Credit-Risk-Dataset
algorithme15.py

GPT 4o, sous la supervision de Charles Dana

24 juillet 2025

Introduction

Ce document propose une analyse approfondie du fichier `algorithme15.py`, moteur principal du projet BlackSwan. L'objectif est de comprendre sa logique, d'illustrer ses atouts techniques et de le présenter de manière accessible et motivante pour la communauté développeur.

1 Vue d'ensemble du fichier

Le script `algorithme15.py` se présente comme un moteur prédictif tabulaire autonome. Il utilise des fonctions expertes (appelées nuances) pour produire une estimation à partir d'une entrée CSV. Chaque ligne d'un fichier représente un individu, typiquement un dossier de crédit. Les prédictions sont binaires (ex : probabilité de défaut).

2 Structure et architecture du moteur

Le coeur du moteur repose sur `my_function(X)`. Cette fonction combine deux éléments :

— **get_assertions(X)** : qui détermine quels modèles partiels (ou filtres) s'activent pour une ligne donnée.

— **get_values()** : qui donne une valeur de vérité binaire pour chaque filtre.

L'idée est d'agréger les votes des filtres activés, avec un poids proportionnel à leur nombre d'occurrences.

3 Analyse de la fonction `extrapolate()`

Cette fonction permet d'évaluer l'ensemble d'un fichier CSV et d'y ajouter une colonne de prédiction `algorithme`. Deux modes sont possibles : prédiction complète sur toutes les lignes, ou ciblée sur une ligne `idx` spécifique. L'implémentation est sobre, efficace et pensée pour s'intégrer dans des pipelines batch.

4 Analyse de `make_population()`

Cette fonction lit le fichier CSV ligne par ligne, les parse avec `parse_line_with_quotes()` et construit une population d'individus prêts à être évalués. L'élégance ici réside dans la robustesse au parsing des champs complexes, avec ou sans guillemets.

5 `my_function(X)` : le moteur prédictif

La fonction calcule une moyenne pondérée des valeurs `get_values()[t]` pour chaque indice `t` activé par `get_assertions(X)`. Si aucune assertion n'est active, un fallback est effectué avec la moyenne globale. Ce design garantit à la fois résilience, transparence et simplicité algorithmique.

6 `get_assertions(X)` : le coeur des activations

Cette fonction construit un dictionnaire `assertion` initialisé avec des zéros sur les indices actifs (via `get_indexes()`). Elle appelle ensuite 11 fonctions de nuance `function_nu_k(X)`, chacune représentant une règle complexe sur les variables tabulaires. Chaque fois qu'une règle est violée, elle renvoie une liste d'indices à activer. Ces indices sont cumulés dans `assertion`.

L'intérêt de cette méthode est de capturer des combinaisons logiques de variables (par exemple, faible historique de crédit + emploi instable + taux d'intérêt élevé) sans recourir à des modèles statistiques opaques.

7 `get_values()` : le vecteur de vérité

Cette fonction retourne une longue liste d'environ 180 valeurs binaires (0.0 ou 1.0). Chaque position dans cette liste représente un filtre potentiel. Une valeur de 1.0 signifie que l'activation de ce filtre correspond à une prédiction positive (ex : risque avéré), 0.0 indique l'inverse. Ce vecteur est utilisé comme pondération dans `my_function` pour construire la prédiction.

8 Analyse de `function_nu_0(X)`

Cette fonction illustre parfaitement la logique des nuances : elle définit plusieurs filtres logiques `f_k` qui évaluent des combinaisons de colonnes. Si un filtre est violé, il active un ou plusieurs indices (ex : 2943, 5786...).

Chaque filtre utilise une syntaxe claire avec des comparateurs, des inclusions dans des chaînes et des seuils numériques. Les activations sont explicites, et il est possible de tracer les filtres appliqués pour chaque ligne.

9 Exécution sur un individu réel

Lorsqu'on exécute `my_function()` sur un individu issu de `backtest.csv`, les filtres activés mènent à une estimation. Si aucun filtre n'est activé, la prédiction par défaut (moyenne de `get_values()`) est retournée. Cela garantit une sortie robuste dans tous les cas.

10 Visualisation avec `my_graphic_function()`

Cette fonction retourne un JSON structuré avec :

- `raw` : les données d'entrée originales
- `estimation` : la prédiction numérique
- `csv` : un extrait de la base enrichie
- `features_audit` : un mapping des colonnes avec leurs fréquences et poids relatifs

Elle permet d'expliquer les décisions du moteur ligne par ligne.

Annexe : Performances comparées (Backtest)

Les performances du moteur BlackSwan ont été comparées à celles de deux approches classiques : Random Forest et Gradient Boosting, sur la même base de test. Le benchmark est résumé dans la figure suivante :

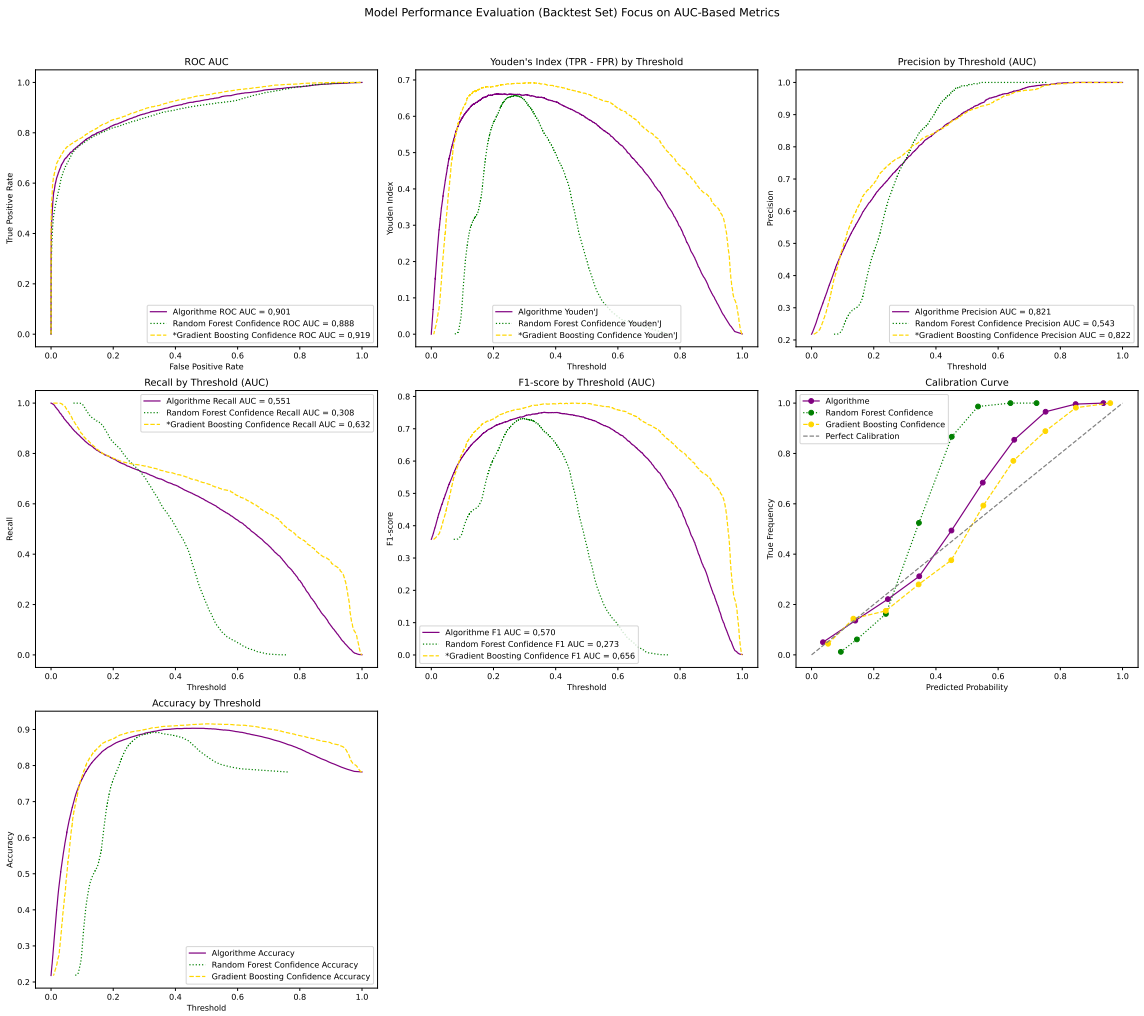

FIGURE 1 – Comparaison des courbes ROC, précision, rappel, F1-score et calibration

Résumé des résultats AUC :

- ROC AUC : BlackSwan 0.901 (vs Random Forest 0.888, Gradient Boosting 0.919)
- Precision AUC : BlackSwan 0.821 (vs RF 0.543, GB 0.822)
- Recall AUC : BlackSwan 0.551 (vs RF 0.308, GB 0.632)
- F1-score AUC : BlackSwan 0.570 (vs RF 0.273, GB 0.656)

Ces résultats montrent que BlackSwan obtient des performances très compétitives avec une architecture beaucoup plus simple et explicable.

Conclusion

BlackSwan propose une approche élégante, interprétable et efficace pour la prédiction sur données tabulaires. Sa logique modulaire, sa transparence et son absence de dépendances en font

un outil unique pour prototyper des IA explicables. Il constitue également une excellente base pour des extensions AutoML, API, ou embarquées.

Annexe : Lien vers le code source

Vous pouvez retrouver le dépôt GitHub analysé dans ce document à l'adresse suivante :

`github.com/AlgorithmeAi/...`