Expérimentation : évaluer l'impact du Kernel PCA sur la classification de sentiments

Afin de compléter l'étude théorique du Kernel PCA menée dans les sections précédentes, nous proposons ici une évaluation empirique de son efficacité dans un contexte d'apprentissage supervisé. L'objectif est de déterminer si cette méthode de réduction de dimension non linéaire permet d'améliorer les performances de modèles classiques de classification appliqués à une tâche de traitement du langage naturel (NLP).
Plus précisément, nous nous intéressons à la tâche de détection de sentiment à partir de critiques de films extraites du corpus IMDb [1]. Nous l'avons réduit à 20000 commentaire dont 10000 positifs et 10000 négatifs. Il s'agit d'un problème de classification binaire, où chaque texte doit être catégorisé comme exprimant un sentiment positif ou négatif. Ce cadre applicatif se prête naturellement à l'étude du Kernel PCA, car les documents textuels vectorisés via un modèle de type Bag-of-Words (CountVectorizer) sont typiquement très hauts dimensionnels et sparsely représentés, des conditions favorables à l'application de techniques de réduction de dimension.
Notre approche expérimentale repose sur plusieurs axes :

- Évaluer si l'application du Kernel PCA améliore les performances de modèles de classification classiques tels que le k-plus proches voisins (KNN), la régression logistique et les machines à vecteurs de support (SVM).
- Étudier l'impact du choix du noyau utilisé dans le Kernel PCA, en comparant notamment les noyaux cosinus, linéaire, RBF et polynomial. (jsp si je garde ça puisque je fais pas vraiment cette analyse)
- Déterminer un nombre optimal de composantes principales à conserver, à l'aide d'une analyse spectrale de la matrice noyau.
- Comparer les performances avec et sans réduction de dimension, en s'appuyant sur des métriques standard (accuracy, F1-score, AUC).
- Mesurer les coûts computationnels associés à chaque approche. (pareil jsp si je le fais ça)

L'ensemble de l'expérimentation a été implémenté en Python, et suit un pipeline rigoureusement structuré, allant du prétraitement textuel à l'évaluation des résultats, en passant par la vectorisation et la projection non linéaire. Les résultats obtenus sont présentés de manière comparative, et nous concluons par une synthèse des observations principales ainsi que des recommandations d'usage du Kernel PCA dans un tel cadre. (vérifier que tout ça c'est bien fais) L'intégralité du code utilisé pour cette expérimentation, ainsi que les figures et résultats complémentaires, est disponible sur le dépôt GitHub suivant : `https://github.com/ErwanR123/Sentiment-Analysis-on-IMDb-Movie-Reviews-using-Kernel-PCA`.

1 Prétraitement du texte

1.1 Objectif du prétraitement

Avant toute modélisation ou réduction de dimension, il est essentiel d'appliquer une phase de prétraitement aux données textuelles brutes. Dans notre cas, les données consistent en des critiques de films issues de la base IMDb, exprimées en langage naturel, non structuré et hétérogène. Ce type de données est particulièrement sujet à diverses sources de bruit textuel, susceptibles d'altérer la qualité de la vectorisation ainsi que l'efficacité des modèles de classification en aval. Parmi les nuisances couramment rencontrées, on peut citer :

- **Les balises HTML** introduites par le formatage du texte (par exemple, `<br/>`) ;
- **La ponctuation et les caractères spéciaux** (ex. : `! ? ...`) qui ne portent généralement pas d'information sémantique utile pour notre tâche ;
- **Les majuscules et minuscules**, qui peuvent engendrer une duplication artificielle du vocabulaire ;
- **Les chiffres**, rarement pertinents dans une analyse de sentiment, mais fréquents dans les dates ou numéros ;
- **Les contractions grammaticales** (par exemple, *didn't*, *I've*), qui doivent être développées pour être correctement analysées ;

- **Les formes conjuguées ou dérivées des mots**, qui doivent être ramenées à une forme canonique (lemmatisation) pour éviter une fragmentation inutile du vocabulaire ;
- **Les mots-outils (stopwords)**, très fréquents dans la langue mais souvent peu informatifs dans un contexte de classification (ex. : *the*, *and*, *of*) ;
- **L'absence d'uniformisation grammaticale**, qui empêche une analyse lexicale cohérente si l'on ne tient pas compte du rôle syntaxique de chaque mot.

Le prétraitement que nous avons mis en place s'appuie donc sur un pipeline complet comprenant les opérations suivantes :

1. **Développement des contractions** : transformation des formes contractées (*didn't* → *did not*, *I've* → *I have*) pour assurer une analyse lexicale correcte ;
2. **Nettoyage brut** : suppression des balises HTML, de la ponctuation, des chiffres, et passage en minuscules ;
3. **Tokenisation** : découpage du texte en unités lexicales (mots) ;
4. **POS-tagging (étiquetage grammatical)** : assignation d'une catégorie grammaticale à chaque mot à l'aide du tagger de NLTK [13] ;
5. **Lemmatisation** : réduction des mots à leur lemme en fonction de leur rôle syntaxique (par exemple, *was* devient *be*, *better* devient *good*). Cette méthode a été préférée à la racinisation (*stemming*), car elle préserve la lisibilité et la signification des mots. Le lemmatiseur utilisé (`WordNetLemmatizer` [13]) s'appuie sur les étiquettes grammaticales pour produire des formes canoniques plus précises ;
6. **Filtrage des stopwords** : suppression des mots peu informatifs, à l'exception des négations (*not, no, never*) et des modulateurs d'intensité (*very, just, too*) car leur présence peut inverser ou nuancer la polarité d'un avis. Cette décision s'appuie sur des travaux récents montrant que l'élimination systématique des *stopwords* peut dégrader les performances en analyse de sentiment [3].

Ce pipeline permet d'obtenir une représentation textuelle plus homogène, réduite en taille, et davantage focalisée sur les composantes sémantiquement discriminantes du texte. Cette étape est primordiale pour garantir une vectorisation efficace et une réduction de dimension pertinente dans la suite de notre étude. L'ordre des opérations a également été pensé pour maximiser la cohérence : la lemmatisation est effectuée après l'étiquetage POS, et le filtrage des stopwords est appliqué en dernier afin de ne pas supprimer prématurément des mots utiles à la désambiguïsation grammaticale.

1.2 Illustration sur un exemple

Pour illustrer concrètement les effets du prétraitement, considérons l'exemple suivant, tiré d'une critique typique contenant plusieurs types de bruit linguistique :

> *"I REALLY didn't like this movie!! It was `<br>` TOO long, with absolutely no plot. Just boring dialogues, and the acting ? Terrible. I've seen better in student films..."*

Ce commentaire contient des majuscules emphatiques, une balise HTML, de la ponctuation excessive, des mots de négation (*didn't, no*), des intensificateurs (*really, too, absolutely, just*), des conjugaisons (*didn't, was, seen*), et des stopwords.
Après application successive des étapes du pipeline, on obtient les représentations suivantes :

- **Après nettoyage brut** : *"i really didnt like this movie it was too long with absolutely no plot just boring dialogues and the acting terrible ive seen better in student films"*
- **Après lemmatisation avec POS-tagging** : *"really do not like movie be too long absolutely no plot just bore dialogue act terrible see well student film"*
- **Après filtrage des stopwords** : *"really not like movie too long absolutely no plot just bore dialogue act terrible see well student film"*

On observe ainsi une réduction significative de la longueur du texte, une normalisation des formes lexicales, et une mise en avant des mots porteurs de sens, tout en conservant les modificateurs de polarité.

1.3 Analyse du corpus après prétraitement

Après application du pipeline de prétraitement décrit précédemment, nous pouvons analyser plus précisément la transformation du corpus. Cette étape permet à la fois d'évaluer l'impact des opérations de nettoyage sur la structure lexicale des données, et de guider certains choix de paramètres, notamment lors de la vectorisation (cf. choix du paramètre `max_features` dans le `CountVectorizer`).

Principales statistiques obtenues avant et après le prétraitement :

- **Nombre total de mots avant prétraitement** : 4 622 845 ;
- **Nombre total de mots après prétraitement** : 2 448 293 ;
- **Nombre de mots uniques après prétraitement** : 111 127 ;
- **Réduction globale** : 47.04% de mots en moins.

On observe une réduction significative de la taille du corpus, traduisant l'élimination des éléments non pertinents et la normalisation lexicale. Le passage de plus de 4,6 millions de mots à environ 2,4 millions illustre à lui seul l'efficacité du filtrage.

Analyse du vocabulaire et couverture des occurrences La figure suivante montre la courbe de couverture cumulative des mots, classés par fréquence décroissante après prétraitement. L'objectif est ici de quantifier la part du vocabulaire la plus représentative du corpus, en analysant le nombre minimal de mots nécessaires pour couvrir une proportion donnée des occurrences.

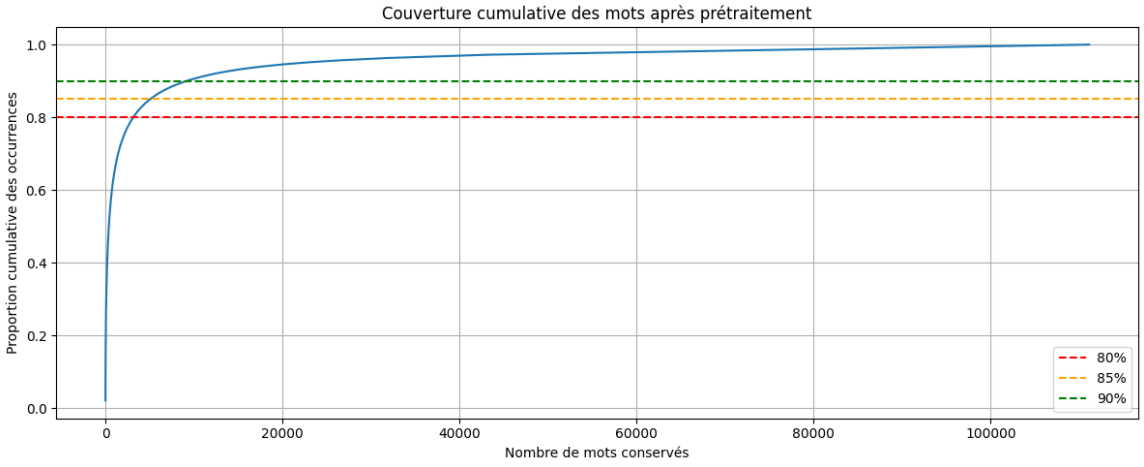

FIGURE 1 – Couverture cumulative des mots après prétraitement

Les résultats montrent une forte concentration de l'information sur un vocabulaire restreint :

- **3 141 mots suffisent à couvrir 80%** des occurrences ;
- **5 051 mots couvrent 85%** ;
- **9 162 mots couvrent 90%** du corpus.

Cette distribution confirme la loi de Zipf [2], selon laquelle une faible proportion des mots du vocabulaire concentre l'essentiel des occurrences. Ce constat justifie empiriquement l'utilisation d'un vocabulaire restreint pour la vectorisation. Néanmoins, afin de ne pas appauvrir excessivement la représentation lexicale et de laisser au Kernel PCA un espace suffisamment riche pour exprimer son potentiel de transformation non linéaire, nous fixons la taille maximale du vocabulaire à `max_features = 10000`. Ce choix permet de capturer plus de 90 % des occurrences du corpus, tout en conservant une diversité lexicale significative pour l'analyse ultérieure.

2 Vectorisation du texte

2.1 Représentation des textes par sac de mots

Après le prétraitement, les critiques textuelles sont encore sous forme de chaînes de caractères. Afin de pouvoir les exploiter dans des modèles d'apprentissage automatique, il est nécessaire de les convertir en une représentation numérique exploitable. L'approche la plus simple et la plus courante consiste à utiliser le modèle du sac de mots (*Bag-of-Words*, ou BoW), qui repose sur l'hypothèse que la présence (ou la fréquence) des mots dans un document est plus importante que leur ordre d'apparition [4].
Ce modèle transforme chaque critique en un vecteur de dimension fixe, dont chaque composante représente la fréquence d'un mot spécifique issu du vocabulaire global appris sur l'ensemble du corpus. Plus formellement, on considère un corpus prétraité $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}$ composé de n documents. Le vocabulaire V est l'ensemble des k mots les plus fréquents dans le corpus (après prétraitement). Chaque document d_i est alors représenté par un vecteur $x_i \in \mathbb{R}^k$, où x_{ij} correspond au nombre d'occurrences du mot j dans d_i.
Cette représentation, bien que simple, permet aux modèles classiques (KNN, régression logistique, SVM, etc.) de traiter les textes comme des vecteurs de caractéristiques standards [5]. Toutefois, elle souffre de deux limites majeures :

- **La très forte dimension** du vecteur lorsque le vocabulaire est riche (plusieurs dizaines de milliers de mots), ce qui augmente les coûts de calcul, et peut rendre les modèles sensibles au sur-apprentissage ;
- **L'absence de prise en compte du contexte** ou de la sémantique entre les mots : deux mots synonymes (ex. : *excellent*, *great*) sont considérés comme indépendants, et un mot unique porte la même importance quel que soit son usage [6].

2.2 Illustration de la vectorisation sur un exemple

Pour mieux comprendre le fonctionnement du modèle *bag-of-words*, considérons de nouveau l'exemple de critique prétraitée présenté précédemment :

> *"really not like movie too long absolutely no plot just bore dialogue act terrible see well student film"*

En pratique, le vocabulaire utilisé pour la vectorisation est appris uniquement sur le corpus d'entraînement. Il peut donc ne contenir qu'une partie des mots du commentaire, tout en incluant des mots fréquents absents de cette critique. Supposons par exemple que le vocabulaire extrait soit le suivant (limité ici à 16 mots pour des raisons de lisibilité) :

```
["excellent", "bad", "movie", "plot", "boring", "great", "terrible", "dialogue",
     "predictable", "not", "like", "fun", "slow", "awful", "love", "just"]
```

Parmi ces mots, certains apparaissent dans la critique (`movie`, `plot`, `not`, `like`, `dialogue`, `terrible`, `just`), tandis que d'autres (`excellent`, `great`, `fun`, etc.) sont absents.
La critique est alors vectorisée en un vecteur de dimension 16, où chaque composante indique le nombre d'occurrences du mot correspondant. On obtient le vecteur suivant :

```
[0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1]
```

La représentation tabulaire suivante permet de visualiser précisément cette vectorisation :

Mot	excellent	bad	movie	plot	boring	great	terrible	dialogue
Occur.	0	0	1	1	0	0	1	1

Mot	predictable	not	like	fun	slow	awful	love	just
Occur.	0	1	1	0	0	0	0	1

TABLE 1 – Représentation vectorielle (BoW) partielle d'une critique prétraitée

Ce type de représentation permet d'utiliser des modèles linéaires classiques. Toutefois, comme on le constate, elle ignore la sémantique fine, la structure grammaticale, et l'ordre des mots. Ces limitations motivent l'usage d'une réduction de dimension non linéaire comme le *Kernel PCA*, qui sera introduit dans la suite.

2.3 Choix de la taille du vocabulaire

Afin de limiter la dimensionnalité tout en conservant l'essentiel de l'information lexicale, nous avons fixé un seuil maximal de `max_features = 10 000` mots pour la vectorisation. Ce choix est motivé empiriquement par l'analyse du corpus après prétraitement : comme nous l'avons vu précédemment, les 10 000 mots les plus fréquents permettent de capturer plus de 90 % des occurrences du corpus. Ce compromis permet d'obtenir un vocabulaire suffisamment riche pour l'analyse tout en maintenant une complexité computationnelle raisonnable [7].

2.4 Implémentation pratique

Nous utilisons l'outil `CountVectorizer` [14] de la bibliothèque `scikit-learn` pour construire la matrice de caractéristiques. Afin d'éviter toute fuite d'information, la vectorisation est effectuée uniquement à partir du sous-ensemble d'entraînement. Plus précisément :

1. Le corpus textuel est d'abord scindé en un ensemble d'entraînement (70 %) et un ensemble de test (30 %) ;
2. Le vocabulaire est appris sur l'ensemble d'entraînement uniquement via `vectorizer.fit` ;
3. Les deux ensembles sont ensuite vectorisés via `transform`, en utilisant le même dictionnaire.

Cela garantit que le modèle n'a pas accès à des informations issues du jeu de test lors de l'apprentissage. Les matrices obtenues X_{train} et X_{test} sont ensuite utilisées comme entrée dans les modèles de classification et les algorithmes de réduction de dimension (cf. section suivante). Bien que cette représentation vectorielle soit directement exploitable par les modèles d'apprentissage, sa très haute dimension incite à appliquer une réduction de dimension, comme le Kernel PCA, pour en extraire une structure plus compacte et potentiellement plus informative.

3 Réduction de dimension par Kernel PCA

3.1 Objectif de la réduction de dimension

La représentation vectorielle obtenue par le modèle BoW est de très haute dimension (10 000 caractéristiques dans notre cas), ce qui pose plusieurs problèmes : augmentation du temps de calcul, risque de sur-apprentissage, sparsité extrême, et difficultés à identifier des structures exploitables par les algorithmes de classification. La réduction de dimension permet de projeter les données dans un espace plus compact, tout en conservant l'information essentielle à la tâche d'apprentissage supervisé.
Le *Kernel PCA* (KPCA) est un candidat particulièrement adapté à cette tâche, car il permet de capturer des structures non linéaires dans les données grâce à l'utilisation d'un noyau, contrairement au PCA classique qui se limite aux relations linéaires.

3.2 Motivation pour le choix du noyau cosinus

Dans le cas de données textuelles vectorisées par un sac de mots, les vecteurs sont très clairsemés (sparse) et souvent normalisés en norme ℓ_2. Dans ce contexte, la **similarité cosinus** est particulièrement adaptée car elle mesure l'angle entre les vecteurs plutôt que leur distance euclidienne brute [8]. Elle est donc plus pertinente pour comparer des documents longs ou courts exprimant un même contenu.
Le noyau cosinus est défini comme :

$$k(x, x') = \frac{\langle x, x' \rangle}{\|x\| \cdot \|x'\|}$$

où $\langle x, x' \rangle$ désigne le produit scalaire et $\|x\|$ la norme euclidienne. Ce noyau satisfait les conditions de positivité définie (Mercer) [9], ce qui garantit son utilisation valide dans le cadre du Kernel PCA.
Ce noyau mesure la similarité angulaire entre deux vecteurs, indépendamment de leur norme. Il est particulièrement bien adapté aux représentations vectorielles issues de la vectorisation textuelle, car :

- les vecteurs de mots ont des longueurs très variables selon la taille du document ;
- l'information porte davantage sur la direction du vecteur (présence relative des mots) que sur son amplitude ;
- la cosinus-similarité est un standard reconnu dans les tâches de classification et de recherche d'information textuelle [8].

3.3 Pourquoi les autres noyaux ne sont pas adaptés

Plusieurs autres noyaux sont classiquement utilisés avec le *Kernel PCA*, notamment :

- **Le noyau linéaire** : $k(x, y) = \langle x, y \rangle$
  Il revient à appliquer une PCA classique dans l'espace d'origine, sans transformation non linéaire. Il ne capte donc pas de structures complexes dans l'espace vectoriel initial.
- **Le noyau RBF (gaussien)** : $k(x, y) = \exp(-\gamma \|x - y\|^2)$
  Bien que puissant pour des données continues et denses, ce noyau dépend fortement de la norme euclidienne, ce qui le rend peu adapté aux vecteurs creux typiques du BoW. Il est également sensible à l'échelle des données et nécessite un réglage délicat de l'hyperparamètre γ.
- **Les noyaux polynomiaux** : $k(x, y) = (\langle x, y \rangle + c)^d$
  Ces noyaux peuvent amplifier artificiellement la variance sur certaines dimensions et manquent de robustesse pour des vecteurs très déséquilibrés comme ceux issus de corpus textuels.
- **Le noyau sigmoïde** : $k(x, y) = \tanh(\alpha \langle x, y \rangle + c)$
  Dérivé des réseaux de neurones, ce noyau est souvent instable, notamment pour des données non centrées ou faiblement normalisées, et rarement utilisé en pratique pour des représentations textuelles.

En comparaison, le noyau cosinus :

- est **invariant à la norme**, ce qui est crucial dans un espace BoW à sparsité élevée ;
- ne nécessite aucun hyperparamètre complexe ;
- s'aligne avec les pratiques standards en traitement du langage naturel.

Pour toutes ces raisons, le noyau cosinus constitue un choix à la fois théoriquement fondé, empiriquement validé dans la littérature, et particulièrement adapté à notre cas d'étude.

3.4 Analyse spectrale et choix du nombre de composantes

Une fois le noyau cosinus choisi, nous avons déterminé le nombre optimal de composantes principales à conserver. Pour cela, nous avons appliqué le Kernel PCA sur l'ensemble d'entraînement X_{train} vectorisé, et analysé la **variance cumulée** capturée par les composantes principales.
La variance cumulée mesure la proportion d'information (ou de dispersion) préservée par les k premières composantes principales. Chaque composante i est associée à une valeur propre λ_i, qui reflète la part de la variance expliquée. La variance cumulée associée aux k premières composantes est définie par :

$$\text{Var}_{\text{cumulée}}(k) = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{p} \lambda_j} \tag{1}$$

où p désigne le nombre total de composantes disponibles (typiquement, $p = \min(n, d)$, avec n le nombre d'échantillons et d la dimension initiale).

Remarque sur la notion de variance expliquée dans le Kernel PCA

Dans le cas du PCA classique, chaque composante principale est associée à une valeur propre λ_i de la matrice de covariance, qui représente directement la quantité de variance expliquée par cette direction. Cette interprétation repose sur le fait que le PCA maximise la variance projetée dans un espace linéaire, et que la somme des λ_i correspond exactement à la variance totale des données.
Dans le cas du **Kernel PCA**, l'analyse se fait dans un espace de Hilbert reproduisant (espace de caractéristiques implicite) dans lequel les données sont projetées via une fonction noyau $k(x, x')$. L'algorithme ne manipule pas les vecteurs projetés eux-mêmes, mais uniquement leur produit scalaire dans cet espace via le noyau.
Les valeurs propres λ_i du noyau centré K obtenu à partir de $k(x, x')$ n'ont donc plus une interprétation directe en termes de variance « physique », mais elles restent liées à la quantité d'information capturée par les composantes principales dans l'espace transformé. Autrement dit :

- λ_i mesure l'inertie (ou la dispersion) des données projetées sur la i-ème composante principale dans l'espace de Hilbert ;

- la somme $\sum_i \lambda_i$ donne une mesure de la complexité (ou diversité) des données dans cet espace ;
- le rapport $\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{p} \lambda_j}$ est interprété comme une *variance cumulée relative*, utilisée comme heuristique pour déterminer le nombre de composantes à conserver.

Cette interprétation est conforme à ce qui est proposé dans la littérature [10], et elle est utilisée dans la plupart des implémentations de Kernel PCA, notamment celle de `scikit-learn`.

> *« Although the feature space is not explicitly computed, the eigenvalues of the centered kernel matrix can still be interpreted as the variances explained by the corresponding eigenvectors. »* [14]

Ainsi, bien qu'il faille rester prudent sur la signification exacte de la variance dans un espace reproduisant implicite, l'analyse des valeurs propres du noyau centré permet néanmoins d'évaluer la proportion d'information capturée par les premières composantes. Cela constitue une approche raisonnable pour guider le choix du nombre de composantes à conserver dans le cadre du Kernel PCA.
Le graphe ci-dessous montre la proportion de variance capturée en fonction du nombre de composantes :

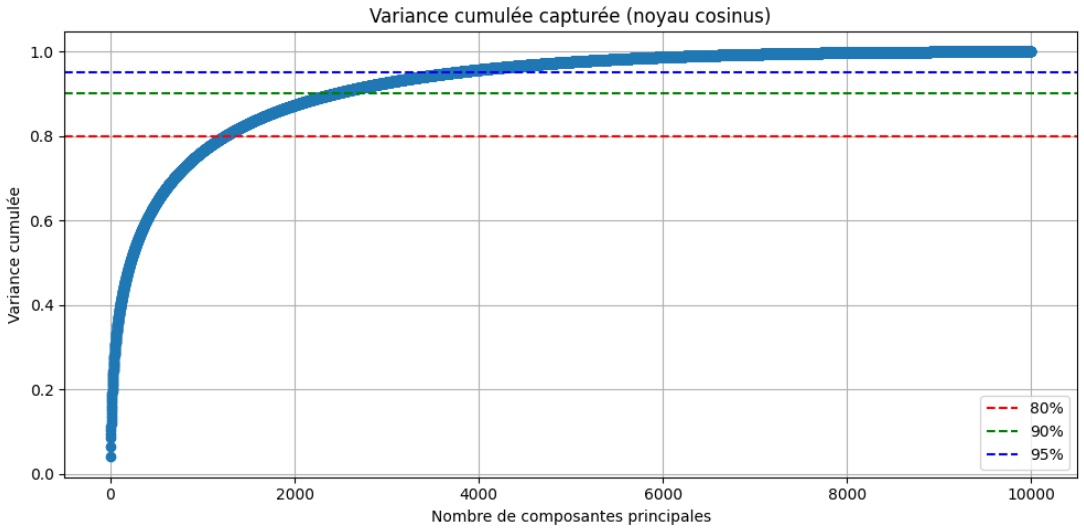

FIGURE 2 – Variance cumulée capturée par les composantes principales (noyau cosinus)

Nous obtenons les résultats suivants :

- 1 244 composantes suffisent à capturer 80% de la variance ;
- 1 709 pour 85% ;
- 2 429 pour 90% ;
- 3 723 pour 95%.

Ces résultats révèlent une décroissance rapide des valeurs propres, ce qui suggère que les données se concentrent dans un sous-espace de dimension bien inférieure à celle du BoW initial. Le choix du nombre de composantes est donc un compromis entre compacité et fidélité de l'information.
Nous retenons par défaut la valeur de 2 429 composantes principales, qui permet de capturer 90 % de la variance. Ce choix garantit une réduction substantielle de la dimension (divisée par plus de 4), tout en conservant l'essentiel de la structure informative du corpus. D'autres valeurs de k seront explorées dans la suite, afin d'évaluer l'impact de cet hyperparamètre sur la performance des modèles, et déterminer s'il est possible d'obtenir d'excellents résultats avec un nombre de composantes encore plus faible.

Application de la projection par Kernel PCA

Une fois le noyau choisi et le nombre optimal de composantes principales fixé à 2 429, nous appliquons le *Kernel PCA* à l'ensemble d'entraînement vectorisé X_{train} afin d'obtenir une projection dans un espace de dimension réduite. Le même modèle de KPCA est ensuite utilisé pour transformer l'ensemble de test X_{test} :

$$X_{\text{train}}^{\text{kpca}} = \text{KPCA.fit\_transform}(X_{\text{train}})$$
$$X_{\text{test}}^{\text{kpca}} = \text{KPCA.transform}(X_{\text{test}})$$

Ces nouvelles représentations serviront de base à l'évaluation comparative des modèles de classification, que nous détaillerons dans la section suivante.

4 Modélisation et évaluation des performances

4.1 Présentation des modèles de classification

Nous avons retenu trois modèles supervisés classiques, représentatifs de familles différentes d'algorithmes, et offrant des profils complémentaires en termes de complexité, de linéarité et de sensibilité à la dimensionnalité : le classifieur des k plus proches voisins (KNN), la régression logistique, et la machine à vecteurs de support (SVM). Leur diversité permet de mieux apprécier l'impact de la réduction de dimension sur des architectures aux comportements distincts.

K plus proches voisins (KNN). Le classifieur KNN est une méthode non paramétrique fondée sur le principe suivant : pour prédire la classe d'un échantillon, on identifie ses k voisins les plus proches dans l'espace des caractéristiques, selon une distance définie (souvent euclidienne), et on assigne la classe majoritaire parmi ces voisins. Cette méthode est simple mais sensible à la dimensionnalité, car elle repose fortement sur la notion de proximité locale.

Régression logistique. La régression logistique est un modèle linéaire discriminatif qui modélise la probabilité qu'un échantillon appartienne à une classe donnée à l'aide de la fonction sigmoïde :

$$P(y = 1 \mid \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x} - b)}.$$

Elle est rapide à entraîner, robuste et bien adaptée à la classification binaire. Toutefois, en l'absence de transformation non linéaire (comme le Kernel PCA), elle ne peut représenter que des frontières linéaires dans l'espace vectoriel.

Support Vector Machine (SVM). Les SVM cherchent à séparer les classes en maximisant la marge entre les exemples positifs et négatifs. Lorsqu'une séparation linéaire n'est pas possible, un noyau permet de projeter les données dans un espace de dimension plus élevée où une telle séparation devient envisageable. Nous utilisons ici le noyau RBF, défini par :

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\right),$$

avec γ un hyperparamètre à régler. Ce noyau a été choisi car il permet d'introduire une non-linéarité dans la frontière de décision sans nécessiter de transformation explicite des données, et il est reconnu pour ses bonnes performances dans de nombreuses tâches de classification, notamment textuelles [12]. Les SVM sont reconnus pour leur efficacité, mais leur temps de calcul peut devenir élevé pour de grandes dimensions ou de grands volumes de données.

4.2 Mise en œuvre des modèles

Les modèles ont été entraînés selon deux configurations distinctes :

1. Sur les données vectorisées (Bag-of-Words) sans réduction de dimension.
2. Sur les données réduites par *Kernel PCA* (noyau cosinus, 2 429 composantes principales).

Dans chaque cas, les performances sont évaluées sur l'ensemble de test à l'aide de deux métriques standards en classification binaire :

Accuracy (taux de bonne classification).

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

où TP, TN, FP, FN désignent respectivement les vrais positifs, vrais négatifs, faux positifs et faux négatifs.

F1-score (moyenne harmonique entre précision et rappel).

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Cette mesure est particulièrement adaptée en présence de classes déséquilibrées, car elle équilibre les erreurs de type I et de type II.

4.3 Résultats comparés

Le tableau suivant présente les performances obtenues par chaque modèle, avec et sans réduction de dimension :

Modèle	Accuracy	F1-score
KNN (sans PCA)	0.6338	0.5623
KNN (avec PCA)	0.7072	0.6885
LogReg (sans PCA)	0.8653	0.8651
LogReg (avec PCA)	0.8710	0.8706
SVM (sans PCA)	0.8675	0.8691
SVM (avec PCA)	0.8737	0.8727

TABLE 2 – Performances des modèles avec et sans réduction de dimension par Kernel PCA

On observe que l'application du Kernel PCA permet une amélioration systématique des performances pour les trois modèles évalués. L'effet est particulièrement marqué pour le KNN, qui est connu pour être très sensible à la dimensionnalité. Cette amélioration justifie empiriquement le recours à une réduction de dimension adaptée avant modélisation.

4.4 Comparaison visuelle

La figure suivante résume graphiquement les résultats obtenus :

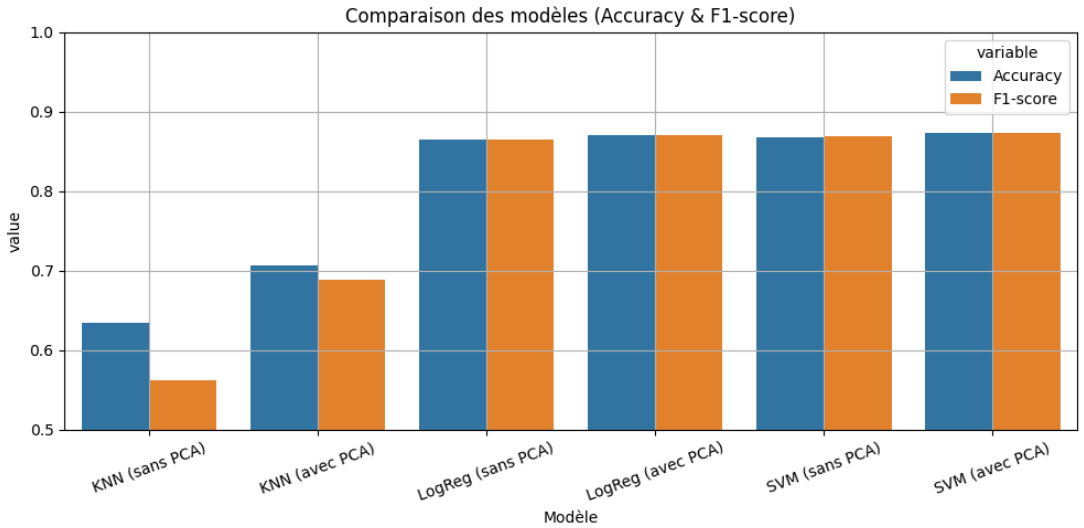

FIGURE 3 – Comparaison des performances des modèles (avec et sans Kernel PCA)

4.5 Courbes ROC

Enfin, nous avons également tracé les courbes ROC (Receiver Operating Characteristic) pour chacun des modèles, avec et sans PCA, afin de comparer la qualité de la séparation entre classes. Ces courbes permettent d'analyser le compromis entre taux de vrais positifs (sensibilité) et taux de faux positifs selon différents seuils de décision.
Les quantités tracées sont les suivantes :

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad \text{(taux de vrais positifs)} \qquad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \quad \text{(taux de faux positifs)}$$

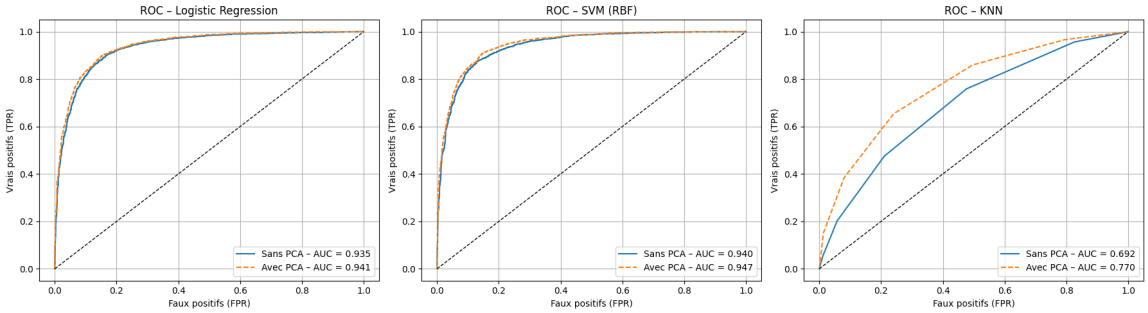

FIGURE 4 – Courbes ROC des modèles évalués (avec et sans Kernel PCA)

Les aires sous les courbes (AUC) confirment les résultats observés précédemment : le Kernel PCA améliore la capacité des modèles à discriminer correctement les deux classes, en leur fournissant une représentation plus informative et adaptée à la tâche de classification.

5 Analyse de sensibilité au nombre de composantes principales

La réduction de dimension par Kernel PCA nécessite le choix d'un nombre de composantes principales k à conserver. Ce paramètre joue un rôle central dans le compromis entre fidélité de la représentation et simplicité du modèle. Pour évaluer l'impact de ce choix, nous avons entraîné et évalué les trois modèles précédemment introduits (KNN, régression logistique, SVM) sur des versions des données réduites par Kernel PCA avec différentes valeurs de k : 50, 100, 200, 300, 500, 700, 1000, 1300, 1600, 1800, 2000, 2429 (valeur par défaut), 3000, 4000 et 5000.

5.1 Résultats expérimentaux

Le tableau suivant présente les scores d'accuracy obtenus pour chaque modèle en fonction du nombre de composantes principales conservées :

Modèle	50	100	200	500	1000	1300	1600	2000	2429	5000	Sans PCA
KNN	0.6707	0.6828	0.6822	0.6512	0.6833	0.6895	0.6848	0.7002	**0.7093**	0.7080	0.6492
LogReg	0.7820	0.8213	0.8388	0.8560	0.8610	0.8627	0.8623	0.8632	**0.8647**	0.8645	0.8607
SVM	0.7808	0.8178	0.8472	0.8607	0.8680	0.8702	0.8695	0.8713	**0.8702**	0.8725	0.8607

TABLE 3 – Accuracy des modèles en fonction du nombre de composantes principales (Kernel PCA), avec comparaison au cas sans réduction

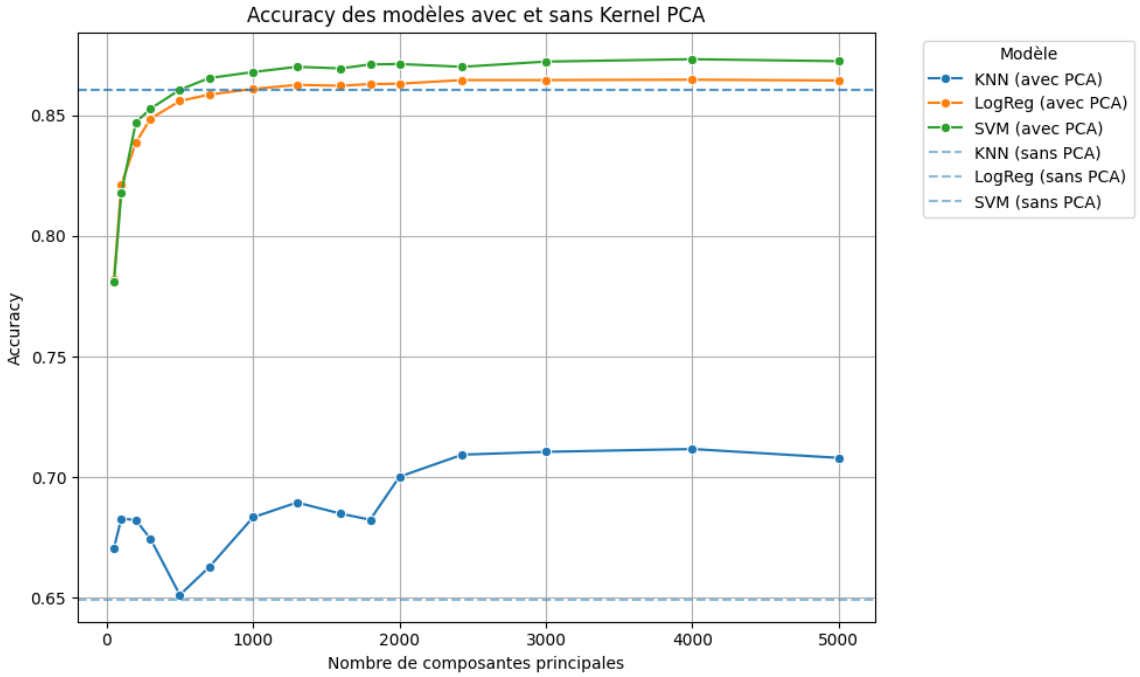

FIGURE 5 – Évolution de l'accuracy des modèles selon le nombre de composantes principales

Le tableau suivant présente les F1-scores correspondants, calculés sur l'ensemble de test pour chaque configuration :

Modèle	50	100	200	500	1000	1300	1600	2000	2429	5000	Sans PCA
KNN	0.6712	0.6796	0.6706	0.5830	0.6527	0.6596	0.6368	0.6765	**0.6869**	0.6947	0.6107
LogReg	0.7819	0.8205	0.8384	0.8552	0.8604	0.8622	0.8616	0.8625	**0.8639**	0.8640	0.8597
SVM	0.7808	0.8180	0.8467	0.8601	0.8675	0.8696	0.8688	0.8706	**0.8693**	0.8725	0.8625

TABLE 4 – F1-scores des modèles en fonction du nombre de composantes principales (Kernel PCA), avec comparaison au cas sans réduction

5.2 Interprétation

On observe que les performances augmentent globalement avec le nombre de composantes, en particulier pour le KNN dont l'accuracy passe de 67 % à plus de 71 % avec l'ajout progressif de composantes. Pour la régression logistique et le SVM, les performances s'améliorent jusqu'à environ 2000–2500 composantes, où elles se stabilisent. Ce comportement valide empiriquement le seuil basé sur la variance cumulée à 90 %, fixé à 2429 composantes.
L'ajout de composantes au-delà de 3000 n'apporte qu'un gain marginal, voire nul, tout en augmentant le coût computationnel. À l'inverse, des valeurs plus faibles comme $k = 500$ ou $k = 1000$ entraînent une perte notable de performance, en particulier pour le KNN. Cela confirme que la majorité de l'information discriminante est capturée dans un sous-espace de dimension intermédiaire (2000–2500).
Les F1-scores suivent une dynamique similaire à celle de l'accuracy, ce qui atteste de la cohérence des modèles en termes d'équilibre entre précision et rappel. On observe notamment une nette amélioration pour tous les modèles après 1000 composantes, avec des performances optimales atteintes entre 2400 et 3000 dimensions.

6 Conclusion de la phase expérimentale sur l'analyse de snetiment

Les expériences menées dans cette étude montrent que l'introduction d'un prétraitement linguistique rigoureux, combinée à une vectorisation adaptée (BoW), permet d'obtenir des représentations textuelles exploitables par des modèles classiques de machine learning. Toutefois, la dimension très élevée de ces représentations limite leur expressivité et ralentit les calculs.
L'application du Kernel PCA, avec un noyau cosinus spécialement adapté aux données textuelles, permet une réduction efficace de la dimension, tout en préservant les informations pertinentes pour la classification.

Les résultats obtenus montrent une amélioration systématique des performances sur les trois modèles testés (KNN, LogReg, SVM), aussi bien en termes d'accuracy que de F1-score.
L'analyse de sensibilité réalisée sur le nombre de composantes confirme qu'un sous-espace de dimension réduite (environ 2 400 composantes) suffit à capturer l'essentiel de l'information, rendant possible une compression efficace du signal sans perte notable de performance.
Cette phase expérimentale valide ainsi l'intérêt du Kernel PCA comme méthode de réduction de dimension dans le cadre de l'analyse de sentiment, en particulier lorsque les représentations initiales sont issues d'une vectorisation BoW classique. Elle ouvre également la voie à des explorations futures, incluant notamment l'usage d'autres types de noyaux, ou l'intégration dans des pipelines plus complexes (word embeddings, modèles neuronaux, etc.).

Références

[1] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics : Human Language Technologies* (pp. 142–150).

[2] Zipf, G. K. (1949). *Human behavior and the principle of least effort*. Addison-Wesley.

[3] Schmidt, A., & Wiegand, M. (2017). A Survey on Hate Speech Detection using Natural Language Processing. In *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media*.

[4] Harris, Z. S. (1954). Distributional structure. *Word*, 10(2–3), 146–162.

[5] Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

[6] Turney, P. D., & Pantel, P. (2010). From frequency to meaning : Vector space models of semantics. *Journal of Artificial Intelligence Research*, 37, 141–188.

[7] Aggarwal, C. C., & Zhai, C. (2012). *Mining Text Data*. Springer.

[8] Huang, A. (2008). Similarity measures for text document clustering. *Proceedings of the 6th New Zealand Computer Science Research Student Conference*.

[9] Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press.

[10] Mika, S., Schölkopf, B., Smola, A. J., Müller, K.-R., Scholz, M., & Rätsch, G. (1999). Kernel PCA and De-Noising in Feature Spaces. *Advances in Neural Information Processing Systems*.

[11] Jolliffe, I. T., & Cadima, J. (2016). Principal component analysis : a review and recent developments. *Philosophical Transactions of the Royal Society A : Mathematical, Physical and Engineering Sciences*, 374(2065).

[12] Joachims, T. (1998). *Text categorization with support vector machines : Learning with many relevant features*. In *Proceedings of the 10th European Conference on Machine Learning (ECML)*, pp. 137–142. Springer.

[13] Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media Inc.

[14] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn : Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.