fake_True']

# Diviser les donn es en ensembles d'entra nement , de validation et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.3, random_state=42)
```

3.5 Optimisation des hyperparamètres avec une méthode métaheuristique

L'optimisation des hyperparamètres du modèle `RandomForestClassifier` est réalisée via une approche métaheuristique. La fonction `optimize` génère une population initiale et optimise les paramètres du modèle en utilisant l'algorithme.

```
class RandomForestOptimizer:
    def __init__(self, param_ranges):
        self.param_ranges = param_ranges

    def params_from_solution(self, solution: np.ndarray):
        params = {}
        for i, (param_name, (min_val, max_val)) in enumerate(self.param_ranges.items()):
            params[param_name] = int(solution[i] * (max_val - min_val) + min_val)
        return params

    def objective_function(self, solution: np.ndarray, X_train, y_train, X_val, y_val):
        model = RandomForestClassifier(**self.params_from_solution(solution))
        model.fit(X_train, y_train)
        predictions = model.predict(X_val)
        return 1 - accuracy_score(y_val, predictions)
```

3.6 Entraînement et évaluation du modèle

Une fois l'optimisation terminée, le meilleur modèle est entraîné sur les données d'entraînement et évalué sur l'ensemble de validation et de test. Les performances sont mesurées à l'aide de diverses métriques (accuracy, précision, rappel, etc.).

```
# Initialiser l'optimiseur
optimizer = RandomForestOptimizer(param_ranges)

# G n rer une population initiale pour l'optimisation
population_size = 40
population = np.random.random((population_size, len(param_ranges)))

# Lancer l'optimisation
final_population, best_model = optimizer.optimize(
    population=population,
    X_train=X_train,
    y_train=y_train,
    X_val=X_val,
    y_val=y_val,
    iterations=10
)

#  valuation finale sur l'ensemble de test
evaluate_model(best_model, X_test, y_test)
```

3.7 Visualisation de l'importance des caractéristiques

L'importance des variables est ensuite visualisée pour identifier les caractéristiques les plus influentes pour la prédiction des profils fictifs.

```
# Visualisation de l'importance des fonctionnalit s
feature_importance = pd.DataFrame({
    'feature': X.columns,
    'importance': best_model.feature_importances_
})

plt.figure(figsize=(10, 6))
feature_importance.sort_values('importance', ascending=True).plot(x
   ='feature', y='importance', kind='barh')
plt.title('Feature␣Importance')
plt.xlabel('Importance')
plt.tight_layout()
plt.show()
```

4 Résultats

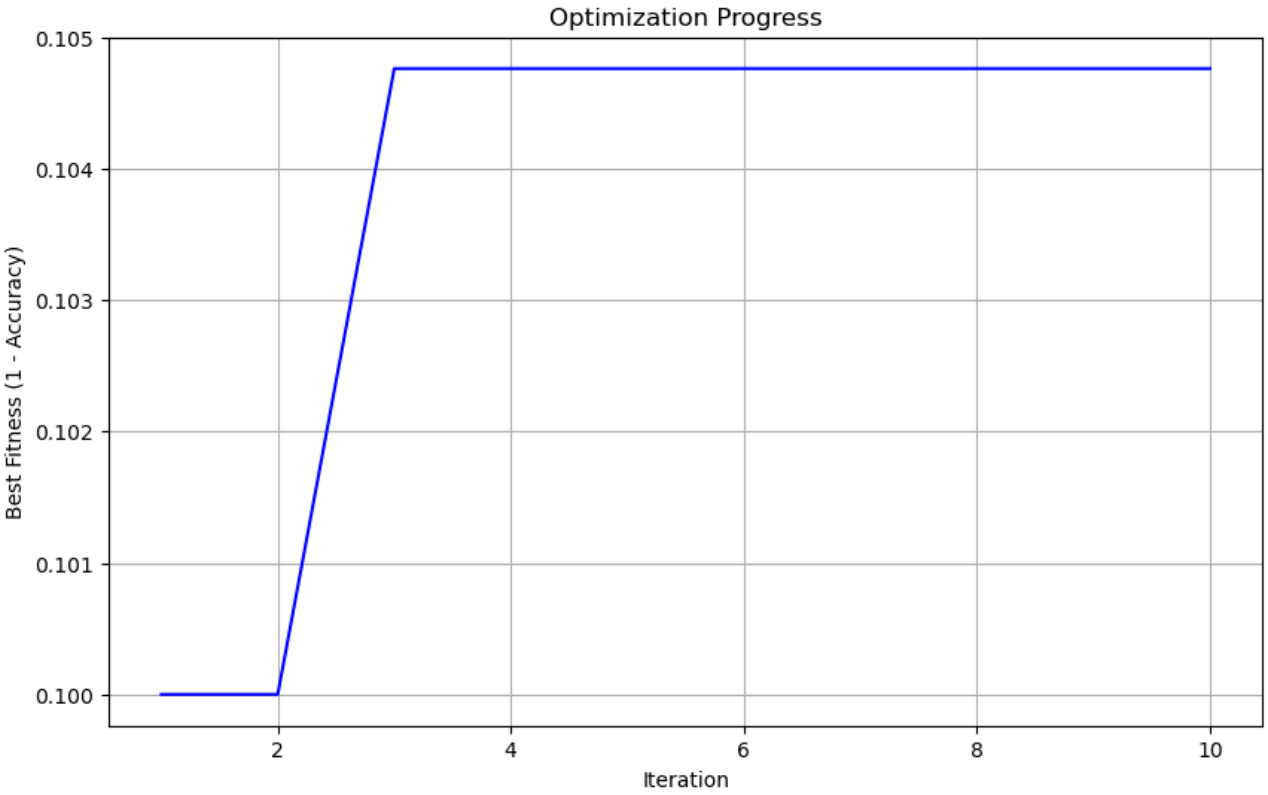


FIGURE 1 – Optimization progress

```
Optimization Complete.
Best Parameters: {'n_estimators': 113, 'max_depth': 8, 'min_samples_split': 6, 'min_samples_leaf': 3}

Evaluation Metrics:
Accuracy: 0.8857
Precision: 0.8785
Recall: 0.8952
F1-score: 0.8868
ROC-AUC: 0.9617

Final Evaluation on Test Set:
Accuracy: 0.9133
Precision: 0.9156
Recall: 0.9156
F1-score: 0.9156
ROC-AUC: 0.9706
<Figure size 1000x600 with 0 Axes>
```

FIGURE 2 – Evaluation Metrics

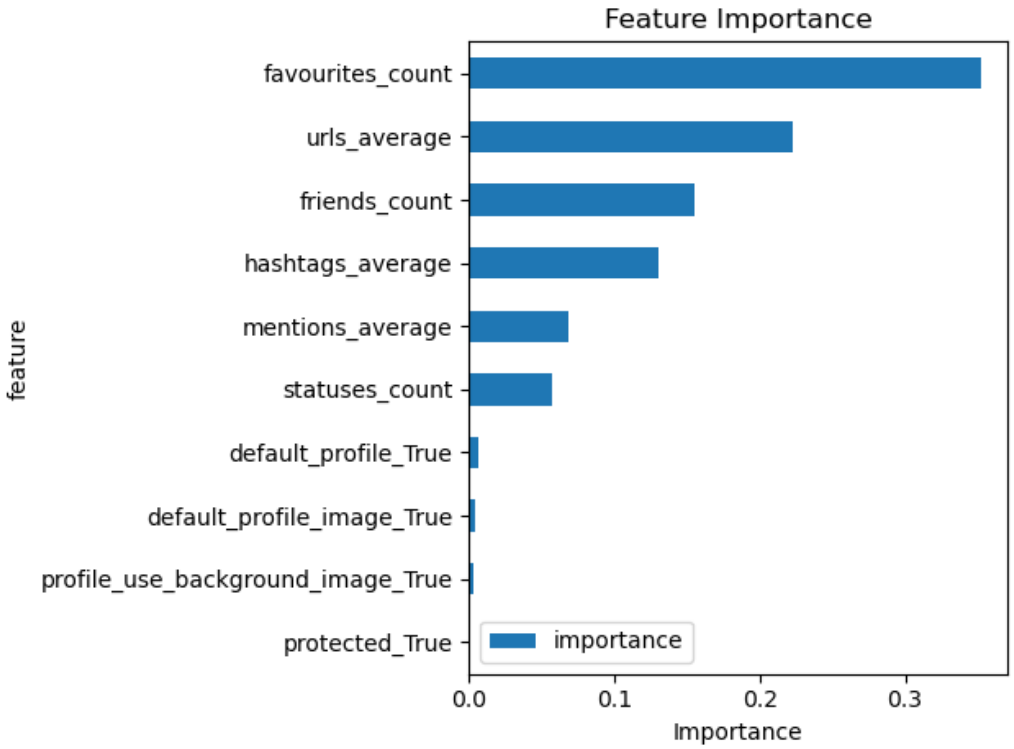


FIGURE 3 – feature importance

5 Sélection des caractéristiques avec `SelectKBest`

Dans cette étape du processus de préparation des données, nous avons utilisé la méthode `SelectKBest` de la bibliothèque `scikit-learn` pour sélectionner les meilleures caractéristiques, c'est-à-dire celles qui sont les plus pertinentes pour prédire la variable cible. La sélection des caractéristiques permet de réduire la dimensionnalité du jeu de données, ce qui peut améliorer la performance du modèle tout en réduisant le risque de surajustement (overfitting).

5.1 Description du processus de sélection

La sélection des caractéristiques a été réalisée en utilisant la fonction de score `F de Fisher` (`f_classif`), qui est bien adaptée pour évaluer la pertinence des caractéristiques numériques en fonction de leur capacité à séparer les différentes classes de la variable cible. La statistique de Fisher mesure l'écart entre les moyennes des différentes classes pour une caractéristique donnée et la variabilité à l'intérieur des classes. Une caractéristique avec un score de Fisher élevé est donc considérée comme plus informative pour prédire la classe cible.

5.2 Paramètres utilisés

— `score_func=f_classif` : Cette fonction de score calcule la statistique F de Fisher pour chaque caractéristique. Plus cette statistique est élevée, plus la caractéristique est pertinente pour la prédiction de la cible.
— `k=min(10, X.shape[1])` : Ce paramètre détermine le nombre de caractéristiques à sélectionner. Dans ce cas, nous avons choisi de sélectionner jusqu'à 10 caractéristiques, mais si le jeu de données contient moins de 10 caractéristiques, toutes sont retenues.

5.3 Résultat de la sélection

Une fois la fonction appliquée, les `k` meilleures caractéristiques ont été sélectionnées et extraites du jeu de données. Ces caractéristiques sont jugées les plus influentes pour la prédiction de la variable cible.

Dans notre cas, les caractéristiques sélectionnées étaient les suivantes :

- `friends_count`
- `favourites_count`
- `hashtags_average`
- `mentions_average`
- `urls_average`
- `statuses_count`
- `protected_True`
- `profile_use_background_image_True`
- `default_profile_True`
- `default_profile_image_True`

Ces variables ont été retenues pour l'entraînement du modèle d'apprentissage automatique, car elles ont montré les scores les plus élevés en termes de leur capacité à prédire la variable cible, à savoir si le profil est authentique ou non.

5.4 Avantages de cette méthode

- **Réduction de la dimensionnalité** : En ne gardant que les caractéristiques les plus pertinentes, nous réduisons la complexité du modèle, ce qui peut mener à un entraînement plus rapide et à une meilleure interprétabilité.
- **Amélioration de la performance du modèle** : La sélection de caractéristiques pertinentes réduit le bruit dans les données, ce qui peut améliorer la précision du modèle en évitant que des caractéristiques non pertinentes ne viennent perturber l'apprentissage.