Analyse Word2Vec d'une base de données *Movies*

Rapport de projet de data science

Philippe ROUMBO

Université Sorbonne Paris Nord

Année universitaire 2025–2026

Table des matières

Chapitre 1

Introduction

L'objectif de ce projet est de combiner deux mondes qui m'intéressent particulièrement : d'un côté, un pipeline de *machine learning* classique sur des variables quantitatives, et de l'autre, une exploration plus avancée du texte avec des embeddings *Word2Vec*. L'idée est de partir d'une base de données de films (issue d'un dataset de type *movies_metadata*) et de construire à la fois :

— un modèle supervisé de régression pour prédire le logarithme des revenus (`ln_revenue`) à partir de variables numériques comme les votes, le budget, etc. ;

— un espace vectoriel des mots utilisés dans les synopsis (`overview`), afin de comprendre comment le vocabulaire des films se structure sémantiquement.

Concrètement, le projet se déroule en plusieurs étapes. D'abord, je nettoie les données, je prépare un premier jeu de variables quantitatives, puis je construis un modèle de régression linéaire qui me sert de *baseline*. Ensuite, je passe à une *Random Forest Regressor*, plus flexible, et j'évalue l'apport réel de ce modèle en termes de métriques (RMSE, MAE, R^2) et de validation croisée.

En parallèle, je consacre une deuxième partie complète au texte : nettoyage des synopsis, entraînement d'un modèle *Word2Vec* (skip-gram), réduction de dimension (ACP, t-SNE) et clustering des embeddings. L'objectif ici n'est pas seulement d'avoir un joli nuage de points, mais de montrer comment les mots se regroupent en « univers » thématiques cohérents (famille, guerre, amour, etc.), et de relier ces résultats au contexte des films.

Ce rapport suit une logique assez naturelle : je commence par poser la problématique, puis je présente les données et la méthodologie, avant de détailler les résultats, la discussion, les limites, et enfin les pistes d'amélioration possibles.

Chapitre 2

Problématique et cadre théorique

2.1 Problématique générale

La question de départ est relativement simple à formuler, mais pas si simple à traiter : *à partir des informations disponibles sur un film (notes, nombre de votes, popularité, etc.), peut-on expliquer et prédire une partie de ses performances au box office, et comment le langage utilisé dans les synopsis s'organise-t-il d'un point de vue sémantique ?*

Autrement dit, le projet repose sur deux volets complémentaires :

1. Un volet **régression supervisée**, pour modéliser la relation entre les caractéristiques d'un film et son revenu (en log).
2. Un volet **NLP / embeddings**, pour analyser la structure lexicale des descriptions de films à l'aide de *Word2Vec*.

2.2 Rappel sur l'apprentissage supervisé

En apprentissage supervisé, on dispose d'un jeu de données

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n},$$

où $x_i \in \mathbb{R}^p$ est un vecteur de caractéristiques (features) décrivant l'observation i, et y_i est la variable cible associée.

Le but est d'apprendre une fonction $f_\theta(\cdot)$, paramétrée par θ, qui approxime la relation inconnue (mais supposée exister) entre x et y :

$$y \approx f^\star(x),$$

que l'on approchera en pratique par une fonction

$$\hat{y} = f_\theta(x)$$

dont les paramètres θ sont appris à partir des données.

En régression, on choisit une **fonction de perte** $L(y, f_\theta(x))$, classiquement l'erreur quadra-

tique :

$$L(y_i, f_\theta(x_i)) = \big(y_i - f_\theta(x_i)\big)^2.$$

On définit alors le **risque empirique** :

$$R_{\text{emp}}(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f_\theta(x_i)) = \frac{1}{n}\sum_{i=1}^{n} \big(y_i - f_\theta(x_i)\big)^2,$$

et l'apprentissage consiste à trouver les paramètres qui le minimisent :

$$\hat{\theta} = \arg\min_\theta R_{\text{emp}}(\theta).$$

En théorie, on s'intéresse au **risque attendu** :

$$R(\theta) = \mathbb{E}_{(X,Y)}\big[L(Y, f_\theta(X))\big],$$

mais comme la vraie distribution de (X, Y) est inconnue, on utilise le risque empirique et des techniques comme la séparation *train/test* ou la *validation croisée*.

2.3 Modèle linéaire et Random Forest

Dans ce projet, je considère deux familles de modèles pour la régression :

— La **régression linéaire** multiple,

— La **Random Forest Regressor**.

2.3.1 Régression linéaire

En régression linéaire, on suppose que la relation entre x et y peut s'écrire sous forme :

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i,$$

ou, en notation vectorielle,

$$y_i = x_i^\top \beta + \varepsilon_i.$$

L'estimateur des moindres carrés ordinaires (MCO / OLS) est donné par :

$$\hat{\beta} = (X^\top X)^{-1} X^\top y,$$

lorsque $X^\top X$ est inversible.

Dans mon cas, je travaille en particulier avec un modèle du type :

$$\ln(\text{revenue}_i) = \beta_0 + \beta_1\, \text{vote\_average}_i + \beta_2\, \ln(\text{vote\_count}_i) + \varepsilon_i.$$

Ici, la variable dépendante est en log, ce qui permet de stabiliser la variance et de réduire l'impact des outliers.

2.3.2 Random Forest Regressor

La Random Forest est un modèle d'ensemble construit à partir de plusieurs arbres de décision : chaque arbre est entraîné sur un échantillon bootstrapé des données, et à chaque nœud, on sélectionne aléatoirement un sous-ensemble de variables candidate pour le split.

Si l'on note $f_1, \ldots, f_T$ les arbres individuels, la prédiction finale de la forêt pour un point x est :

$$\hat{f}(x) = \frac{1}{T} \sum_{t=1}^{T} f_t(x).$$

En régression, la Random Forest cherche à minimiser une fonction liée à la MSE en construisant progressivement des arbres qui réduisent la variance intra-feuille.

2.4 Word2Vec : cadre mathématique

Pour la partie texte, j'utilise un modèle **Word2Vec** de type *skip-gram*, qui apprend des représentations vectorielles des mots à partir de leurs contextes.

Après nettoyage, chaque synopsis j devient une séquence de mots :

$$\text{overview}_j = (w_{j1}, w_{j2}, \ldots, w_{jT_j}).$$

Le skip-gram apprend à prédire les mots de contexte w_c à partir d'un mot central w_t. Pour chaque paire (w_t, w_c), le modèle associe un vecteur d'entrée $v_{w_t} \in \mathbb{R}^d$ et un vecteur de sortie $v'_{w_c} \in \mathbb{R}^d$, et définit :

$$p(w_c \mid w_t) = \frac{\exp\left({v'_{w_c}}^{\top} v_{w_t}\right)}{\sum_{w \in V} \exp\left({v'_w}^{\top} v_{w_t}\right)},$$

où V est le vocabulaire.

L'entraînement consiste à maximiser la probabilité des contextes observés :

$$\max_{\{v_w, v'_w\}} \sum_{\text{paires } (w_t, w_c)} \log p(w_c \mid w_t),$$

en pratique avec des variantes comme le *negative sampling*.

Au final, je retiens les vecteurs d'entrée v_w comme **embeddings** des mots :

$$w \longmapsto v_w \in \mathbb{R}^d.$$

La similarité entre deux mots w_i, w_j peut alors se mesurer via la similarité cosinus :

$$\text{sim}(w_i, w_j) = \frac{v_{w_i}^{\top} v_{w_j}}{\|v_{w_i}\| \, \|v_{w_j}\|}.$$

Chapitre 3

Données et preprocessing

3.1 Description du dataset

Le dataset utilisé contient des informations sur des films : titre, date de sortie, revenu au box office, budget, votes, notation moyenne, popularité, indicateurs `adult` et `video`, ainsi que la colonne textuelle `overview` (synopsis).

Je sélectionne en particulier :

— `revenue`, `budget`, `popularity`, `vote_average`, `vote_count`,

— `adult`, `video` (variables binaires),

— `overview` pour la partie texte.

Je convertis les colonnes numériques en types adaptés, je filtre les valeurs aberrantes (par exemple les revenus négatifs ou manifestement invalides), et je crée des variables dérivées comme :

$$\texttt{ln\_revenue} = \ln(\text{revenue}), \quad \texttt{ln\_vote\_count} = \ln(\text{vote\_count}).$$

3.2 Mini EDA

Je commence par quelques statistiques descriptives et visualisations.

Histogrammes et distributions

Je trace des histogrammes pour `revenue`, `budget`, `ln_revenue`, etc., pour observer la forte asymétrie des revenus et l'effet de la transformation logarithmique.

Matrice de corrélation

Je calcule ensuite une matrice de corrélation sur les variables quantitatives. On observe notamment :

— une corrélation forte entre `revenue` et `vote_count` (environ 0.8) ;

— une corrélation modérée entre `vote_average` et `vote_count` ;

— des corrélations très faibles pour les variables `adult` et `video`, ce qui laisse penser qu'elles ne jouent qu'un rôle marginal dans la variation des revenus.

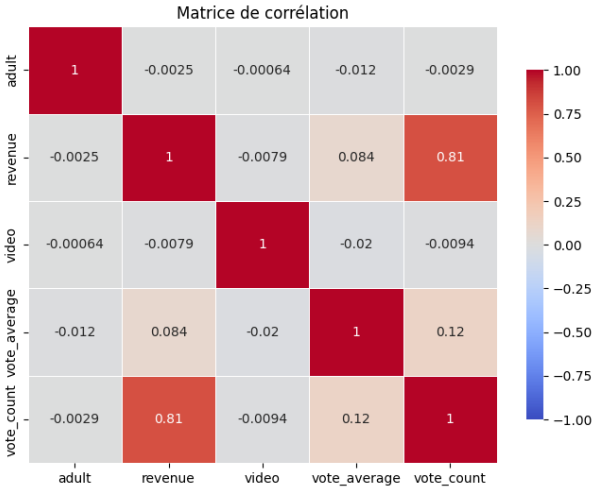

FIGURE 3.1 – Matrice de corrélation

Globalement, cette EDA montre que le nombre de votes (`vote_count`) est une variable clé, ce qui est logique : films très exposés → plus de votes → plus de revenus.

Chapitre 4

Méthodologie

4.1 Pipeline de modélisation supervisée

Pour la partie **prédiction de `ln_revenue`**, le pipeline est le suivant :

1. Sélection des variables explicatives (features) quantitatives et binaires.
2. Définition de la cible $y = \ln(\text{revenue})$.
3. Découpage du dataset en `train` / `test` (par exemple 80/20).
4. Standardisation des variables si nécessaire.
5. Entraînement d'une régression linéaire comme *baseline*.
6. Entraînement d'une Random Forest Regressor.
7. Évaluation sur le jeu de test via RMSE, MAE, R^2.
8. Validation croisée (5-fold) pour tester la robustesse de la Random Forest.

4.2 Préparation du texte pour Word2Vec

Pour la partie **texte**, je procède ainsi :

1. Nettoyage de la colonne `overview` : minuscules, suppression de la ponctuation, des caractères spéciaux, etc.
2. Suppression des *stopwords* (mots très fréquents peu informatifs).
3. Tokenisation : chaque overview devient une liste de mots.
4. Constitution d'un corpus sous la forme d'une liste de listes.
5. Entraînement de Word2Vec (skip-gram) sur ce corpus.

4.3 Réduction de dimension et clustering des embeddings

Une fois les vecteurs de mots appris, je veux les visualiser et les segmenter. La démarche est la suivante :

1. Construction d'une matrice X contenant les embeddings des mots sélectionnés.
2. Standardisation de X.

3. Application d'une ACP pour réduire la dimension et analyser la variance expliquée.
4. Utilisation de t-SNE pour obtenir une carte 2D plus intuitive des mots.
5. Clustering (K-means) sur les embeddings ou sur les composantes principales.
6. Évaluation des clusters via silhouette, Davies–Bouldin, Calinski–Harabasz.
7. Interprétation sémantique des clusters (thèmes).

Chapitre 5

Résultats

5.1 Résultats des modèles de régression

5.1.1 Régression linéaire

La régression linéaire sur `ln_revenue` en fonction de `vote_average` et `ln_vote_count` donne :

$$\ln(\text{revenue}_i) = \beta_0 + \beta_1 \,\text{vote\_average}_i + \beta_2 \,\ln(\text{vote\_count}_i) + \varepsilon_i.$$

Les coefficients estimés (arrondis) sont approximativement :

$$\beta_1 \approx -0.47, \qquad \beta_2 \approx 2.45.$$

Le modèle explique environ $R^2 \approx 0.43$ de la variance de `ln_revenue`. La MSE et ses dérivés se lisent comme :

$$\text{MSE} = \frac{1}{n_{\text{test}}} \sum_{i \in \text{test}} \left(y_i - \hat{f}(x_i)\right)^2,$$

$$\text{RMSE} = \sqrt{\text{MSE}}, \quad \text{MAE} = \frac{1}{n_{\text{test}}} \sum_{i \in \text{test}} |y_i - \hat{f}(x_i)|.$$

Dans mon cas, j'obtiens (en échelle log) une RMSE autour de 4.57 et une MAE autour de 3.24. Cela montre que le modèle capte un signal réel, mais que la variance résiduelle reste très importante.

Interprétation de β_2 (log–log). Comme la variable dépendante et `vote_count` sont toutes les deux en log, β_2 s'interprète comme une **élasticité** :

$$\frac{\partial \ln(\text{revenue})}{\partial \ln(\text{vote\_count})} = \beta_2.$$

Donc, pour de petites variations,

$$\%\Delta\text{revenue} \approx \beta_2 \times \%\Delta\text{vote\_count}.$$

	Model	RMSE	MAE	R2
0	LinearRegression	4.570385	3.237445	0.436667
1	RandomForest	4.657347	2.318216	0.415025

FIGURE 5.1 – Résultats OLS vs Random Forest

Avec $\beta_2 \approx 2.45$, une augmentation de 1 % de `vote_count` est associée à une augmentation d'environ 2.45 % du revenu, toutes choses égales par ailleurs.

Interprétation de β_1 (log–niveau). Ici, on est en format **log–niveau** pour `vote_average`. La variation d'un point de `vote_average` impacte ln(revenue) d'environ β_1 :

$$\Delta \ln(\text{revenue}) \approx \beta_1.$$

Pour obtenir une variation en pourcentage,

$$\%\Delta\text{revenue} = (e^{\beta_1} - 1) \times 100.$$

Avec $\beta_1 \approx -0.468$,

$$e^{-0.468} \approx 0.63 \quad \Rightarrow \quad \%\Delta\text{revenue} \approx (0.63 - 1) \times 100 \approx -37\%.$$

À popularité donnée (même `ln_vote_count`), gagner 1 point de note moyenne est donc associé à une baisse d'environ 37 % du revenu moyen. C'est contre-intuitif, mais cela colle avec l'idée que les films les plus « qualitatifs » ne sont pas forcément les plus « bankables » au box office.

5.1.2 Random Forest Regressor

La Random Forest obtient un R^2 légèrement inférieur (autour de 0.415) mais une MAE meilleure (2.32). Les deux modèles sont donc du même ordre de grandeur, avec des comportements un peu différents : la RF est plus précise sur les cas « moyens » mais se plante plus violemment sur certains outliers, ce qui augmente légèrement sa RMSE.

5.1.3 Validation croisée

La validation croisée 5-fold montre une RMSE moyenne autour de 4.43 (écart-type 1.32). On voit donc que :

— le modèle n'est pas en train de complètement sur-apprendre un split particulier ;

— mais ses performances restent sensibles à la façon dont on découpe les données (présence de cas extrêmes).

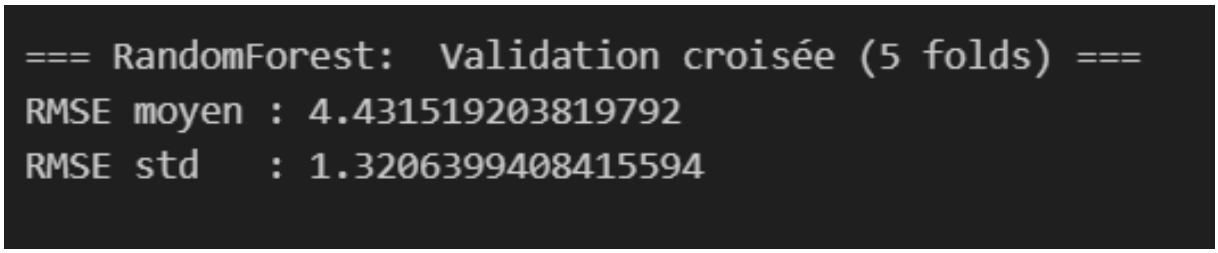

FIGURE 5.2 – Validation Croisée

5.2 Résultats Word2Vec

5.2.1 Qualité des embeddings

Quelques tests simples montrent que les embeddings sont cohérents. Par exemple, les mots les plus proches de `"bad"` contiennent `"good"`, et ceux de `"call"` contiennent `"caller"`, `"telephone"`, etc. Ce sont des *sanity checks* importants : si Word2Vec renvoyait des voisins complètement absurdes, il n'y aurait pas de raison de lui faire confiance pour la suite.

5.2.2 ACP et variance expliquée

L'ACP sur les embeddings standardisés montre qu'il faut un nombre assez important de composantes (pour atteindre par exemple 90 % de variance expliquée). La première composante capture une part relativement faible de la variance globale, ce qui est logique dans des espaces d'embeddings de grande dimension.

5.2.3 Clustering et indices de qualité

Pour choisir le nombre de clusters k pour K-means, je regarde la courbe de l'inertie (méthode du coude) et le score de silhouette moyen. Les résultats ne montrent pas un coude ultra net, mais un compromis raisonnable se situe autour de $k = 3$.

L'indice de silhouette reste assez faible (proche de 0.08), ce qui signifie que les clusters sont peu séparés et se chevauchent beaucoup. L'indice de Davies–Bouldin et celui de Calinski–Harabasz racontent la même histoire : la structure en clusters est présente mais « molle », ce qui est typique pour des données lexicales complexes où les thèmes se recoupent.

5.2.4 Interprétation sémantique des clusters

Malgré des scores de clustering modestes, l'inspection des mots par cluster permet d'identifier des grands thèmes :

- un cluster orienté **vie intime / famille / relations** (mots du type `mother`, `father`, `home`, `family`, `girl`, `wife`, etc.) ;
- un cluster plus **macro / contexte global / récit** (mots comme `story`, `film`, `world`, `war`, `people`, `years`) ;
- un cluster orienté **action, tension, police, guerre**, avec des termes comme `war`, `police`, `death`, `city`, `secret`, etc.

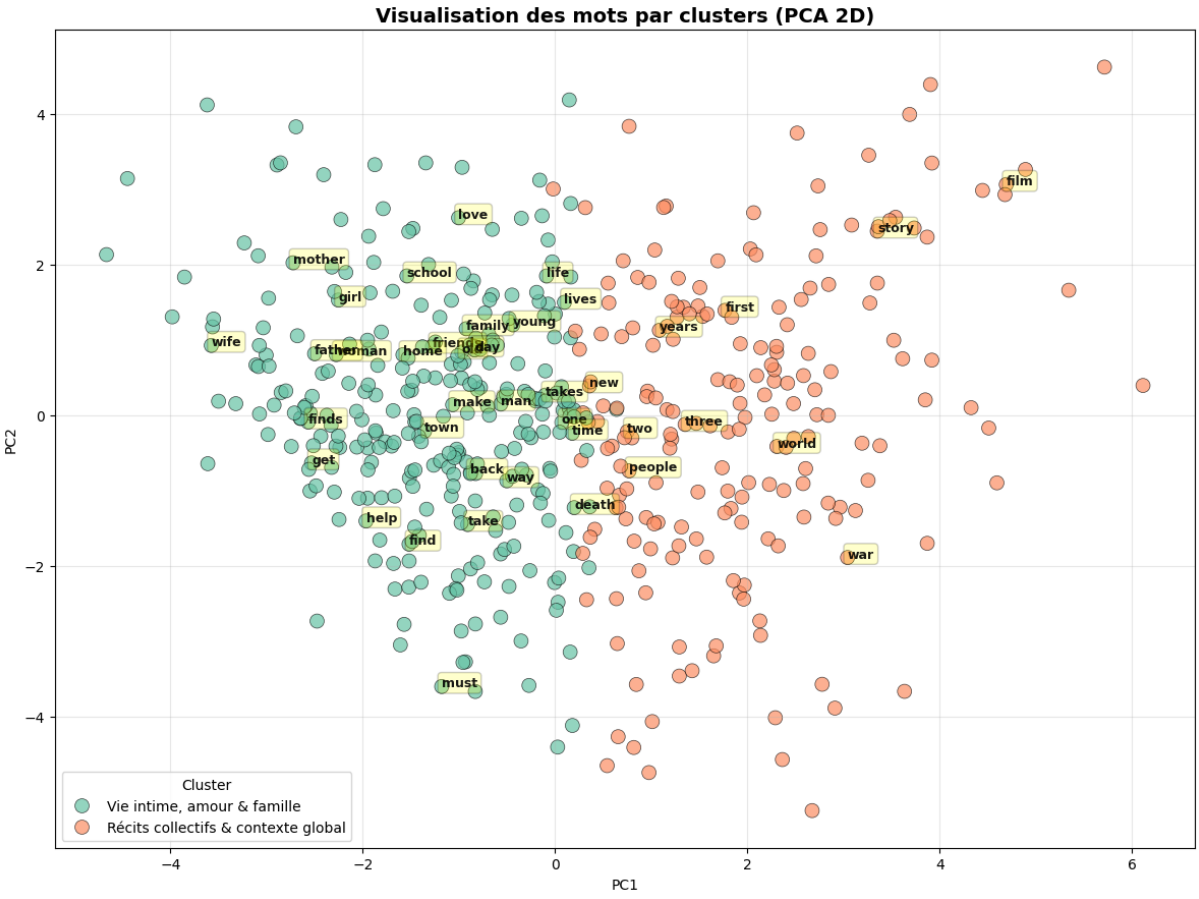

FIGURE 5.3 – Segmentation des Embedding

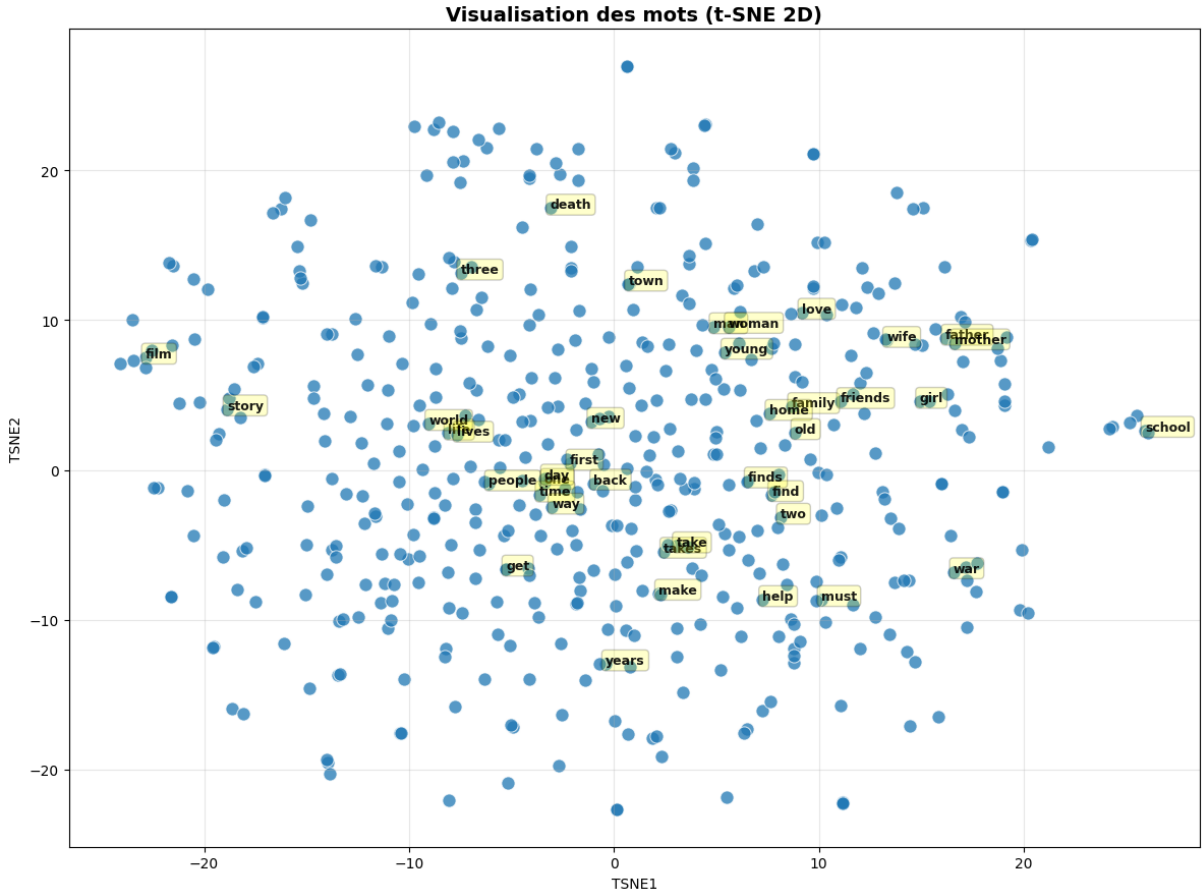

FIGURE 5.4 – t-SNE

Ces cartes montrent que le vocabulaire des films se structure en « univers » qui font sens du point de vue narratif.

5.2.5 Visualisation t-SNE

Enfin, l'application de t-SNE sur les embeddings (ou sur les composantes principales) permet de produire une carte 2D plus lisible, où l'on voit très clairement des zones de mots proches sémantiquement. On retrouve les thèmes identifiés plus haut (relations familiales, monde, guerre, etc.). Ce genre de visualisation est surtout utile pour le **storytelling** et pour montrer qu'au-delà des formules, le modèle capture réellement des proximités lexicales cohérentes.

Chapitre 6

Discussion

Globalement, la partie **ML supervisé** montre que, avec des variables assez classiques (votes, popularité, indicateurs binaires), on arrive à expliquer une fraction limitée de la variance de `ln_revenue` (environ 40–45 %). Ce n'est pas ridicule, mais cela reste loin d'un modèle prédictif véritablement précis, ce qui est logique étant donné la complexité du phénomène (campagnes marketing, licences, franchises, etc.).

La Random Forest, plus flexible, améliore la MAE mais pas le R^2. On voit donc qu'il existe des non-linéarités et des interactions que la linéaire ne capture pas, mais que la Random Forest, en l'état, ne parvient pas non plus à transformer en un gain massif de performance.

Côté **Word2Vec**, le modèle apprend un espace lexical cohérent, dans lequel les mots proches partagent des contextes similaires. Les analyses en ACP, en t-SNE et en clustering montrent que les mots se regroupent selon de grands axes thématiques (famille, guerre, récit global, etc.). Les indices de clustering ne sont pas exceptionnels, mais ce n'est pas surprenant : il est rare d'obtenir des groupes très nets sur un vocabulaire riche et hétérogène.

L'intérêt principal de cette partie n'est donc pas la « performance » au sens strict, mais plutôt la capacité à illustrer comment le langage des synopsis se structure, et comment on peut manipuler ces représentations pour faire du clustering ou des visualisations sémantiques.

Chapitre 7

Limites

Plusieurs limites du projet méritent d'être soulignées :

— **Variables explicatives limitées** : le modèle de régression ne voit qu'une petite partie de la réalité (quelques indicateurs numériques) et ignore beaucoup d'informations économiques ou marketing.

— **Données bruitées et outliers** : les revenus des films sont extrêmement dispersés, avec des blockbusters qui dominent largement, ce qui rend toute modélisation très sensible aux valeurs extrêmes.

— **Word2Vec basé uniquement sur les synopsis** : les embeddings ne capturent qu'une partie du contenu narratif (les textes d'overview), et pas, par exemple, les genres explicites, ni les informations sur les réalisateurs, acteurs, etc.

— **Clustering sensible aux hyperparamètres** : les résultats de K-means et de t-SNE peuvent varier selon la normalisation, le nombre de composantes PCA, le paramètre de perplexité de t-SNE, etc.

Chapitre 8

Pistes d'amélioration

Plusieurs pistes peuvent être explorées pour aller plus loin :

— **Enrichir les features** : intégrer des variables supplémentaires (genres, durée, informations sur la distribution, présence de stars, etc.) pour améliorer la partie prédictive.

— **Intégrer les embeddings au modèle de revenu** : au lieu de garder Word2Vec uniquement pour l'analyse sémantique, on pourrait agréger les embeddings au niveau du film (moyenne des vecteurs de mots, ou Doc2Vec) et les injecter comme features dans un modèle de régression.

— **Essayer d'autres modèles** : Gradient Boosting, XGBoost, LightGBM, ou même des modèles neuronaux, pour voir si des architectures plus complexes captent mieux les interactions.

— **Utiliser des modèles de langage plus récents** : BERT, GPT ou d'autres modèles de type Transformer pourraient fournir des embeddings contextuels plus riches, au-delà de Word2Vec.

Chapitre 9

Conclusion

Ce projet m'a permis de mettre en place, de bout en bout, un pipeline complet de data science qui combine des techniques de **modélisation supervisée** et des méthodes **NLP** basées sur des embeddings de mots.

Sur la partie régression, la régression linéaire fournit une baseline claire et interprétable, tandis que la Random Forest montre qu'il y a bien des non-linéarités, même si le gain en R^2 reste limité. Les mesures d'erreur (RMSE, MAE) confirment que le problème est intrinsèquement difficile, ce qui n'est pas surprenant dans un contexte de box office.

Côté texte, Word2Vec offre une représentation vectorielle intéressante des synopsis, et les analyses de type ACP, t-SNE et clustering montrent que le vocabulaire des films s'organise en grands univers thématiques relativement cohérents. Ces embeddings constituent une base prometteuse pour enrichir des modèles supervisés ou pour réaliser des analyses exploratoires plus fines.

En résumé, ce travail pose les fondations d'une approche hybride qui combine **quantitatif** et **textuel** pour mieux comprendre et modéliser le monde des films. Les résultats sont déjà parlants, mais ouvrent aussi de nombreuses perspectives pour des versions futures plus complètes, plus riches et plus ambitieuses.

Annexe A

Annexe : Code source Python

Dans cette annexe, je prévois d'inclure les extraits de code Python les plus importants issus du notebook (chargement des données, preprocessing, modèles, Word2Vec, ACP, clustering, etc.).

```
import pandas as pd                # Manipulation de tableaux / dataframes
    (lecture CSV, nettoyage, etc.)
import gensim                      # Lib NLP qui me sert pour entra ner
   le mod le Word2Vec
from nltk.corpus import stopwords  # Liste de mots vides (the, a, le, la
   ...)    filtrer dans les textes
import nltk                        # Outils NLP (ici surtout pour
   t l charger/gerer les stopwords)
import re                          # Expressions r guli res -> nettoyer
   le texte (ponctuation, caract res sp ciaux, etc.)
import numpy as np                 # Calculs num riques (log, arrays,
   gestion des NaN, etc.)
import statsmodels.formula.api as smf  # Mod les statistiques (
   r gression lin aire avec summary d taill )

from sklearn.decomposition import PCA        # R duction de dimension (
   PCA) sur les vecteurs Word2Vec
from sklearn.preprocessing import StandardScaler  # Standardisation des
   features avant PCA / clustering
from sklearn.manifold import TSNE            # t-SNE pour projeter les
   embeddings en 2D (visualisation)

from sklearn.metrics import silhouette_score # Mesure pour  valuer  la
   qualit  des clusters K-Means
from sklearn.cluster import KMeans           # Algorithme de clustering
   K-Means

import matplotlib.pyplot as plt    # Graphiques (scatter, courbes, etc.)
import seaborn as sns              # Visualisations un peu plus styl es
```

```
    (heatmap, scatter, etc.)

import kagglehub                    # Pour t l charger automatiquement
    le dataset depuis Kaggle
import missingno as msno            # Visualisation des valeurs manquantes
     dans le dataframe
from sklearn.model_selection import train_test_split, cross_val_score,
    GridSearchCV  #  Pour  la validation Crois e
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score,
    mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import silhouette_score, davies_bouldin_score,
    calinski_harabasz_score
```

```
# V rifier la version de pandas au cas  o
pd.__version__
```

```
# T l charger le dataset depuis Kaggle via kagglehub (si besoin)
path = kagglehub.dataset_download("rounakbanik/the-movies-dataset")
path
```

```
import os

# Chemin du fichier movies_metadata.csv dans le dossier t l charg
csv_path = os.path.join(path, "movies_metadata.csv")
csv_path
```

```
# Chargement du fichier CSV dans un DataFrame pandas
df = pd.read_csv(csv_path, low_memory=False)
df.head()
```

```
df.info()
```

```
# S lection des colonnes utiles pour la suite du projet
cols = [
    "revenue", "budget", "popularity", "vote_average", "vote_count",
    "adult", "video", "overview"
]
movies = df[cols].copy()
movies.head()
```

```
movies.info()
```

```
# Gestion des types : revenue et budget en num rique
movies["revenue"] = pd.to_numeric(movies["revenue"], errors="coerce")
movies["budget"] = pd.to_numeric(movies["budget"], errors="coerce")
```

```
# Conversion des champs bool ens / binaires (adult, video)
movies["adult"] = movies["adult"].astype("category")
movies["video"] = movies["video"].astype("category")

movies.info()
```

```
# Visualisation des valeurs manquantes
msno.matrix(movies)
plt.show()
```

```
# Suppression des lignes sans revenue ou vote_count ou overview
movies = movies.dropna(subset=["revenue", "vote_count", "overview"])
movies.shape
```

```
# Cr ation des variables log-transform es
movies["ln_revenue"] = np.log1p(movies["revenue"])
movies["ln_vote_count"] = np.log1p(movies["vote_count"])

movies[["revenue", "ln_revenue", "vote_count", "ln_vote_count"]].head()
```

```
# Mini stats descriptives
movies[["revenue", "budget", "popularity", "vote_average", "vote_count"
    ]].describe()
```

```
# Histogrammes de revenue et ln_revenue pour voir la diff rence
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
sns.histplot(movies["revenue"], bins=50, ax=axes[0])
axes[0].set_title("Distribution de revenue (brut)")

sns.histplot(movies["ln_revenue"], bins=50, ax=axes[1])
axes[1].set_title("Distribution de ln_revenue")

plt.tight_layout()
plt.show()
```

```
# Matrice de corr lation sur les variables quantitatives
num_cols = ["revenue", "budget", "popularity", "vote_average", "
    vote_count", "ln_revenue", "ln_vote_count"]
corr = movies[num_cols].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Matrice de corr lation (variables quantitatives)")
plt.show()
```

```
corr["revenue"].sort_values(ascending=False)
```

```
# Jeu de donn es pour la r gression : quelques variables simples pour
    d marrer
features = ["vote_average", "ln_vote_count"]
X = movies[features].copy()
y = movies["ln_revenue"].copy()

X.head(), y.head()
```

```
# Split train / test (80/20)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
X_train.shape, X_test.shape
```

```
# Mod le baseline : r gression lin aire
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

y_pred_lr = lin_reg.predict(X_test)

# Calcul des m triques
mse_lr = mean_squared_error(y_test, y_pred_lr)
rmse_lr = np.sqrt(mse_lr)
mae_lr = mean_absolute_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

rmse_lr, mae_lr, r2_lr
```

```
# R sum  OLS pour bien interpr ter les coefficients
movies_reg = movies[["ln_revenue", "vote_average", "ln_vote_count"]].
    dropna()

model_ols = smf.ols("ln_revenue ~ vote_average + ln_vote_count", data=
    movies_reg).fit()
print(model_ols.summary())
```

```
# Mod le Random Forest Regressor
rf = RandomForestRegressor(
    n_estimators=300,
    random_state=42,
    n_jobs=-1
)

rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

mse_rf = mean_squared_error(y_test, y_pred_rf)
```

```
rmse_rf = np.sqrt(mse_rf)
mae_rf = mean_absolute_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

rmse_rf, mae_rf, r2_rf
```

```
# Comparaison rapide des deux mod les
results = pd.DataFrame({
    "Mod le": ["R gression lin aire", "Random Forest"],
    "RMSE": [rmse_lr, rmse_rf],
    "MAE": [mae_lr, mae_rf],
    "R2": [r2_lr, r2_rf]
})
results
```

```
# Validation crois e sur la Random Forest (RMSE n gatif -> on remet le
    signe)
neg_mse_scores = cross_val_score(
    rf, X, y,
    cv=5,
    scoring="neg_mean_squared_error",
    n_jobs=-1
)

rmse_scores = np.sqrt(-neg_mse_scores)
rmse_scores, rmse_scores.mean(), rmse_scores.std()
```

```
# Pr paration du texte : overview
movies["overview"] = movies["overview"].astype(str)
movies["overview"].head()
```

```
nltk.download("stopwords")
stop_words = set(stopwords.words("english"))
```

```
def clean_text(text):
    # Minuscules
    text = text.lower()
    # Retirer tout ce qui n'est pas lettre ou espace
    text = re.sub(r"[^a-z\s]", " ", text)
    # Split
    tokens = text.split()
    # Retirer les stopwords et les tokens trop courts
    tokens = [t for t in tokens if t not in stop_words and len(t) > 2]
    return tokens

# Application sur la colonne overview
movies["tokens"] = movies["overview"].apply(clean_text)
movies[["overview", "tokens"]].head()
```

```
# Construction du corpus pour Word2Vec
corpus = movies["tokens"].tolist()
len(corpus), corpus[0][:30]
```

```
# Entra nement du mod le Word2Vec
w2v_model = gensim.models.Word2Vec(
    sentences=corpus,
    vector_size=100,
    window=10,
    min_count=5,
    sg=1,
    workers=4,
    epochs=10,
    seed=42
)

w2v_model
```

```
# petite v rification du vocabulaire
len(w2v_model.wv.index_to_key), w2v_model.wv.index_to_key[:20]
```

```
# Tests de similarit  pour sanity-check
w2v_model.wv.most_similar("bad", topn=10)
```

```
w2v_model.wv.most_similar("call", topn=10)
```

```
# S lection d'un sous-ensemble de mots pour l'ACP + clustering
# (par exemple les 1000 mots les plus fr quents)
top_n = 1000
vocab = w2v_model.wv.index_to_key[:top_n]

# Matrice d'embeddings
X_emb = np.array([w2v_model.wv[w] for w in vocab])
X_emb.shape
```

```
# Standardisation des embeddings
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_emb)
X_scaled.shape
```

```
# ACP sur les embeddings
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

explained_var = pca.explained_variance_ratio_
cum_explained_var = explained_var.cumsum()
```

```
plt.figure(figsize=(8, 5))
plt.plot(range(1, len(cum_explained_var) + 1), cum_explained_var, marker
    ="o")
plt.axhline(0.9, color="red", linestyle="--", label="90% variance
    expliqu e")
plt.xlabel("Nombre de composantes")
plt.ylabel("Variance expliqu e cumul e")
plt.title(" boulis  des valeurs propres (ACP sur Word2Vec)")
plt.legend()
plt.grid(True)
plt.show()
```

```
# Combien de composantes pour atteindre 90% ?
n_90 = np.argmax(cum_explained_var >= 0.90) + 1
n_90
```

```
# Visualisation en 2D des deux premi res composantes PCA
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], alpha=0.4, s=10)
plt.title("Projection PCA 2D des embeddings Word2Vec (top {} mots)".
    format(top_n))
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

```
# t-SNE sur les embeddings (optionnellement apr s PCA)
pca_50 = PCA(n_components=min(50, X_scaled.shape[1]), random_state=42)
X_pca_50 = pca_50.fit_transform(X_scaled)

tsne = TSNE(
    n_components=2,
    perplexity=30,
    random_state=42,
    learning_rate="auto",
    init="pca"
)

X_tsne = tsne.fit_transform(X_pca_50)
X_tsne.shape
```

```
# Visualisation t-SNE
plt.figure(figsize=(8, 6))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], alpha=0.6, s=10)
plt.title("Projection t-SNE des embeddings Word2Vec (top {} mots)".
    format(top_n))
plt.xlabel("t-SNE 1")
plt.ylabel("t-SNE 2")
plt.show()
```

```
# Choix du nombre de clusters K : on teste plusieurs K
inertias = []
sil_scores = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = kmeans.fit_predict(X_pca_50)
    inertias.append(kmeans.inertia_)
    sil_scores.append(silhouette_score(X_pca_50, labels))

inertias, sil_scores
```

```
# Affichage des courbes inertie / silhouette
fig, ax1 = plt.subplots(figsize=(8, 5))

color1 = "tab:blue"
ax1.set_xlabel("Nombre de clusters (k)")
ax1.set_ylabel("Inertie", color=color1)
ax1.plot(K_range, inertias, marker="o", color=color1)
ax1.tick_params(axis="y", labelcolor=color1)

ax2 = ax1.twinx()
color2 = "tab:orange"
ax2.set_ylabel("Score de silhouette", color=color2)
ax2.plot(K_range, sil_scores, marker="s", color=color2)
ax2.tick_params(axis="y", labelcolor=color2)

plt.title("Inertie et silhouette en fonction de k (K-Means sur
    embeddings)")
fig.tight_layout()
plt.show()
```

```
# On fixe k = 3 par exemple (en se basant sur la silhouette et la
    lisibilit  des th mes)
k_opt = 3
kmeans_opt = KMeans(n_clusters=k_opt, random_state=42, n_init=10)
labels_opt = kmeans_opt.fit_predict(X_pca_50)

np.unique(labels_opt, return_counts=True)
```

```
# Ajout des labels de clusters au DataFrame des mots
words_df = pd.DataFrame({
    "word": vocab,
    "cluster": labels_opt
})
words_df.head()
```

```
# Visualisation PCA 2D colore e par cluster
plt.figure(figsize=(8, 6))
for c in range(k_opt):
    mask = (labels_opt == c)
    plt.scatter(
        X_pca[mask, 0],
        X_pca[mask, 1],
        s=10,
        alpha=0.7,
        label=f"Cluster {c}"
    )

plt.title("PCA 2D des embeddings Word2Vec (couleurs = clusters)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend()
plt.show()
```

```
# Visualisation t-SNE 2D color e par cluster
plt.figure(figsize=(8, 6))
for c in range(k_opt):
    mask = (labels_opt == c)
    plt.scatter(
        X_tsne[mask, 0],
        X_tsne[mask, 1],
        s=10,
        alpha=0.7,
        label=f"Cluster {c}"
    )

plt.title("t-SNE des embeddings Word2Vec (couleurs = clusters)")
plt.xlabel("t-SNE 1")
plt.ylabel("t-SNE 2")
plt.legend()
plt.show()
```

```
# Liste des mots par cluster pour interpr tation
for c in range(k_opt):
    print(f"\n=== Cluster {c} ===")
    print(", ".join(words_df[words_df["cluster"] == c]["word"].head(50).
        tolist()))
```

```
# Indices de qualit  du clustering (sur X_pca_50)
sil_avg = silhouette_score(X_pca_50, labels_opt)
db_index = davies_bouldin_score(X_pca_50, labels_opt)
ch_index = calinski_harabasz_score(X_pca_50, labels_opt)
```

```
sil_avg, db_index, ch_index
```

```
# Exemple : focus sur quelques mots pour voir leurs positions en t-SNE
focus_words = ["family", "love", "war", "world", "police", "home", "
    friends", "father", "story", "film"]
focus_idx = [vocab.index(w) for w in focus_words if w in vocab]

plt.figure(figsize=(8, 6))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], s=5, alpha=0.15, color="
    lightgray")

for idx in focus_idx:
    x, y = X_tsne[idx, 0], X_tsne[idx, 1]
    w = vocab[idx]
    plt.scatter(x, y, s=30)
    plt.text(x + 0.5, y + 0.5, w, fontsize=9)

plt.title("t-SNE : position de quelques mots-cl s")
plt.xlabel("t-SNE 1")
plt.ylabel("t-SNE 2")
plt.show()
```