Rapport Technique :
Système de Prévision de Température Basé sur Weka et Java pour l'Analyse de Séries Temporelles

EL Majdi Walid

Encadé par Prof. HAJJI TARIK

19 mai 2025

Résumé

Résumé : Ce rapport détaille la conception, l'implémentation et l'évaluation d'un système de prévision des températures journalières minimales et maximales, en mettant l'accent sur l'utilisation de la bibliothèque Weka au sein d'un environnement Java pour l'analyse de séries temporelles. Le système entraîne et compare trois algorithmes d'apprentissage automatique – Régression Linéaire, Forêt Aléatoire et Régression par Vecteurs de Support (SVR) – pour prédire les températures. Le rapport aborde l'architecture du système, le traitement des données, les spécificités de l'utilisation de Weka pour les séries temporelles, y compris ses limitations qui ont guidé le choix d'un jeu de données de complexité modérée (températures historiques). Il couvre également la sélection du modèle, une brève description de l'interface utilisateur graphique (GUI) développée avec Swing, les défis techniques rencontrés et les pistes d'améliorations futures. Les contraintes inhérentes à Weka et à l'écosystème Java pour le ML, telles que l'absence de support natif pour certaines architectures neuronales avancées et le manque d'infrastructures cloud gratuites avec accélération GPU (contrairement à l'écosystème Python), sont discutées comme facteurs justifiant la portée et la méthodologie du projet.

Mots-clés : Prévision de température, Séries temporelles, Apprentissage automatique, Weka, Java, Régression Linéaire, Forêt Aléatoire, SVR, RMSE, MAE, Limitations de Weka.

Table des matières

1 Introduction

L'apprentissage automatique (Machine Learning, ML) offre des outils puissants pour l'analyse de données et la création de modèles prédictifs dans divers domaines. Ce rapport se concentre sur l'application de ces techniques, spécifiquement la bibliothèque Weka en conjonction avec Java, au problème de la prévision de température, un cas d'étude classique de l'analyse de séries temporelles. Bien que l'exercice initial ait pu suggérer d'autres applications comme la détection de fraudes, l'objectif ici est d'explorer les capacités et les contraintes de Weka/Java pour la modélisation temporelle.

La prévision météorologique, et plus particulièrement la prévision de température, nécessite l'analyse de données historiques pour identifier des tendances et des motifs permettant d'anticiper les valeurs futures. Ce projet vise à construire un système capable de prédire les températures minimales et maximales quotidiennes. Les technologies clés employées sont le langage de programmation Java, la bibliothèque d'apprentissage automatique Weka, et la bibliothèque Swing pour une interface utilisateur graphique rudimentaire.

Ce rapport présente en détail la méthodologie adoptée, l'architecture du système, les modèles de ML implémentés, leur évaluation, une discussion approfondie sur les limitations de Weka et comment elles ont influencé le choix des données et des modèles, les défis techniques rencontrés, et les perspectives d'évolution du projet. Le code source du projet est disponible sur GitHub : https://github.com/elm19/java_timeseries_project.

2 Présentation du Projet

2.1 Objectifs

Le principal objectif de ce projet est d'explorer l'utilisation de Weka et Java pour développer un système de prévision de séries temporelles pour la prédiction des températures. Plus spécifiquement, il s'agit de :

- Implémenter et comparer plusieurs modèles d'apprentissage automatique disponibles dans Weka.
- Évaluer rigoureusement les performances des modèles en utilisant des métriques standards.
- Sélectionner le meilleur modèle basé sur un score pondéré.
- Persister les modèles entraînés pour une utilisation ultérieure.
- Développer une interface graphique utilisateur (GUI) simple pour visualiser les prédictions.

2.2 Technologies Utilisées

- **Langage de programmation :** Java (JDK 8 ou supérieur).
- **Bibliothèque d'apprentissage automatique :** Weka (Waikato Environment for Knowledge Analysis).
- **Interface Graphique Utilisateur (GUI) :** Java Swing (pour une visualisation basique).
- **Bibliothèques additionnelles :** MTJ (Matrix Toolkits for Java), Core (Utils), ARPACK (ARnoldi PACKage), Time Series Forecasting package pour Weka.

2.3 Composants Principaux

Le système est structuré autour de deux composants majeurs :

1. `TimeSeriesForecaster.java` **:** Responsable de l'entraînement des modèles.
2. `TemperatureDisplay.java` **:** Fournit une interface graphique simple pour afficher les prédictions.

2.4 Jeu de Données

Le système utilise un jeu de données historiques de températures journalières minimales et maximales collectées à partir de 1997 (`data/daily_temp.csv`). Ce choix d'un jeu de données relativement simple et de taille modérée est en partie dicté par les limitations de l'environnement Weka/Java discutées ultérieurement.

3 Implémentation Technique et Considérations sur Weka

3.1 Modèles d'Apprentissage Automatique

Trois modèles d'apprentissage automatique de Weka ont été implémentés :

1. **Régression Linéaire (Linear Regression)**
2. **Forêt Aléatoire (Random Forest)**
3. **Régression par Vecteurs de Support (Support Vector Regression - SVR)**

Des modèles distincts sont entraînés pour les températures minimales et maximales.

3.2 Approche de Séries Temporelles avec Weka

Weka, via son package `timeseriesForecasting`, gère la transformation des données de séries temporelles en un format d'apprentissage supervisé. Ce processus implique typiquement la création d'une "fenêtre glissante" (sliding window) où les valeurs passées de la série (et potentiellement d'autres variables exogènes) sont utilisées comme attributs pour prédire les valeurs futures. Par exemple, pour prédire la température au jour t, les températures des jours $t-1, t-2, \ldots, t-k$ (où k est la taille de la fenêtre, un paramètre configurable dans Weka) deviennent les variables explicatives. Weka gère cette transformation en interne lorsqu'on utilise les outils dédiés aux séries temporelles, simplifiant ainsi la tâche de l'utilisateur qui n'a pas à implémenter manuellement cette logique de fenêtrage.

```
1 TSForecaster forecaster = new TSForecaster();
2 forecaster.setFieldsToForecast("MinTemp,MaxTemp");
3 forecaster.getTSLagMaker().setTimeLag(6); // 6 est la taille de la fen tre
```

Listing 1 – Configuration du TSForecaster (exemple)

3.3 Limitations de Weka et Justification des Choix Technologiques

L'utilisation de Weka dans un environnement Java pour l'analyse de séries temporelles, bien que pratique pour accéder à une suite d'algorithmes classiques, vient avec certaines limitations qui ont influencé la conception de ce projet et le choix du jeu de données :

— **Gestion des Grands Jeux de Données :** Weka, étant principalement basé sur Java et opérant souvent en mémoire, peut rencontrer des difficultés de performance et de scalabilité avec de très grands jeux de données. Cela contraste avec des environnements Python où des bibliothèques comme Dask ou des intégrations avec Spark permettent de gérer plus facilement des données volumineuses.

— **Absence de Modèles Complexes (Deep Learning) :** Weka n'intègre pas nativement des architectures neuronales avancées spécifiquement conçues pour les séries temporelles, telles que les réseaux de neurones récurrents (RNN), les Long Short-Term Memory (LSTM), les Gated Recurrent Units (GRU) ou même des modèles statistiques plus sophistiqués comme SARIMA avec une configuration aisée pour des saisonnalités multiples. Cette absence limite la capacité à modéliser des dynamiques temporelles très complexes qui pourraient être présentes dans d'autres types de données (par exemple, financières, trafic, etc.).

— **Justification du Choix des Données :** Face à ces limitations, l'utilisation d'un jeu de données de température, qui est relativement structuré et ne présente pas nécessairement le même degré de complexité que d'autres séries temporelles, est un choix pragmatique. Il permet de se concentrer sur l'application des modèles classiques de Weka sans se heurter immédiatement à ses limites en termes de complexité de modèle ou de volume de données.

— **Interface Graphique de Weka (Explorer) :** Il est important de noter que Weka fournit sa propre interface graphique ("Explorer") qui permet d'effectuer de nombreuses tâches d'apprentissage automatique, y compris l'entraînement et la sauvegarde de modèles, sans écrire de code Java. Cette interface est très utile pour l'exploration rapide et le prototypage, et constitue une alternative plus simple à l'intégration programmatique pour de nombreux utilisateurs.

— **Contraintes de l'Écosystème Java vs. Python :** L'écosystème Python bénéficie d'un large support pour l'apprentissage automatique, y compris l'accès à des plateformes cloud gratuites (comme Google Colaboratory) offrant des ressources GPU pour accélérer l'entraînement de modèles complexes. De telles options sont moins courantes ou moins directement accessibles pour les projets purement Java/Weka, ce qui peut également contraindre l'exploration de modèles gourmands en calcul ou l'utilisation de très grands jeux de données. Les avertissements de logs concernant 'com.github.fommil.netlib' (ARPACK, BLAS, LAPACK) dans ce projet suggèrent que les implémentations natives optimisées de ces bibliothèques d'algèbre linéaire n'ont pas été chargées, ce qui pourrait affecter la vitesse d'entraînement.

Ces facteurs combinés justifient l'approche adoptée dans ce projet : utiliser un jeu de données de taille et de complexité gérables, et se concentrer sur les capacités fondamentales de Weka pour les séries temporelles.

3.4 Évaluation des Modèles et Validation Croisée

Validation croisée à 10 plis. Métriques : RMSE, MAE, Coefficient de Corrélation.

3.5 Sélection du Modèle

Score pondéré : RMSE (40%), MAE (40%), Coefficient de Corrélation (20%).

3.6 Persistance des Modèles

Utilisation de `weka.core.SerializationHelper`. Modèles stockés dans `model/min/` et `model/max/`.

4 Architecture du Système

4.1 `TimeSeriesForecaster.java`

Responsabilités :

— Chargement et Prétraitement des Données (gestion des lignes malformées, transformation en instances Weka).

— Configuration de Weka pour les Séries Temporelles.

— Entraînement des Modèles.

— Évaluation des Modèles.

— Sélection et Sauvegarde des Modèles.

4.2 `TemperatureDisplay.java`

Ce composant gère une interface utilisateur basique :

— Chargement des Modèles Persistés.

— Génération des Prédictions pour affichage.

— Affichage simple des données sur 7 jours.

4.3 Structure des Données et Stockage

— **Données d'Entrée :** Fichier CSV `data/daily_temp.csv`.

— **Stockage des Modèles :** Répertoires `model/min/` et `model/max/`.

5 Interface Utilisateur (GUI)

L'interface graphique, implémentée avec Java Swing dans `TemperatureDisplay.java`, est un composant secondaire du projet, visant principalement à offrir une visualisation sommaire des prédictions. Elle affiche les températures minimales et maximales pour une période de 7 jours (3 passés, actuel, 3 futurs), avec une mise en évidence du jour actuel. Étant donné l'accent mis sur l'utilisation de Weka et la modélisation, le développement de la GUI n'a pas été une priorité et son esthétique est fonctionnelle plutôt que raffinée.

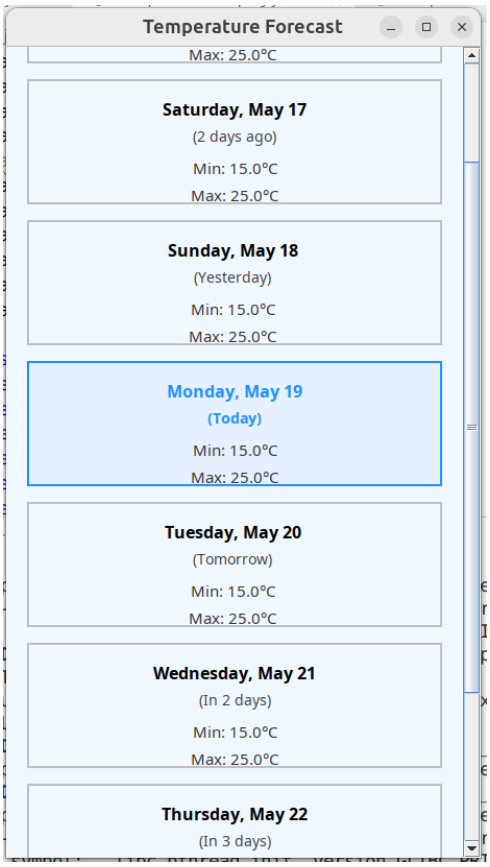
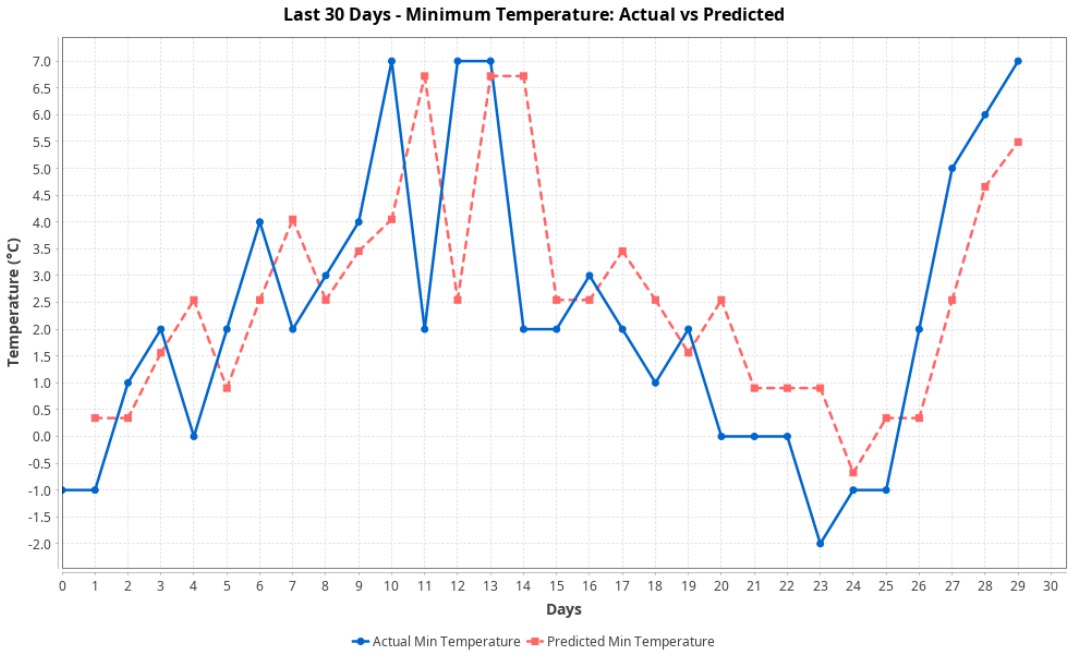


FIGURE 1 – Aperçu de l'interface graphique basique du système.

6 Processus de Développement

Le développement a suivi les étapes suivantes :



1. Préparation et Prétraitement des Données.
2. Entraînement et Évaluation des Modèles avec Weka.
3. Implémentation de la Persistance des Modèles.
4. Développement basique de la GUI.
5. Tests et Raffinements.

7 Résultats et Discussion

7.1 Performances des Modèles

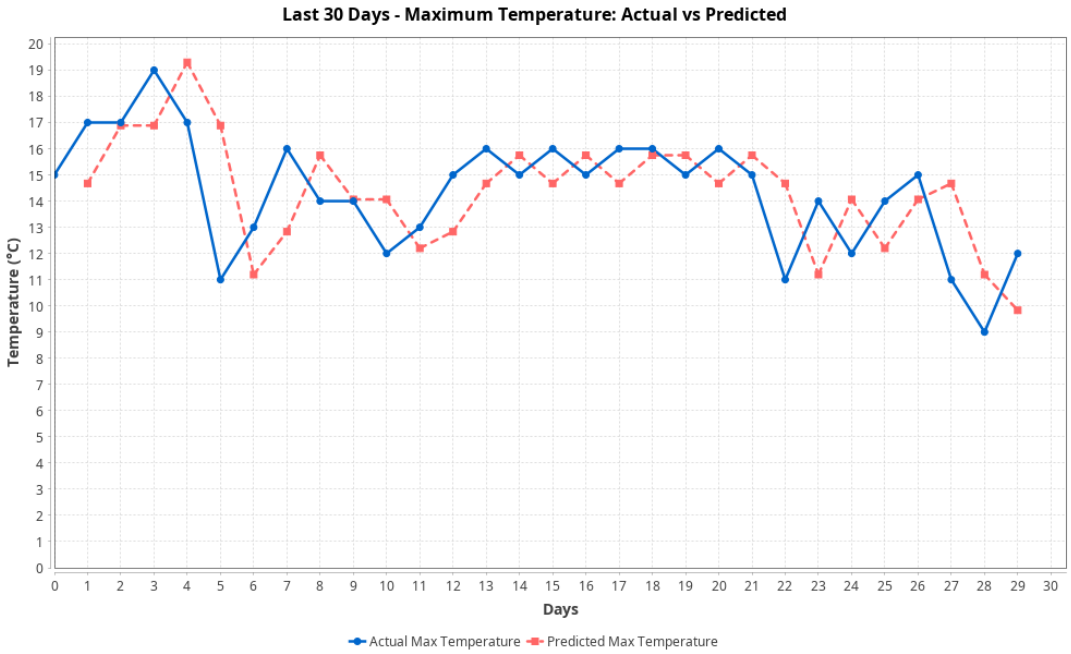
Les résultats de l'évaluation des modèles, obtenus après l'exécution de `TimeSeriesForecaster.java`, sont présentés ci-dessous.

7.1.1 Modèles pour la Température Minimale

TABLE 1 – Performances des modèles pour la prévision de la température minimale.

Modèle	RMSE	MAE	Coeff. Corrélation
Régression Linéaire	2.7269	2.1290	0.9170
Forêt Aléatoire	2.7058	2.1190	0.9184
Régression Vecteurs Support (SVR)	2.7792	2.1150	0.9170

Meilleur modèle pour la température minimale (basé sur le score pondéré) : Forêt Aléatoire.



7.1.2 Modèles pour la Température Maximale

TABLE 2 – Performances des modèles pour la prévision de la température maximale.

Modèle	RMSE	MAE	Coeff. Corrélation
Régression Linéaire	2.6533	2.0706	0.9542
Forêt Aléatoire	2.6307	2.0473	0.9550
Régression Vecteurs Support (SVR)	2.6825	2.0431	0.9542

Meilleur modèle pour la température maximale (basé sur le score pondéré) : Forêt Aléatoire.

7.2 Analyse des Résultats

La Forêt Aléatoire s'est avérée être le modèle le plus performant pour les deux types de températures, comme indiqué dans les Tableaux 1 et 2. Les erreurs (RMSE, MAE) sont de l'ordre de 2-3°C, et les coefficients de corrélation sont élevés (>0.91), indiquant une bonne adéquation des modèles aux données, dans les limites des algorithmes utilisés.

7.3 Dynamisme des Prédictions

Il est important de clarifier que le système, dans sa forme actuelle, n'implémente pas un mécanisme de mise à jour dynamique des prédictions en temps réel (par exemple, une tâche qui s'exécute chaque jour pour actualiser automatiquement la prévision affichée dans la GUI sans intervention). Le "dynamisme" réside dans la capacité du système à générer de nouvelles prédictions si le processus d'entraînement ou de prédiction est relancé après l'ajout de nouvelles données au fichier `daily_temp.csv`. Par exemple, si la température du jour est ajoutée au fichier CSV, et que `TemperatureDisplay.java` est relancé (ou si une fonction de "rafraîchissement" chargeant les dernières données et ré-appliquant le modèle était implémentée), les prédictions pour les jours futurs seraient recalculées sur la base de ces informations plus récentes. L'architecture actuelle repose donc sur une mise à jour manuelle des données suivie d'une ré-exécution pour obtenir des prévisions actualisées.

7.4 Implications des Limitations de Weka sur les Résultats

Les limitations de Weka discutées précédemment (taille des données, complexité des modèles) signifient que les résultats obtenus, bien que bons pour les modèles classiques, pourraient potentiellement être améliorés avec des outils permettant d'explorer des architectures plus complexes ou de traiter des volumes de données plus importants. Ce projet démontre ce qui est réalisable dans le cadre de Weka avec un jeu de données approprié.

8 Défis Techniques Rencontrés

— Adaptation des données chronologiques pour Weka.

— Mise en place du pipeline d'évaluation et de sélection.

— Gestion des dépendances Java et des avertissements liés aux bibliothèques natives.

9 Conclusion et Améliorations Futures

Ce projet a permis d'explorer l'utilisation de Java et Weka pour la prévision de séries temporelles de température. Le modèle de Forêt Aléatoire a été identifié comme le plus performant. Les contraintes de Weka et de l'écosystème Java pour le ML ont guidé les choix méthodologiques.

Plusieurs pistes d'améliorations futures peuvent être envisagées :

— **Automatisation de la Mise à Jour des Données :** Implémenter un mécanisme pour récupérer automatiquement les dernières données de température et relancer le processus de prédiction.

— **Exploration d'Autres Modèles Weka :** Tester d'autres algorithmes de Weka ou des configurations plus avancées du package `timeseriesForecasting`.

— **Ingénierie des Caractéristiques (Feature Engineering) :** Extraire des caractéristiques temporelles plus sophistiquées (saisonnalité, jours fériés, etc.) pour améliorer les modèles.

— **Comparaison avec d'Autres Frameworks :** À titre d'étude, comparer les performances avec des solutions basées sur Python (par exemple, scikit-learn, Statsmodels, Prophet, ou des modèles LSTM avec Keras/TensorFlow) pour évaluer l'apport potentiel de modèles plus complexes ou d'un écosystème différent.

Ce projet constitue une base solide pour de futurs développements dans le domaine de la prévision météorologique assistée par l'apprentissage automatique, en particulier pour ceux qui souhaitent opérer au sein de l'écosystème Java/Weka.