MACHINE LEARNING

Prévision du taux de grippe par région en France

Master M2 – MoSEF

3 janvier 2026

Plan de la présentation

Contexte et objectif

- **Objectif :** Prédire le **taux de grippe hebdomadaire** (pour 100 000 habitants) par région en France.
- **Métrique :** RMSE (Root Mean Squared Error)
- **Données disponibles :**
  - Données météorologiques SYNOP (154 fichiers, granularité horaire)
  - Google Trends (22 régions, granularité mensuelle)
  - Population INSEE (granularité annuelle)
  - Fichiers train/test avec taux de grippe hebdomadaire
- **Défi principal :** Harmoniser des sources de données à **granularités différentes** (horaire, mensuel, annuel) vers une granularité **hebdomadaire**.

Pipeline global

Vue d'ensemble du traitement des données

Source	Granularité	Transformation
SYNOP (météo)	Horaire	Agrégation hebdomadaire
↓		
Google Trends	Mensuel	Forward fill
↓		
Population INSEE	Annuel	Interpolation spline
↓		
`train_enrichi.csv` / `test_enrichi.csv`		
↓		
Modèle CatBoost (V12)		

Étape 1 : Agrégation SYNOP (horaire → hebdomadaire)

- **Objectif :** Transformer 154 fichiers SYNOP (observations horaires par station) en **variables hebdomadaires par région**.
- **Pourquoi c'est important :**
  - **Alignement d'échelle** : le modèle prédit par région/semaine
  - **Réduction du bruit** : l'horaire est volatile, l'hebdo stabilise
  - **Comparabilité** : même règle pour toutes les régions
- **Features créées :**
  - `temp_mean`, `temp_min`, `temp_max`
  - `humidity_mean`, `wind_speed_max`
  - `precipitation_sum`
- **Sortie :** `meteo_weekly.csv`

Mapping station → région

Problème : Les données SYNOP sont par **station**, pas par région.

Solution : Mapping manuel + fallback par coordonnées géographiques.

```
def get_region_from_coords(lat, lon, station_id=None):
    if station_id and station_id in MANUAL_STATION_MAPPING:
        return MANUAL_STATION_MAPPING[station_id]
    # fallback: regles lat/lon (approx)
    ...

df["region_code"] = df["numer_sta"].map(station_to_region)
df = df[df["region_code"].notna()]
```

Agrégation hebdomadaire :

```
df["week"] = df["datetime"].apply(get_week_id) # YYYYWW
weekly_df = df.groupby(["week", "region_code"]).agg(agg_dict)
```

Défi 1 : Franche-Comté (aucune station météo)

Problème :

- Aucune station SYNOP en Franche-Comté
- Impossible de calculer les features météo

Options considérées :

1. Imputer avec moyenne des régions voisines
2. Utiliser la station la plus proche

Solution retenue :

- Station **Bâle-Mulhouse** (frontalière)
- Justification : continuité spatiale de la météo

Région	Stations
Alsace	2
Lorraine	1
Bourgogne	1
Franche-Comté	**0**

↓

```
MANUAL_STATION_MAPPING = {
  "07299": "FRANCHE-COMTE"
  # Bale-Mulhouse
}
```

Étape 2 : Fusion avec Google Trends (mensuel → hebdomadaire)

- **Objectif :** Enrichir les données météo avec les recherches Google "grippe".
- **Difficulté :** Granularités différentes
  - Météo : **hebdomadaire** (clé `YYYYWW`)
  - Google : **mensuel** (clé `YYYY-MM`)
- **Solution :** Convertir semaine ISO en mois (via le jeudi de la semaine)
- **Features Google :**
  - `google_grippe` : volume brut
  - `google_grippe_no_aviaire` : sans "grippe aviaire"
  - `google_grippe_filtered` : filtré (sans H1N1, aviaire, etc.)
- **Sortie :** `synop_hebdo_google_enrichi.csv`

Alignement temporel : semaine → mois

Principe : Le jeudi ancre la semaine ISO (standard).

```
def week_to_month(week_id):
    year = week_id // 100
    week = week_id % 100
    jan4 = datetime(year, 1, 4)
    week1_monday = jan4 - timedelta(days=jan4.weekday())
    target_monday = week1_monday + timedelta(weeks=week - 1)
    target_thursday = target_monday + timedelta(days=3)
    return target_thursday.strftime("%Y-%m")

synop["month"] = synop["week"].apply(week_to_month)
merged = synop.merge(google_df, on=["month", "region_name"], how="left")
```

Harmonisation des noms de régions :

```
FILENAME_TO_REGION = {
  "IledeFrance": "ILE-DE-FRANCE",
  "ProvenceAlpesCotedAzur": "PROVENCE-ALPES-COTE-D-AZUR",
  ...
}
```

Étape 3 : Population INSEE (annuel → hebdomadaire)

- **Objectif :** Ajouter des indicateurs démographiques par région.
- **Pourquoi c'est important :**
  - **Effet taille** : plus de population = plus de cas potentiels
  - **Effet composition** : la grippe touche différemment selon l'âge
  - **Contrôle structurel** : séparer démographie vs météo/comportement
- **Difficulté :** Données **annuelles** vs granularité **hebdomadaire**
- **Solution :** Interpolation **spline cubique**
  - Trajectoire lisse et réaliste
  - Pas de "sauts" artificiels (vs forward-fill)
- **Sortie :** `synop_hebdo_complet.csv`

Interpolation spline cubique

```
from scipy.interpolate import CubicSpline

# Points d'ancrage : 1er janvier de chaque annee
anchor_dates = [datetime(year, 1, 1) for year in region_data["year"]]
anchor_numeric = [date_to_numeric(d) for d in anchor_dates]

# Spline cubique (extrapolation autorisee)
spline = CubicSpline(anchor_numeric, values, extrapolate=True)

# Interpoler pour chaque semaine
week_date = week_to_date(week_id) # jeudi ISO
pop_week = float(spline(date_to_numeric(week_date)))
```

Features dérivées :

```
merged["pop_ratio_elderly"] = (
    merged["pop_60_74"] + merged["pop_75_plus"]
) / merged["pop_total"]
```

Avantage spline vs forward-fill : Évite l'effet "escalier" non réaliste.

Défi 2 : Traitement des valeurs manquantes

Type de donnée	**Méthode**	**Justification**
Météo (temp, humidité)	Médiane	Robuste aux outliers
Google Trends	Valeur 0	Absence = pas d'intérêt
Lags (1ère semaine)	Médiane globale	Évite NaN en début de série
Features historiques	Médiane par région	Conserve distribution locale
Prédictions	Clip à 0	Taux négatif impossible

Principe général :

- **Médiane** plutôt que moyenne $\Rightarrow$ robustesse aux outliers
- **Imputation contextuelle** (par région/semaine quand possible)

Défi 3 : Outlier 2009 (Pandémie H1N1)

Constat :

Année	Moyenne	Max
2004	22	675
2005	95	1855
2006	60	1235
2007	73	1444
2008	75	1416
2009	**191**	**2478**
2010	30	569
2011	72	1573

Impact :

- Moyenne **3× plus élevée**
- Maximum historique absolu
- Risque d'overfitting

Stratégie retenue :

- **Non exclusion** (garder l'info)
- **Régularisation forte**
- Alternative testée : exclure 2009

Modèle V12 : vision globale

- **Algorithme :** CatBoostRegressor (gradient boosting)
- **Contraintes du problème :**
  - Forte **auto-corrélation temporelle** (effet mémoire)
  - **Saisonnalité marquée** (pics hivernaux)
  - Hétérogénéité **régionale**
  - Risque d'**overfitting** (peu d'années)
- **Choix méthodologiques :**
  - Sélection de **15 features** (sur 50+ disponibles)
  - **Régularisation forte**
  - **Validation temporelle** (dernière année complète)
- **Idée centrale :** La grippe est un phénomène **dynamique** dont l'évolution dépend fortement de son **passé récent**.

Feature Engineering : lags et saisonnalité

1) Mémoire épidémique (lags)

```
train["taux_lag1"] = train.groupby("region_code")["TauxGrippe"].shift(1)
train["taux_lag2"] = train.groupby("region_code")["TauxGrippe"].shift(2)
train["taux_diff1"] = train["taux_lag1"] - train["taux_lag2"]
```

Interprétation :

- `taux_lag1` : inertie immédiate
- `taux_diff1` : accélération/ralentissement

2) Saisonnalité explicite

```
df["sin_1"] = sin(2*pi*week_num/52)
df["cos_1"] = cos(2*pi*week_num/52)
df["is_flu_season"] = (week_num <= 12) | (week_num >= 45)
```

Pourquoi : Encodage continu (sin/cos) + régime binaire hiver.

Feature Engineering : contexte régional et Google

1) Historique régional (profil saisonnier)

```
agg = hist.groupby(["region_code","week_num"])["TauxGrippe"].agg(
    ["mean","median","std","max"]
)
df = df.merge(agg, on=["region_code","week_num"], how="left")
```

Lecture :

- `rw_max` : intensité maximale typique
- `rw_std` : volatilité habituelle

2) Google Trends (signal comportemental)

```
df["google_log"] = log1p(df["google_grippe_filtered"])
df["google_x_rw"] = df["google_log"] * df["rw_mean"]
```

Pourquoi : Volume de recherche = proxy de perception sanitaire.

3) Interactions clés

```
df["lag1_x_season"] = df["taux_lag1"] * df["is_flu_season"]
df["lag1_x_google"] = df["taux_lag1"] * df["google_log"]
```

Les 15 features sélectionnées

Feature	Importance
lag1_x_season	28.0%
taux_lag1	18.7%
lag1_x_google	14.2%
rw_max	12.2%
rw_std	8.9%
Top 5	*82%*
taux_diff1	5.6%
google_log	2.3%
taux_lag2	2.0%
...	...

Observation clé :

- **5 features** font **82%** du travail
- L'inertie temporelle domine
- Les 10 autres : <18% combinées

Conclusion :
Modèle simple avec features bien choisies > modèle complexe

Stratégie anti-overfitting

1) Validation temporelle stricte (pas de fuite)

```
train_f["year"] = train_f["week"].astype(str).str[:4]
train = data[year < max_year] # 2004-2010
val = data[year == max_year] # 2011
```

2) Régularisation forte du modèle

```
CatBoostRegressor(
    depth=4, # arbres peu profonds
    l2_leaf_reg=12, # penalisation L2
    min_data_in_leaf=60, # feuilles larges
    bagging_temperature=0.8,
    early_stopping_rounds=25
)
```

Effet :

- Arbres peu profonds $\Rightarrow$ règles simples
- Feuilles larges $\Rightarrow$ moins sensible au bruit

Prédiction récursive

Problème : En production, les valeurs passées du test sont inconnues.

Solution : Prédiction semaine par semaine, en utilisant les prédictions précédentes.

```
for week in test_weeks:
    # Utiliser les predictions precedentes comme lags
    week_data["taux_lag1"] = week_data["region_code"].map(last_pred)
    week_data["taux_lag2"] = week_data["region_code"].map(second_last_pred)

    # Predire
    pred_week = model.predict(week_data[features])

    # Mettre a jour les lags pour la semaine suivante
    second_last_pred[region] = last_pred[region]
    last_pred[region] = pred_week
```

Risque : Propagation des erreurs $\Rightarrow$ importance de la régularisation.

Évolution des modèles

Observation surprenante : Plus de features $\Rightarrow$ **pire** score Kaggle

Version	Features	Val RMSE	Kaggle RMSE
V7 (baseline)	18	58	$\sim$100
V10 (+ régularisation)	18	68	91
V12 (sélection)	**15**	**69**	**88**
V13 (minimal)	11	69	–

Leçons apprises :

- Val RMSE bas $\neq$ bon score Kaggle (**overfitting**)
- **Régularisation** : indispensable pour généraliser
- **Moins de features** : plus robuste
- **Validation réaliste** : année complète (pas split 80/20)

Résultat final

Modèle V12

Algorithme	CatBoostRegressor
Features	15
Validation RMSE	69
Kaggle RMSE	**88**

Clé du succès :
Simplicité + Régularisation forte + Validation temporelle

Synthèse et enseignements

Pipeline complet :

1. Agrégation SYNOP (horaire $\rightarrow$ hebdo)
2. Fusion Google Trends (mensuel $\rightarrow$ hebdo)
3. Interpolation population (annuel $\rightarrow$ hebdo, spline cubique)
4. Feature engineering (lags, saisonnalité, interactions)
5. Modèle CatBoost régularisé (15 features)

Enseignements clés :

- **Qualité des données** $>$ quantité de features
- **Régularisation** essentielle pour généraliser
- **Validation temporelle** réaliste (pas de fuite)
- Le **lag t-1** reste le meilleur prédicteur (inertie épidémique)

Défis surmontés : Franche-Comté (pas de station) ▪ Granularités hétérogènes ▪ Outlier 2009

Merci !

Questions ?