Réduction de dimension

Plan

Introduction

- Manipulation et traitement de larges volumes de données
- *Exemple* : avis des utilisateurs sur Netflix : $\approx 10^8$ utilisateurs, $\approx 10^4$ films.
- Nécessité de *réduire les dimensions* d'un nuage de points
- Étant donné un nuage de m points $x_1, \ldots, x_m$ dans $\mathbb{R}^n$, avec m grand, comment résumer l'information contenue dans cet ensemble de points ?
- Approche inscrite dans la continuité des cours d'ADD et de statistique : factorisations matricielles

Factorisation de matrices

- $x_1, \ldots, x_m$ m points de $\mathbb{R}^n$ (m individus, n variables)
- $m \gg n$: grand nombre d'individus
- Nuage représenté par

$$X = \begin{pmatrix} - & x_1 & - \\ & \vdots & \\ - & x_m & - \end{pmatrix} \quad \text{taille } m \times n$$

- **Objectif** : Écrire $\underset{(n\times m)}{X^T} \approx \underset{(n\times k)}{W} \cdot \underset{(k\times m)}{H}$ où $k \ll n \ll m$
- Gain : $n \cdot k + k \cdot m = (n+m)k \ll nm$

L'Analyse en Composantes Principales

- Première factorisation : donnée par l'Analyse en Composantes Principales (ACP)
- **Idée** : pour k prescrit, chercher $v_1, \ldots, v_k$ dans $\mathbb{R}^n$ orthonormés tels que chaque x_i s'écrit de façon approchée comme combinaison linéaire de $v_1, \ldots, v_k$
- Équivalent à rechercher les vecteurs propres associés aux plus grandes valeurs propres de la matrice symétrique semi-définie positive $X^T X$
- **Objectif** : implémenter la recherche des "plus grands" vecteurs propres sans recourir à Numpy ou Scipy
- **Cheminement** : étude d'un algorithme naïf (méthode des puissances) puis étude d'un algorithme robuste (QR)

Méthode des puissances

- Méthode itérative qui converge vers un vecteur propre associé à la plus grande valeur propre
- Très simple à implémenter
- Mais converge lentement et numériquement instable (pour nos tests, conduit fréquemment à des overflows)
- Utilisée par Google PageRank : adaptée à un certain type de matrices ?

Algorithme QR

- Matrice tridiagonale symétrique : $\begin{pmatrix} a_1 & b_1 & & & \\ b_1 & a_2 & b_2 & & \\ & b_2 & \ddots & \ddots & \\ & & \ddots & \ddots & b_{n-1} \\ & & & b_{n-1} & a_n \end{pmatrix}$
- **Idée** : Symétrique → Tridiagonale symétrique → Diagonale
- Différence majeure : calcule tous les vecteurs propres (et toutes les valeurs propres)

Algorithme QR (commentaires)

- Numériquement très stable, converge plus rapidement que la méthode des puissances, converge toujours pourvu que le nombre d'itérations soit suffisant
- Bien plus lent que l'algorithme utilisé par Scipy ($\approx$ minutes vs instantané)
- *Commentaires* : Implémentation des différentes sous-fonctions a demandé le plus de travail, difficultés dans le passage du pseudo-code à un code sans Numpy (cas de bord, manipulation de matrices blocs)
- *Application* : compression d'une image

La Non-Negative Matrix Factorization

- **Objectif** : Écrire $\underset{(n\times m)}{X^T} \approx \underset{(n\times k)}{W} \cdot \underset{(k\times m)}{H}$ où les coefficients de X, W et H sont ≥ 0
- **Philosophie** : Colonnes de X^T (individus) obtenues comme combinaisons linéaires des colonnes de W (peu nombreuses). Positivité des coefficients $\implies$ phénomène de superposition (au sens d'addition)
- Colonnes de W font apparaître des features latents du nuage de données

Implémentation

- Problème qui n'est pas linéaire, ni convexe. Convergence vers des minima locaux (pas forcément globaux) : suivant les matrices de départ, deux exécutions peuvent aboutir à des factorisations différentes
- Pas d'implémentation naïve, absence de monographes sur le sujet $\Longrightarrow$ difficile de trouver des implémentations accessibles (hors articles de recherche)
- Implémentation disponible disponible dans `sklearn.decomposition`

Application

- **Objectif** : répliquer les résultats de l'article fondateur sur la NMF (1999)
- $3 \cdot 10^4$ articles d'encyclopédie et lexique des 10^4 mots anglais les plus courants (hors trivialités). Pour un mot donné, compter les occurrences dans chaque article. X^T est 15276×30991.
- L'exécution de la NMF est longue (≈ 40 minutes pour $k = 400$). Les colonnes de W regroupent les mots par thème :
  - `cities, urban, city, towns, planning,...`
  - `government, majority, section, supreme, court, representatives,...`

Mesure de la qualité de la factorisation

- Définir une norme sur l'espace des matrices afin de quantifier $\|X^T - WH\|$
- Pour l'ACP, choix naturel de la norme de Frobenius (dérive du produit scalaire $\mathrm{tr}(A^T B)$). Assure la convergence pour chaque coefficient de X^T
- Pour la NMF, moins clair. Dans l'article, utilisation de la divergence de Kullback-Leibler. Il n'y a plus convergence vers les coefficients de la matrice de départ : les 0 sont remplacés par des valeurs "devinées" (utilisé par Netflix)

Conclusion

- Résumer de l'information en factorisant des matrices selon deux points de vue : ACP et NMF
- Progrès en programmation (code structuré et commenté), en algèbre linéaire et en analyse numérique
- Gain d'autonomie (lectures de ressources et d'articles de recherche en anglais)