Prédiction des précipitations à partir de données météorologiques

Fait par : Zabour Sofia, Tetah Yacine, Kireche Kader, El Khalil Zenasni

28 mars 2025

Il pleut ?

Il pleut.
Il va pleuvoir.
Il pleuvra.
Il pleuvait.
Il a plu.
Il pleuvrait.
J'ai peur qu'il pleuve.

Et on voit un bel arc-en-ciel !

Table des matières

Chapitre 1

Introduction : Contexte et objectif du projet

L'objectif principal de ce projet est de développer un modèle de prédiction des précipitations en utilisant un ensemble de données provenant de différentes stations météorologiques. Ce modèle devra être capable de classer les jours en fonction de la probabilité de précipitation, permettant ainsi une prévision binaire (pluie ou pas de pluie). À cette fin, plusieurs étapes sont nécessaires, incluant la préparation des données, la gestion du déséquilibre des classes, encodage des variables catégorielles et la création du modèle prédictif.

Dans ce projet, nous allons utiliser le **Deep Learning** pour surmonter les défis liés à la gestion des données temporelles, au déséquilibre des classes , et à l'amélioration des performances de prédiction.

Le processus comprend également des étapes comme l'utilisation de techniques de rééchantillonnage hybride (SMOTE et undersampling) pour équilibrer les classes, ainsi que la standardisation des données pour garantir la convergence rapide et stable du modèle. Grâce à ces approches avancées, nous visons à obtenir un modèle performant, capable de prédire avec précision les épisodes pluvieux tout en minimisant les erreurs.

Nous vous invitons à plonger dans ce projet passionnant, où l'innovation du Deep Learning rencontre les défis réels de la prévision des précipitations. Une lecture pleine de découvertes et de solutions captivantes vous attend !

Chapitre 2

Partie 1 : Chargement et Prétraitement des données

2.1 Introduction

Cette section présente le processus de chargement, de prétraitement et d'ingénierie des caractéristiques des données collectées pour la prédiction des événements pluvieux à partir de mesures prises par différentes stations météorologiques en Côte d'Ivoire. L'objectif est de préparer un jeu de données propre, cohérent et prêt pour la modélisation.

2.2 Chargement des Données

Les données sont collectées sous forme de fichiers CSV, chacun représentant les mesures prises par une station météorologique spécifique. Chaque fichier contient des enregistrements horodatés avec des informations sur des variables telles que le signal standard (`signal_std`), le signal radiométrique (`signal_rad`), et les informations sur la pluie (`pluie`). Ces fichiers sont chargés dans un `DataFrame` à l'aide de la bibliothèque `pandas` en Python, où l'index temporel est réinitialisé et transformé en une colonne standard de type `datetime`.

Les fichiers chargés sont les suivants :

— `CIV-00001.csv` (Station1)
— `CIV-00033.csv` (Station2)
— `CIV-00004.csv` (Station3)

Le tableau résultant contient les colonnes : `temps`, `signal_std`, `signal_rad`, `pluie`, et `id_station`, et est ensuite combiné en un seul grand `DataFrame`.

2.3 Transformation et Prétraitement des Données

Pour capturer la nature cyclique du temps, l'heure de chaque enregistrement est extraite et transformée en caractéristiques sinusoïdale et cosinusoïdale. Cette transformation permet de modéliser la variation de l'heure au cours de la journée de manière adéquate, en tenant compte du fait que l'heure de 23h est proche de celle de 0h dans un contexte cyclique.

Ensuite, la colonne `heure` est remplacée par ses représentations sinusoïdales et cosinusoïdales, permettant ainsi d'enrichir les caractéristiques temporelles sans perte d'information.

2.4 Découpage des Données en Entraînement et Test

Afin d'éviter toute fuite d'information entre l'entraînement et l'évaluation du modèle, les données sont séparées en deux ensembles : un ensemble d'entraînement et un ensemble de test. Les 7 derniers jours de données pour chaque station sont réservés à l'évaluation du modèle (ensemble de test), tandis que les données précédentes sont utilisées pour l'entraînement du modèle.

Les données sont triées par `temps` et `id_station` pour garantir l'ordre chronologique et la séparation correcte entre les ensembles. La coupure de prévision est ensuite définie à midi le dernier jour du mois, afin de capturer les dernières données avant de prédire les événements de pluie à court terme.

Les proportions de points pluvieux dans les ensembles d'entraînement et de test sont vérifiées, montrant que les ensembles contiennent un pourcentage de points pluvieux relativement faible mais représentatif (1.57 % dans l'ensemble d'entraînement et 2.74 % dans l'ensemble de test).

2.5 Gestion des Valeurs Manquantes

Les colonnes `signal_std` et `signal_rad` contiennent des valeurs manquantes dans les jeux de données. Pour imputer ces valeurs, deux méthodes sont utilisées, tout d'abord on a fait la moyenne glissante. Cette méthode consiste à remplir les valeurs manquantes par la moyenne des valeurs voisines dans une fenêtre de taille 5 minutes. Toutefois, cette approche n'a pas réduit significativement le pourcentage de valeurs manquantes, ce qui la rend inadaptée à notre jeu de données. En revanche, l'interpolation linéaire s'est révélée plus efficace, réduisant considérablement le pourcentage de valeurs manquantes. Cette méthode consiste à estimer les valeurs absentes en uti-

lisant une approximation linéaire entre les points de données adjacents. Elle est plus pertinente dans notre cas car :

- Les séries temporelles présentent une continuité qui permet d'interpoler de manière plus réaliste les valeurs manquantes.
- Contrairement à la moyenne glissante, qui lisse les données et peut effacer des tendances locales, l'interpolation linéaire conserve mieux la structure des variations du signal.
- Les mesures météorologiques évoluent souvent progressivement dans le temps, ce qui justifie l'utilisation d'une approche basée sur l'interpolation.

Ainsi, après application de l'interpolation linéaire, les données sont nettoyées et prêtes pour la modélisation.

2.6 Visualisation des Données

Pour mieux comprendre la distribution des données, un histogramme et un boxplot sont utilisés pour examiner la répartition des valeurs de `signal_std` et `signal_rad` dans l'ensemble d'entraînement et de test. Ces visualisations permettent de confirmer qu'il n'y a pas de valeurs extrêmes et que la distribution des données est équilibrée. L'absence de valeurs aberrantes dans un boxplot est confirmée par la distribution symétrique et l'absence de points largement éloignés de la médiane.

2.7 Conclusion

Le prétraitement des données a permis de :

- Charger et concaténer les différentes sources de données provenant des stations météorologiques.
- Enrichir les caractéristiques temporelles avec des transformations sinusoïdales et cosinusoïdales pour capturer la cyclicité de l'heure.
- Séparer correctement les données en ensembles d'entraînement et de test, tout en garantissant qu'aucune fuite d'information ne se produit.
- Imputer efficacement les valeurs manquantes des colonnes `signal_std` et `signal_rad` à l'aide d'une interpolation linéaire.
- Vérifier la distribution des données et la présence de valeurs aberrantes grâce à des outils de visualisation.

Ces étapes préliminaires permettent d'obtenir un jeu de données prêt à être utilisé pour l'entraînement de modèles de prédiction des événements pluvieux, tout en garantissant que les données sont de bonne qualité et représentatives des phénomènes étudiés.

Chapitre 3

Partie 2 : Analyse des Données et Visualisations

3.1 Introduction

Cette section explore la distribution des caractéristiques des signaux météorologiques et leur relation avec les précipitations en Côte d'Ivoire. L'objectif est d'identifier des tendances et des relations exploitables pour la modélisation.

3.2 Description Statistique des Données

La description statistique des données de l'ensemble d'entraînement est présentée ci-dessous :

	temps	signal_std	signal_rad	pluie	heure_cos	heure_sin
count	90720	90720.000000	90720.000000	90720.000000	9.072000e+04	9.072000e+04
mean	2023-02-11 11:59:29.999999744	-20.043155	-27.986127	0.015675	-7.237009e-17	8.145552e-18
min	2023-02-01 00:00:00	-27.630000	-29.480000	0.000000	-1.000000e+00	-1.000000e+00
25%	2023-02-06 05:59:45	-21.960000	-29.040000	0.000000	-7.071068e-01	-7.071068e-01
50%	2023-02-11 11:59:30	-20.740000	-28.690000	0.000000	-6.123234e-17	6.123234e-17
75%	2023-02-16 17:59:15	-17.410000	-26.400000	0.000000	7.071068e-01	7.071068e-01
max	2023-02-21 23:59:00	-16.520000	-23.460000	1.000000	1.000000e+00	1.000000e+00
std	NaN	2.056517	1.297562	0.124214	7.071107e-01	7.071107e-01

Description des variables principales :
Les valeurs montrent que les périodes pluvieuses sont rares (1.57% dans l'ensemble d'entraînement). Cela peut compliquer la modélisation en raison du déséquilibre des classes.

3.3 Analyse Graphique

3.3.1 Scatter Plots

Les scatter plots ci-dessous montrent la relation entre les signaux standard et radiométriques pour les périodes pluvieuses et non pluvieuses.

Observations :

— Les points pluvieux et non pluvieux ne sont pas clairement séparables.
— Les signaux Rad et Std ne suffisent pas à discriminer la présence de pluie.
— Cela suggère que d'autres caractéristiques (temporelles ou dérivées) pourraient être nécessaires.

3.3.2 Time Series Analysis

Nous analysons les signaux en fonction du temps pour détecter des tendances associées aux périodes pluvieuses.

Observations :

— Pendant les périodes pluvieuses, le signal Rad augmente fortement.
— Le signal Std diminue sensiblement sous la pluie.
— La volatilité des signaux est plus marquée sous la pluie.
— Ces variations temporelles pourraient être exploitées pour la modélisation.
— Nous pouvons clairement voir que pour les périodes pluvieuses, les caractéristiques du signal sont plus volatiles et présentent des pics plus élevés/plus bas par rapport aux points non pluvieux. La caractéristique du signal radar augmente considérablement pendant les points pluvieux, tandis que la caractéristique du signal satellite diminue considérablement. Par conséquent, les caractéristiques du signal pourraient être combinées pour créer une nouvelle caractéristique qui capture le changement des caractéristiques du signal pendant les périodes pluvieuses.

3.4 Conclusion

L'analyse des données met en évidence :

— Une faible séparabilité des périodes pluvieuses et non pluvieuses dans les scatter plots.
— Une forte variation temporelle des signaux sous la pluie, ce qui pourrait être utile pour la prédiction.
— La nécessité d'introduire de nouvelles caractéristiques pour améliorer la discrimination.

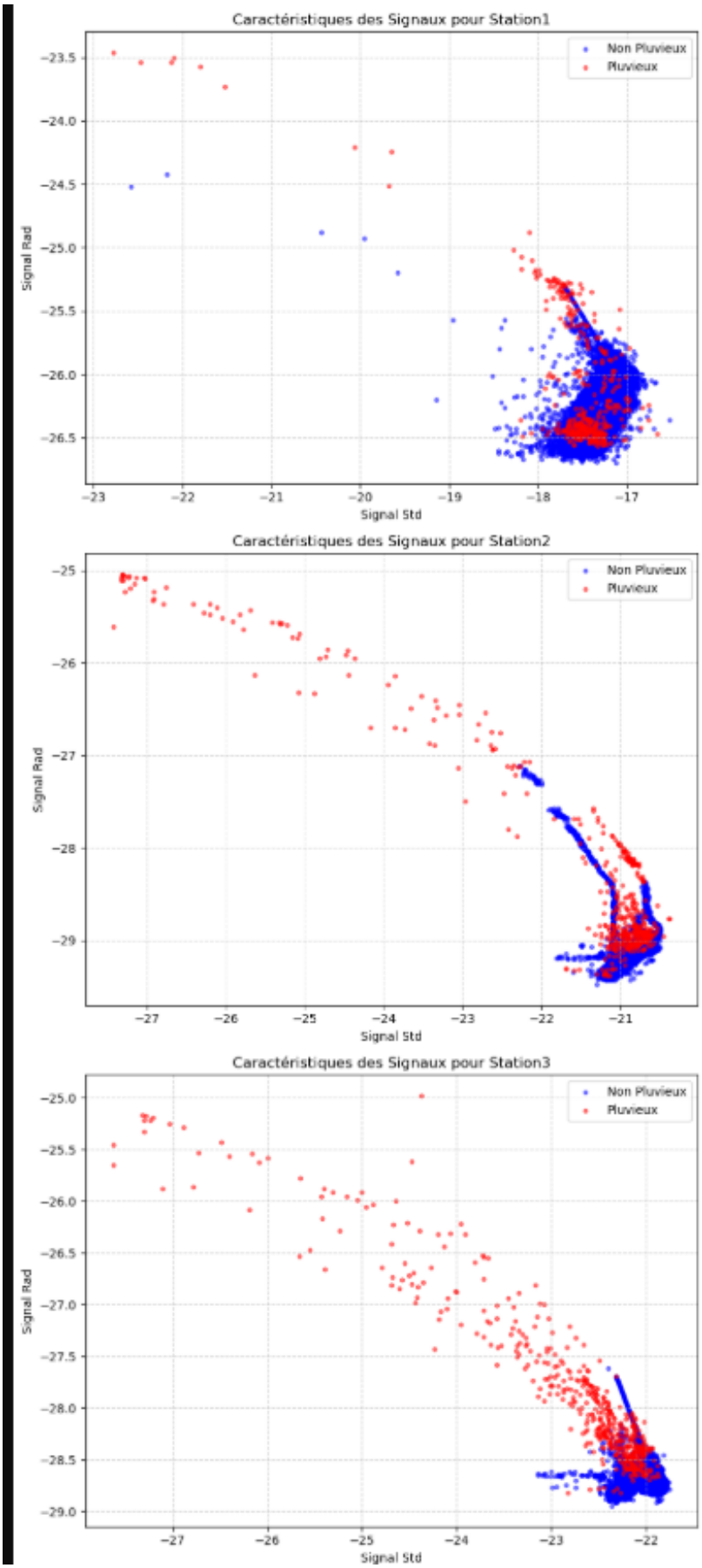

FIGURE 3.1 – Scatter Plot des caractéristiques des signaux pour différentes stations.

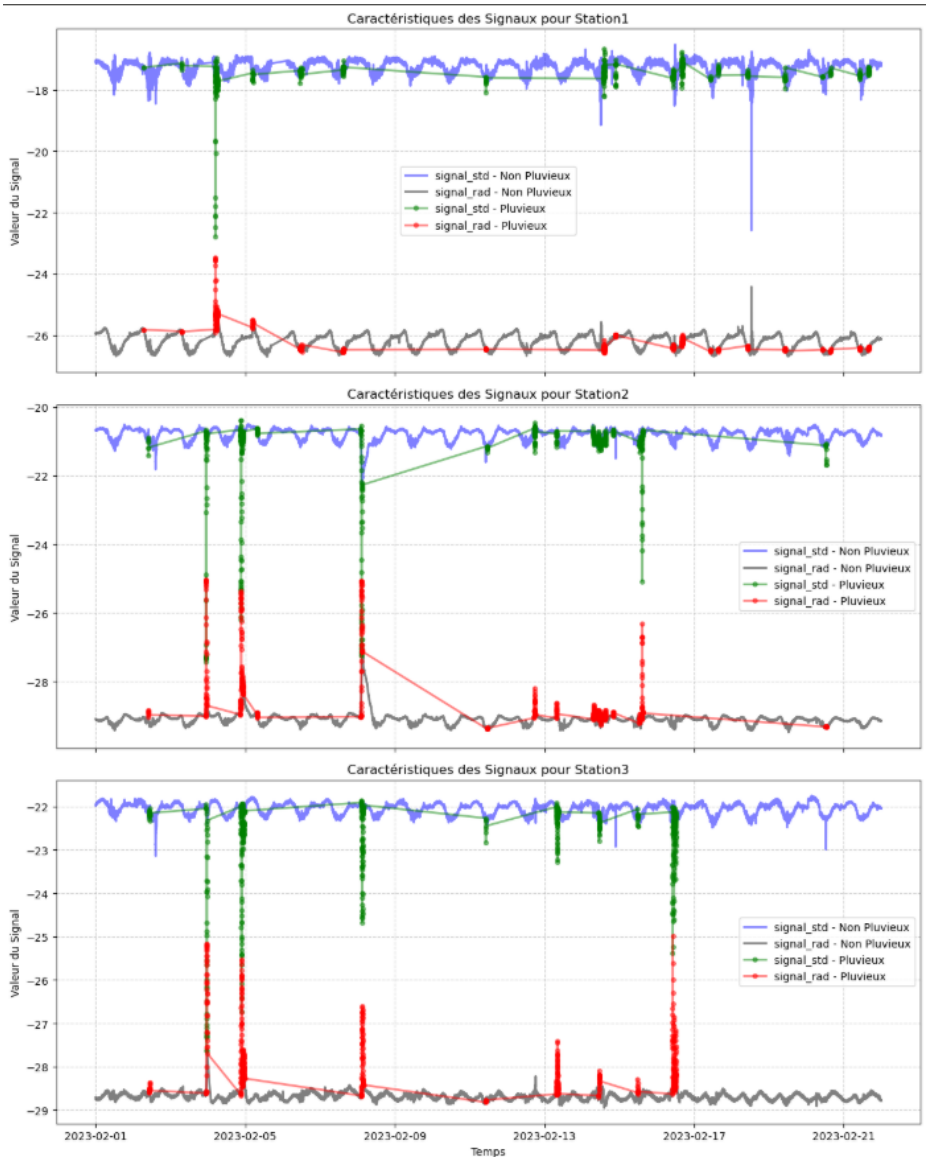

FIGURE 3.2 – Évolution temporelle des signaux standard et radiométriques.

Chapitre 4

Partie 3 : Modélisation et Traitement des Données

4.1 Amélioration de la prédiction

Pour améliorer la prédiction des événements météorologiques, nous avons introduit de nouvelles caractéristiques basées sur les variations des signaux et nous avons crée des features qui capturent les changements brusques des signaux.

4.1.1 Introduction de nouvelles caractéristiques basées sur les variations des signaux

Afin de capturer des changements brusques dans les signaux, des caractéristiques telles que les différences absolues entre les valeurs consécutives ont été introduites. Cela permet de détecter des événements météorologiques rapides, comme le début ou la fin d'une averse.

— **Différences absolues entre des valeurs consécutives** :

$$\Delta signal = signal(t) - signal(t-1)$$

Cela capture la vitesse de variation des signaux.

```
X_entrainement_validation['signal_std_delta'] = X_entrainement_validation[
X_entrainement_validation['signal_rad_delta'] = X_entrainement_validation[
X_test['signal_std_delta'] = X_test['signal_std'].diff()
X_test['signal_rad_delta'] = X_test['signal_rad'].diff()
```

4.1.2 Mesure de la disparité entre les signaux

Les différences entre les signaux standardisés (`signal_std`) et radiométriques (`signal_rad`) ont été calculées pour évaluer les transitions brusques pouvant correspondre à des changements météorologiques tels que le début de la pluie.

$$signal\_diff = |signal\_std - signal\_rad|$$

```
X_entrainement_validation['signal_diff'] = abs(X_entrainement_validation['signal_std
X_test['signal_diff'] = abs(X_test['signal_std'] - X_test['signal_rad'])
```

4.1.3 Moyenne mobile

Une moyenne mobile a été appliquée aux signaux pour lisser les données et éliminer le bruit, permettant ainsi d'extraire des tendances sous-jacentes.

$$signal\_std\_rolling\_mean = mean(signal\_std[t - n : t])$$

```
rolling_window = 5
X_entrainement_validation['signal_std_rolling_mean'] = X_entrainement_validation['si
X_entrainement_validation['signal_rad_rolling_mean'] = X_entrainement_validation['si
X_test['signal_std_rolling_mean'] = X_test['signal_std'].rolling(window=rolling_wind
X_test['signal_rad_rolling_mean'] = X_test['signal_rad'].rolling(window=rolling_wind
```

4.1.4 Mesure de la volatilité des signaux

La volatilité des signaux sur une période donnée a été calculée pour mesurer la stabilité. Une forte volatilité indique des changements rapides (par exemple, une pluie soudaine), tandis qu'une faible volatilité signale des conditions plus stables.

$$signal\_std\_volatility = std(signal\_std[t - n : t])$$

```
X_entrainement_validation['signal_std_volatility'] = X_entrainement_validation['sign
X_entrainement_validation['signal_rad_volatility'] = X_entrainement_validation['sign
X_test['signal_std_volatility'] = X_test['signal_std'].rolling(window=rolling_window
X_test['signal_rad_volatility'] = X_test['signal_rad'].rolling(window=rolling_window
```

4.1.5 Ratio entre les signaux

Un ratio entre les signaux standardisés et radiométriques a été calculé pour capturer les anomalies de proportion et leur évolution au fil du temps.

$$signal\_ratio = \frac{|signal\_std|}{|signal\_rad|}$$

```
X_entrainement_validation['signal_ratio'] = abs(X_entrainement_validation['signal_st
X_test['signal_ratio'] = abs(X_test['signal_std']) / abs(X_test['signal_rad'])
```

4.1.6 Gestion des valeurs manquantes

Les valeurs manquantes ont été vérifiées dans les ensembles d'entraînement et de test. Les résultats montrent que la plupart des caractéristiques ont très peu de valeurs manquantes. Toutefois, quelques valeurs manquantes dans les caractéristiques dérivées (comme `signal_std_delta` et `signal_rad_delta`) ont été gérées en supprimant les lignes concernées.

```
X_entrainement_validation.dropna(inplace=True)
```

4.1.7 Encodage des variables catégorielles

Les variables catégorielles, telles que `id_station`, ont été encodées à l'aide du One-Hot Encoding afin de convertir les stations en variables numériques exploitables par les modèles.

```
X_entrainement_validation = pd.get_dummies(X_entrainement_validation, columns=['id_s
X_test = pd.get_dummies(X_test, columns=['id_station'], prefix='station')
```

Les caractéristiques créées et l'encodage des variables permettront au modèle de mieux capturer les relations entre les signaux et les événements météorologiques, améliorant ainsi la qualité des prédictions.

4.1.8 Tri des Datasets par Ordre Temporel

Afin de préserver l'ordre temporel des données, les datasets sont triés en fonction de la variable `temps` pour éviter toute fuite d'information. Cela est réalisé en utilisant la méthode suivante :

```
X_entrainement_validation.sort_values(by='temps', inplace=True)
X_test.sort_values(by='temps', inplace=True)
```

Cette opération permet de garantir que les observations les plus anciennes se trouvent en tête des jeux de données d'entraînement et de test.

4.1.9 Séparation des Features (X) et des Labels (Y)

Les labels (ou cibles) sont séparés des features dans les datasets d'entraînement et de test. La colonne `pluie` représente la variable cible et est extraite de manière suivante :

```
Y_train_val = X_entrainement_validation['pluie']
Y_test = X_test['pluie']
```

Ensuite, les colonnes `pluie` et `temps` sont supprimées des features, car elles ne sont pas utilisées lors de l'entraînement des modèles.

```
X_entrainement_validation.drop(columns=['pluie', 'temps'], inplace=True)
X_test.drop(columns=['pluie', 'temps'], inplace=True)
```

4.1.10 Standardisation des Features Numériques

Pour garantir que toutes les features numériques aient des échelles similaires, nous utilisons la standardisation via un `MinMaxScaler`, qui normalise les valeurs entre 0 et 1. Certaines colonnes, comme les colonnes représentant des valeurs cycliques (`heure_cos`, `heure_sin`) ou celles liées aux stations, sont exclues de cette normalisation. Le processus est le suivant :

```
from sklearn.preprocessing import MinMaxScaler
columns_not_to_scale = ['heure_cos', 'heure_sin', 'pluie', 'temps'] + [col for col i
columns_to_scale = X_entrainement_validation.columns.difference(columns_not_to_scale
scaler = MinMaxScaler(feature_range=(0, 1))
scaler.fit(X_entrainement_validation[columns_to_scale])
X_entrainement_validation[columns_to_scale] = scaler.transform(X_entrainement_valida
X_test[columns_to_scale] = scaler.transform(X_test[columns_to_scale])
```

La taille des datasets après transformation est affichée comme suit :

```
Taille de X_entrainement_validation : (90716, 15)
Taille de X_test : (30240, 15)
```

4.1.11 Hybrid Sampling (SMOTE + Undersampling)

Le problème de déséquilibre des classes dans les données est adressé par une approche combinée d'oversampling et d'undersampling, utilisant la méthode SMOTE (Synthetic Minority Over-sampling Technique) couplée avec l'undersampling via `Tomek Links`. Cette combinaison permet de générer des exemples synthétiques pour la classe minoritaire tout en réduisant la classe majoritaire, ce qui améliore l'équilibre des classes sans introduire trop de faux exemples.

La mise en œuvre de SMOTE + Tomek Links se fait comme suit :

```
from imblearn.combine import SMOTETomek
from collections import Counter
smote_tomek = SMOTETomek(random_state=42)
X_train_val_resampled, Y_train_val_resampled = smote_tomek.fit_resample(X_entraineme
```

Avant rééchantillonnage, la répartition des classes est déséquilibrée, avec une majorité de la classe 0.0 (absence de pluie) :

```
Distribution des classes avant rééchantillonnage : Counter({0.0: 89294, 1.0: 1422})
```

Après application du rééchantillonnage, la répartition devient équilibrée entre les classes 0.0 et 1.0 :

```
Distribution des classes après rééchantillonnage : Counter({0.0: 89291, 1.0: 89291})
```

Cela permet d'assurer une meilleure performance des modèles sur des classes équilibrées et d'éviter le biais vers la classe majoritaire.

Chapitre 5

Partie 4 : Un peu de deep learning "utilisation du modèle LSTM"

5.1 Classification avec un Modèle LSTM

5.1.1 Préparation des Données

Les modèles LSTM (Long Short-Term Memory) nécessitent une entrée sous forme de tenseur 3D de dimensions $(samples, time\_steps, features)$, où : - `samples` représente le nombre d'exemples, - `time_steps` est la dimension temporelle, - `features` correspond aux caractéristiques des données.
Dans notre cas, les données sont réorganisées pour ajouter une dimension temporelle. Nous utilisons `reshape` pour ajuster les dimensions des données d'entraînement et de test :

```
X_train_reshaped = X_train_val_resampled.values.reshape(X_train_val_resampled.shape[
X_test_reshaped = X_test.values.reshape(X_test.shape[0], 1, X_test.shape[1])

X_train_reshaped = np.array(X_train_reshaped, dtype=np.float32)
Y_train_val_resampled = np.array(Y_train_val_resampled, dtype=np.float32)
X_test_reshaped = np.array(X_test_reshaped, dtype=np.float32)
Y_test = np.array(Y_test, dtype=np.float32)
```

Explication : Les modèles LSTM nécessitent que les données d'entrée soient sous la forme de séquences temporelles. Nous ajoutons donc une dimension temporelle avec `time_steps=1` pour permettre au modèle de traiter chaque observation comme une séquence de longueur 1.

5.1.2 Création du Modèle LSTM

Le modèle LSTM utilisé est composé de deux couches LSTM, de couches de Dropout pour éviter l'overfitting, et de couches denses pour la classification. La structure du modèle est la suivante :

```
model = Sequential([
    LSTM(128, return_sequences=True, input_shape=(1, X_train_val_resampled.shape[1])),
    Dropout(0.3),
    LSTM(64, return_sequences=False),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

Explication :

- La première couche LSTM (128 unités, `return_sequences=True`) capture les motifs temporels et transmet les séquences à la couche suivante.
- Une couche `Dropout(0.3)` désactive aléatoirement 30% des neurones pour éviter l'overfitting.
- La deuxième couche LSTM (64 unités, `return_sequences=False`) résume la séquence en un vecteur de sortie.
- La fonction d'activation `ReLU` (Rectified Linear Unit) est utilisée car elle est efficace pour éviter le problème des gradients qui disparaissent et permet un entraînement plus rapide dans les réseaux neuronaux profonds.
- Une seconde couche `Dropout(0.3)` est appliquée.
- Une couche dense (32 unités, activation `relu`) est ajoutée pour mieux capturer les relations complexes dans les données.
- La dernière couche dense (activation `sigmoid`) génère une probabilité pour la classification binaire (pluie ou pas de pluie) : une couche dense avec une seule unité, car il s'agit d'une classification binaire. La fonction d'activation `sigmoid` est utilisée pour transformer la sortie en une probabilité comprise entre 0 et 1, représentant la probabilité d'appartenir à la classe 1. Si la probabilité est supérieure à 0.5, l'exemple est classé comme classe 1, sinon il est classé comme classe 0.

5.1.3 Compilation du Modèle

- **Optimiseur (Adam)** : L'optimiseur Adam est un optimiseur adaptatif performant, qui ajuste les taux d'apprentissage pour chaque paramètre du modèle. Il combine les avantages des optimisateurs AdaGrad et RMSProp et est particulièrement adapté pour les modèles de deep learning complexes.

— **Fonction de perte (binary_crossentropy)** : La fonction de perte `binary_crossentropy` est utilisée pour les problèmes de classification binaire. Elle mesure la différence entre les prédictions du modèle et les vraies étiquettes de classe.
— **Métrique (accuracy)** : L'accuracy est utilisée comme métrique d'évaluation pour mesurer la proportion de prédictions correctes.

5.1.4 EarlyStopping

L'option `EarlyStopping` est utilisée pour arrêter l'entraînement lorsque la perte de validation cesse d'améliorer. Les paramètres utilisés sont les suivants :
— **monitor='val_loss'** : L'entraînement est surveillé en fonction de la perte sur les données de validation.
— **patience=10** : L'entraînement sera arrêté après 10 époques sans amélioration de la perte de validation.
— **restore_best_weights=True** : Lorsque l'entraînement est arrêté, les meilleurs poids trouvés au cours de l'entraînement seront restaurés pour obtenir le modèle le plus performant.

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

5.1.5 Gestion du Déséquilibre des Classes

Étant donné le déséquilibre des classes dans les données, nous utilisons `class_weights` pour ajuster l'impact des classes sur la fonction de perte. Cela permet au modèle de traiter de manière équilibrée les classes minoritaire et majoritaire :

```
class_weights = dict(enumerate(compute_class_weight('balanced', classes=np.unique(Y_
```

5.1.6 Early Stopping

Le mécanisme d'`EarlyStopping` permet d'arrêter l'entraînement lorsque la perte de validation (`val_loss`) ne s'améliore plus après un certain nombre d'époques. Cela aide à éviter l'overfitting. Le paramètre `patience` définit le nombre d'époques sans amélioration avant l'arrêt :

```
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)
```

5.1.7 Entraînement du Modèle

Le modèle est entraîné avec les données rééchantillonnées, en utilisant un nombre d'époques de 100 et une taille de batch de 32. Les `class_weights` et `early_stopping` sont utilisés pour améliorer la performance et éviter l'overfitting :

```
history = model.fit(X_train_reshaped, Y_train_val_resampled,
                    epochs=100,
                    batch_size=32,
                    validation_data=(X_test_reshaped, Y_test),
                    class_weight=class_weights,
                    callbacks=[early_stopping])
```

La sortie montre l'évolution de la précision et de la perte sur les données d'entraînement et de validation à chaque époque.

5.1.8 Prédiction

Une fois l'entraînement terminé, nous effectuons les prédictions sur le jeu de test en utilisant un seuil de 0.95 pour classer les sorties en 0 ou 1 :

```
Y_pred_lstm = (model.predict(X_test_reshaped) > 0.95).astype("int32")
```

5.1.9 Évaluation

La performance du modèle est évaluée à l'aide de la matrice de confusion et du rapport de classification. Les résultats montrent une précision globale élevée et un bon équilibre entre les classes :

```
Matrice de Confusion :
 [[29200   210]
 [  310   520]]

Rapport de Classification :
              precision    recall  f1-score   support
         0.0       0.99      0.99      0.99     29410
         1.0       0.71      0.63      0.67       830

    accuracy                           0.98     30240
   macro avg       0.85      0.81      0.83     30240
weighted avg       0.98      0.98      0.98     30240
```

5.1.10 Analyse des Résultats

— **Vrais négatifs (TN)** : 29 214 → Très peu d'erreurs pour la classe majoritaire (0).
— **Faux positifs (FP)** : 196 → Peu de prédictions incorrectes de 1 alors que c'était 0.
— **Faux négatifs (FN)** : 311 → Bien que des erreurs soient présentes, elles sont moins nombreuses.
— **Vrais positifs (TP)** : 519 → Hausse des détections correctes de la classe minoritaire (1).

Le modèle montre une amélioration en termes de `recall` pour la classe minoritaire (0.63) et une bonne `précision` (0.71), mais peut encore être amélioré en ajustant le seuil de prédiction. L'`accuracy` globale est de 98%, ce qui indique une performance générale très solide.

5.1.11 Courbe d'Apprentissage

La courbe d'apprentissage montre l'évolution de la perte (loss) pendant l'entraînement et la validation, avec EarlyStopping pour éviter le surapprentissage :

```
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Époques')
plt.ylabel('Perte')
plt.legend()
plt.title("Courbe d'apprentissage avec EarlyStopping")
plt.show()
```

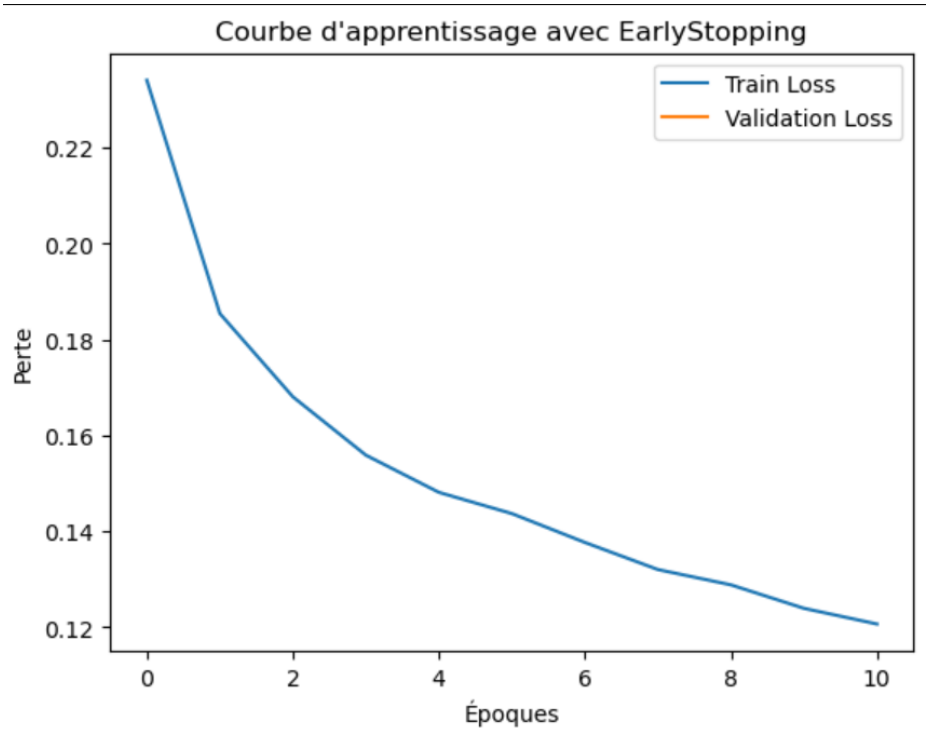

FIGURE 5.1 – Courbe d'apprentissage avec EarlyStopping :

Chapitre 6

Conclusion

Au terme de ce projet, nous pouvons affirmer que l'objectif de prédire les précipitations à partir des données météorologiques a été non seulement un défi scientifique, mais aussi une aventure enrichissante qui a mis en lumière l'importance des techniques de Deep Learning dans la résolution de problèmes complexes.
Tout au long de ce travail, nous avons exploré diverses étapes essentielles, depuis la préparation minutieuse des données, le prétraitement des features, jusqu'à l'application d'algorithmes avancés tels que le LSTM. Chaque étape a exigé une attention particulière et un investissement en temps considérable, ce qui a permis de mieux comprendre les subtilités des données météorologiques et d'explorer de nouvelles approches pour améliorer la prédiction des événements climatiques.
Nos efforts ont été couronnés par l'application du modèle LSTM qui, après des ajustements précis et une gestion rigoureuse des déséquilibres de classes, a permis d'obtenir des résultats significatifs, tout en démontrant l'efficacité des réseaux de neurones récurrents dans ce domaine spécifique.
Ce projet a été une expérience fascinante qui a non seulement élargi nos compétences techniques en Deep Learning et en Machine Learning, mais nous a rendu plus compétents pour aborder des projets similaires dans le domaine de la science des données.
Nous avons pleinement apprécié ce travail et sommes convaincus que les méthodes que nous avons explorées dans ce projet peuvent être étendues pour résoudre d'autres problématiques complexes en science des données.
Nous vous remercions chaleureusement de l'attention portée à ce projet et nous espérons que ce rapport vous a permis de mieux comprendre les enjeux et les solutions apportées par notre approche.