

Datamining et IA

Étude de cas

Segmentation stratégique "Retail"

Objectifs

- Maîtriser le pipeline complet de Data Mining : du nettoyage à l'interprétation.
- Comprendre et implémenter la réduction de dimensionnalité.
- Comparer des approches de clustering (centré vs densité).
- Développer un esprit critique sur la validité statistique vs l'utilité métier.

Partie 1 : Les Clients

1. Prétraitement et ingénierie RFM

Le jeu de données fourni (online-retail) contient des transactions brutes. Votre premier défi est de passer d'une vision "transaction" à une vision "client".

L'approche RFM

Pour chaque client unique, vous devez calculer trois métriques :

- **Récence (R)** : Nombre de jours entre la date la plus récente du dataset et la dernière commande du client.
- **Fréquence (F)** : Nombre total de transactions (factures uniques).
- **Montant (M)** : Somme cumulée du chiffre d'affaires généré par le client.

Travail à réaliser

1. Nettoyez les données (annulations, prix unitaires à zéro, IDs manquants).
2. Calculez le score *RFM*.
3. **Analyse de la distribution** : Observez l'asymétrie (*skewness*) de vos variables. Pourquoi une simple standardisation suffit-elle rarement sur des données de montant ?

2. Réduction de dimension par PCA

Avant de segmenter, nous devons réduire le bruit et visualiser l'espace des données.

Qu'est-ce que l'Analyse en composantes principales (PCA) ?

La PCA vise à transformer des variables corrélées en nouvelles variables décorrélées appelées **Composantes Principales**. Mathématiquement, cela revient à changer de repère pour s'aligner sur les axes où les données sont le plus étalées (variance maximale).

Un exemple

Imaginons deux variables simplifiées pour deux clients : **Quantité (x)** et **Prix unitaire (y)**.

- Client A : (1, 1)
- Client B : (3, 3)

1. Centrage des données

On calcule la moyenne $\mu_x = 2$ et $\mu_y = 2$. On soustrait la moyenne à chaque point :

- $A' = (-1, -1)$
- $B' = (1, 1)$

2. La matrice de covariance (C)

Elle mesure comment les variables évoluent ensemble. Pour nos données centrées, elle se calcule ainsi :

$$C = \begin{pmatrix} \text{var}(x) & \text{cov}(x, y) \\ \text{cov}(y, x) & \text{var}(y) \end{pmatrix}$$

Dans notre cas simplifié (avec un dénominateur $n-1 = 1$), on obtient :

$$C = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

3. Recherche des valeurs propres (λ)

Les valeurs propres représentent la variance capturée par chaque nouvel axe. On les trouve en résolvant l'équation où le **déterminant** de la matrice ($C - \lambda I$) est nul :

$$\det \begin{pmatrix} 2 - \lambda & 2 \\ 2 & 2 - \lambda \end{pmatrix} = 0$$

$$(2 - \lambda)(2 - \lambda) - (2 \times 2) = 0$$

$$\lambda^2 - 4\lambda = 0 \Rightarrow \lambda_1 = 4, \lambda_2 = 0$$

Interprétation : La première composante capture 100 % de la variance ($\frac{4}{4-0}$), la seconde n'apporte rien. Les points sont parfaitement alignés sur une diagonale.

Mise en œuvre informatique

Dans la pratique, vous n'aurez pas à calculer de déterminants manuellement. Les bibliothèques optimisées utilisent des méthodes de décomposition comme la **SVD (Singular Value Decomposition)** pour traiter des matrices de milliers de lignes en quelques millisecondes.

Bibliothèques Python recommandées

Pour manipuler la PCA et les données, vous utiliserez principalement :

- **Scikit-Learn** (`sklearn.decomposition.PCA`) : La référence. Elle permet de définir le nombre de composantes ou le pourcentage de variance souhaité.
- **Pandas** : Indispensable pour la préparation initiale (méthodes `get_dummies`, `groupby`, `pivot_table`).
- **StandardScaler** (`sklearn.preprocessing`) : Crucial avant une PCA pour mettre toutes les variables à la même échelle (moyenne 0, écart-type 1).
- **Matplotlib / Seaborn** : Pour visualiser le "Scree plot" (graphique des valeurs propres) et les cercles de corrélation.

Travail à réaliser

1. **Pourquoi** une variable avec une variance immense (ex: le montant total en euros) écraserait-elle les autres variables (ex: le nombre de commandes) si on ne standardise pas les données avant la PCA ?
2. Si le déterminant de votre matrice de covariance est proche de zéro, qu'est-ce que cela indique sur la relation entre vos variables originales ?

3. Clustering algorithmique

Le **clustering** (ou partitionnement de données) est la tâche phare de l'**apprentissage non supervisé** (*unsupervised learning*). Contrairement à la classification où l'on cherche à prédire une étiquette déjà connue (ex: "est-ce un spam ?"), le clustering consiste à regrouper des objets de telle sorte que les objets d'un même groupe soient plus similaires entre eux qu'avec ceux des autres groupes.

Dans un contexte "Retail", le clustering permet de passer d'une masse informe de millions de transactions à une vision stratégique composée de quelques **segments de clientèle** cohérents.

Algorithme A : K-Means

Le K-Means est un algorithme itératif qui cherche à minimiser la somme des carrés au sein de chaque groupe (**Within-Cluster Sum of Squares - WCSS**) (somme des carrés des distances entre chaque point d'un cluster et son centroïde).

Pseudo-code :

1. Initialiser K centroïdes de manière aléatoire.
2. Assigner chaque point au centroïde le plus proche (distance euclidienne).
3. Recalculer la position de chaque centroïde (moyenne des points du cluster).
4. Répéter les étapes 2 et 3 jusqu'à convergence (les centroïdes ne bougent plus).

La méthode du coude (Elbow Method) :

Pour trouver le K optimal (nombre de clusters), on calcule la $WCSS$ pour différentes valeurs de K :

$$WCSS = \sum_{i=0}^n (x_i - c_K)^2$$

Il existe une corrélation naturelle entre K et le $WCSS$:

1. Si $K = 1$ (un seul groupe), le $WCSS$ est maximal.
2. Si $K = n$ (chaque point est son propre cluster), le $WCSS$ est égal à 0.

Cherchez le point d'inflexion sur le graphique $WCSS = f(K)$ où l'ajout d'un cluster n'apporte plus un gain de précision significatif.

Algorithme B : DBSCAN

Le **DBSCAN** (*Density-Based Spatial Clustering of Applications with Noise*) est une approche radicalement différente du K-Means. Alors que le K-Means cherche à partitionner les données en fonction de centres de gravité, le DBSCAN les regroupe en fonction de leur densité locale

1. Le concept fondamental : la densité locale

Au lieu de supposer que les clusters sont des "bulles" sphériques, le DBSCAN considère qu'un cluster est une zone continue de haute densité, séparée des autres par des zones de basse densité (du vide ou du bruit).

Les deux paramètres clés

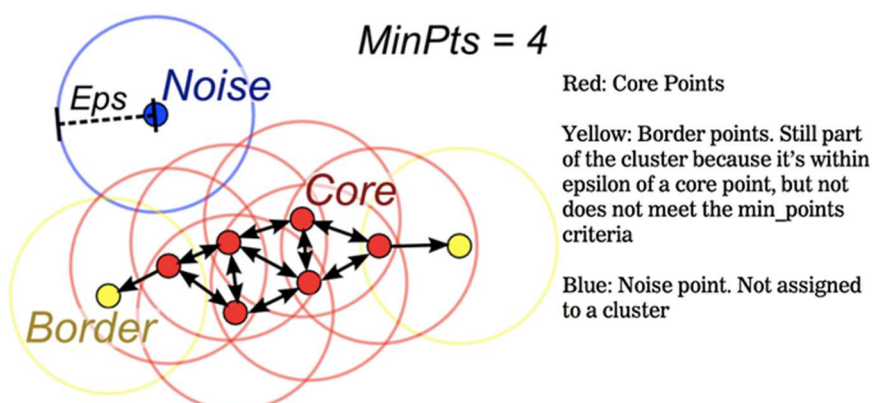
Pour définir cette densité, l'algorithme repose sur deux curseurs :

- **Epsilon (ϵ)** : C'est la distance maximale pour que deux points soient considérés comme "voisins". C'est le rayon de recherche autour d'un point.
- **MinSamples** : C'est le nombre minimum de points (voisins) qui doivent se trouver dans le rayon ϵ pour qu'une zone soit considérée comme "dense".

2. La classification des points

C'est ici que réside la force de l'algorithme. Chaque point du dataset reçoit un statut spécifique :

1. **Core point (Point central)** : Un point qui possède au moins MinSamples dans son voisinage de rayon ϵ .
2. **Border point (Point frontière)** : Un point qui a moins de MinSamples dans son voisinage, mais qui est situé dans le voisinage d'un *Core point*.
3. **Noise point (Point de bruit/Outlier)** : Un point qui n'est ni un *Core point*, ni un *Border point*. Ces points ne sont affectés à aucun cluster.



3. Pseudo-code

L'algorithme se propage par "contagion" de densité :

1. Pour chaque point P non visité dans le dataset :
2. Identifier tous les points dans le voisinage ϵ de P .
3. **SI** le nombre de points est $\geq \text{MinSamples}$:
 - Un nouveau cluster est créé.
 - On inclut tous les voisins dans ce cluster.
 - **Tant que** de nouveaux *Core points* sont ajoutés au cluster, on étend le voisinage de manière récursive (le cluster "grandit").
4. **SINON** :
 - Marquer P comme **Bruit** (attention : il pourra devenir un *Border point* plus tard si un autre cluster le rattrape).
5. Répéter jusqu'à ce que tous les points aient été traités.

4. Exemple concret : détection de fraude ou comportement atypique

Imaginons l'analyse des données *Online Retail* avec deux variables : le nombre de connexions par mois et le montant total dépensé.

- **Le groupe principal** : La majorité des clients se connecte 2 à 4 fois et dépense entre 50€ et 200€. Ils sont très proches les uns des autres dans l'espace des données. DBSCAN va identifier cette zone dense comme un cluster unique.
- **Les "Baleines" (Whales)** : Quelques clients dépensent 10 000€ mais ne se connectent qu'une fois. Ils sont isolés.
 - **K-Means** les forcerait à rejoindre un groupe ou créerait un groupe artificiel autour d'eux, déplaçant le centre de gravité.
 - **DBSCAN** les classera en **Bruit (Noise)**. Cela permet de nettoyer votre segmentation principale tout en isolant ces cas particuliers pour une analyse spécifique.

5. Pourquoi utiliser le DBSCAN ?

Avantages comparatifs :

- **Pas de K à définir** : L'algorithme trouve le nombre de clusters tout seul.
- **Formes arbitraires** : Il peut trouver des clusters en forme de croissant, de cercle ou de lignes, là où le K-Means échouerait.
- **Robustesse aux outliers** : C'est l'un des meilleurs outils pour la détection d'anomalies.

Le point de vigilance : Comment choisir ϵ ? Si ϵ est trop petit, tout est considéré comme du bruit. S'il est trop grand, tous les clusters fusionnent. Une technique courante (que vous pouvez chercher) est la **méthode du K-dist graph**.

Bibliothèques Python utiles

- **Scikit-Learn** (`sklearn.cluster.DBSCAN`) : La fonction principale.
 - *Argument clé* : **eps** et **min_samples**.
- **KneeLocator** (`kneed`) : Pour aider à détecter automatiquement le "coude" dans le graphique des distances.

4. Validation et interprétation métier

Un bon cluster mathématique n'est pas toujours un bon cluster marketing.

1. **Silhouette Score** : Calculez ce score pour valider la séparation de vos groupes. Un score proche de 1 indique que le point est très loin des clusters voisins.
2. **Radar Chart** : Visualisez les caractéristiques moyennes de chaque segment sur un graphique radar.
3. **Interprétation** :
 - Comment appelez-vous le groupe ayant un *R* faible, un *F* élevé et un *M* élevé ?
 - Quelle action marketing proposez-vous pour un groupe ayant un *R* élevé mais un *M* historiquement très fort ?

Livrables attendus

- Un rapport technique répondant aux questions, avec les résultats des algorithmes.
- Le code documenté (noms de fonctions et variables en anglais).
- Une présentation synthétique des "Personas" clients identifiés.

Partie 2 : les articles

Dans cette section, l'objectif n'est plus d'analyser le comportement des acheteurs, mais la structure même de l'offre commerciale. Nous allons chercher à regrouper les produits non pas par leur prix, mais par la **proximité sémantique** de leurs descriptions.

1. Préparation des données textuelles (NLP)

Le dataset contient une colonne Description. Avant d'utiliser un réseau de neurones, un nettoyage rigoureux est nécessaire.

- **Extraction** : Créez un nouveau DataFrame contenant uniquement les produits uniques (basé sur StockCode et Description).
- **Nettoyage** : Transformez les descriptions pour faciliter le traitement :
 - Passage en minuscules.
 - Suppression de la ponctuation et des caractères spéciaux.
 - Suppression des *stopwords* (mots fréquents sans valeur sémantique comme "the", "and", "a").
- **Vectorisation** : Les modèles d'IA ne lisent pas de texte brut. Utilisez la technique **TF-IDF** (*Term Frequency-Inverse Document Frequency*) pour transformer chaque description en un vecteur numérique.

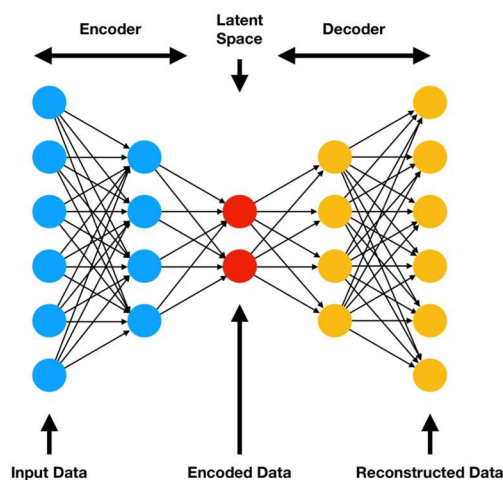
2. Architecture de l'autoencodeur avec TensorFlow

Pour réduire la dimensionnalité de vos vecteurs textuels tout en capturant des relations complexes, vous allez implémenter un **Autoencodeur**. C'est un réseau de neurones qui apprend à compresser les données puis à les reconstruire.

1 Structure du modèle

Le réseau doit être composé de deux parties symétriques :

1. **L'encodeur** : Il réduit la taille du vecteur d'entrée (ex: 500 mots) vers une couche centrale très étroite appelée **espace latent** (ex: 32 neurones).
2. **Le décodeur** : Il tente de reconstruire le vecteur original à partir de l'espace latent.



2 Fonction de perte (Loss function)

Le modèle s'entraîne en minimisant l'erreur de reconstruction. On utilise généralement l'erreur quadratique moyenne (*MSE*) :

$$MSE = \frac{1}{n} \sum_{i=1}^n (x^i - \hat{x}^i)^2$$

Où x est le vecteur original et \hat{x} est le vecteur reconstruit par le réseau.

3. Mise en œuvre sous TensorFlow / Keras

Vous devez concevoir le modèle en utilisant l'API Keras.

Consignes de développement :

- Utilisez des couches Dense avec des fonctions d'activation ReLU pour les couches cachées.
- La couche de sortie doit utiliser une activation Sigmoid (si vos données sont normalisées entre 0 et 1).

Travail à réaliser :

1. Entraînez le modèle sur vos vecteurs TF-IDF.
2. Une fois l'entraînement terminé, extrayez uniquement la partie **Encodeur** pour transformer chaque description de produit en un vecteur de l'espace latent (dimension 32).

4. Clustering dans l'espace latent

Maintenant que chaque produit est représenté par un vecteur "intelligent" de 32 nombres :

1. Appliquez l'algorithme **K-Means** sur ces nouveaux vecteurs.
2. Utilisez à nouveau la **méthode du coude** pour déterminer le nombre de catégories de produits optimal.
3. Visualisez les clusters en utilisant une réduction de dimension (PCA) pour voir si les groupes de produits sont bien séparés.

5. Interprétation et application métier

- **Analyse qualitative** : Prenez 5 produits au hasard dans un même cluster. Leurs descriptions semblent-elles traiter du même univers ?
- **Système de recommandation** : Expliquez comment ce modèle pourrait permettre de suggérer un produit "similaire" à un client sans avoir besoin de son historique d'achat passé.
- **Esprit critique** : Pourquoi le Deep Learning est-il ici plus puissant qu'un simple clustering sur les mots ? Quelles sont les limites de cette approche si les descriptions sont trop courtes (ex: "Blue Vase") ?

Livrables attendus :

- Le code de l'autoencodeur documenté.
- Une visualisation (Scatter plot) des clusters de produits obtenus.