UNIVERSITÉ PARIS DESCARTES | USPC Université Sorbonne Paris Cité

DATA MINING

-

Rapport de projet

Enseignants :

Mme Séverine Affeldt

Mr Lazhar Labiod

Étudiants :

LARBI Abderrahmane

Numéro étudiant : 21810919

DAHI Mohammed

Numéro étudiant : 21811772

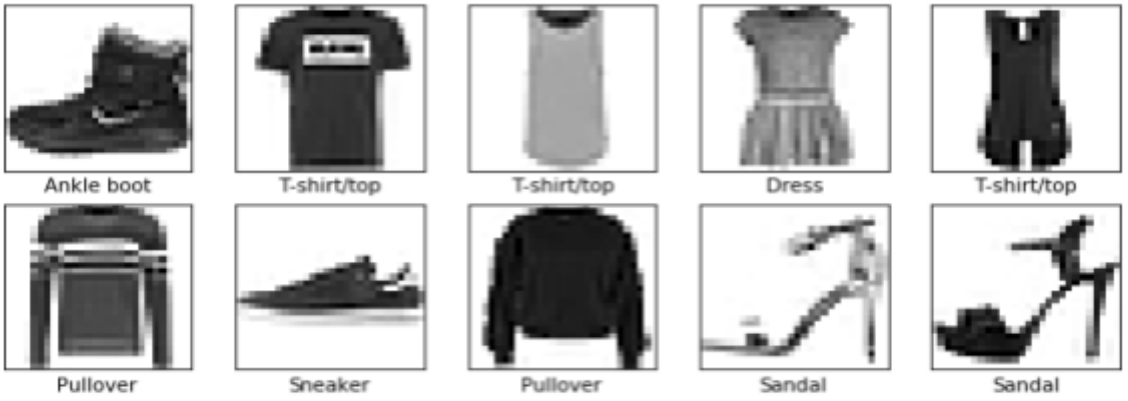
LACARNE Mohammed

Numéro étudiant : 21811224

Présentation du dataset Fashion Mnist:

Le dataset Mnist des chiffre en écriture manuscrite est l'un des datasets les plus utilisés pour l'exploration des réseaux de neurones et est devenu une référence pour tester la précision des modèles. Récemment, Zalando research a publié un nouveau dataset avec 10 différents produits de mode. Appelé fashion Mnist , ce dataset a pour but le remplacement du MNIST originale qui est devenu facile pour les modèles de machine learning.
Le dataset fashion MNIST se compose de 70,000 images en noir et blanc dans 10 catégories. Ces images montrent des articles de vêtements à basse résolution (28x28px).

L'image ci-dessous montre un exemple de 10 images tirées du dataset aléatoirement:



Aperçu du jeu de données fashion-mnist

Notre objectif est d'explorer, visualiser et classifier le jeu de données Fashion MNIST, pour cela voici les différentes méthodes que nous avons appliquer :

1. Réduction de dimension pour Fashion-MNIST avec une PCA
2. Réduction de dimension pour Fashion-MNIST avec t-SNE
3. Réduction de dimension pour Fashion-MNIST avec deep autoencoder
4. Clustering dans un espace latent de petite dimension
5. Clustering avec SVM
6. Clustering avec LDA

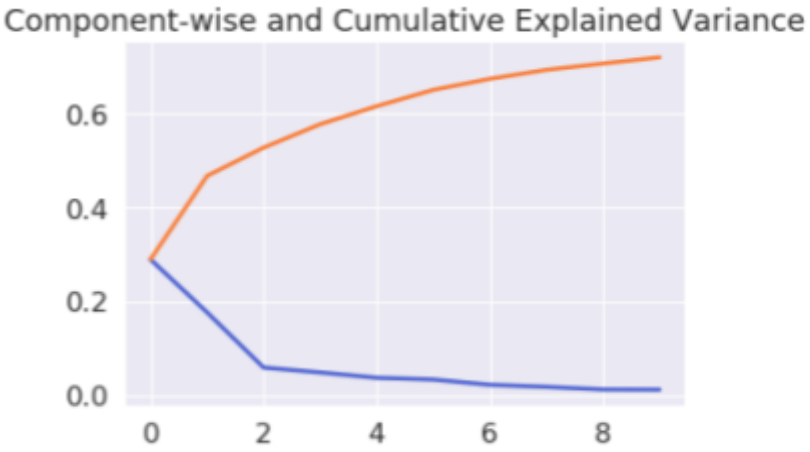
Expérimentation:

Réduction de dimension pour Fashion-MNIST avec une PCA :

L'analyse en composante principale (PCA est l'exemple le plus simple de réduction de dimensionnalité qui consiste à prendre une matrices avec plusieurs observations et la compressée en

une matrice avec moins d'observations tout en préservant le plus possible l'information contenu dans la matrice de départ. Dans le cadre de notre expérience, nous avons appliquer une ACP en utilisant la bibliothèque sklearn sous python sur l'ensemble du jeu de données en choisissant n = 4.

Le graphe ci-dessous montre le pourcentage de variance expliqué par chaque composante principale en bleu ainsi que la variance cumulative en rouge.



Courbe de la variance cumulative et expliquer pour chaque composantes.

On remarque que les deux premières composantes principales expliquent à elles seule 46% de la variance du jeu de données, ce qui reste assez faible. Les composantes principales trois et quatre n'apportent pas un impact considérable puisque la courbe cumulative de la variance expliqué est de 60% pour ces quatre premières composantes combinées.

La figure suivante montre la visualisation du jeu de données pour chaque combinaisons des quatres premières composantes principales :

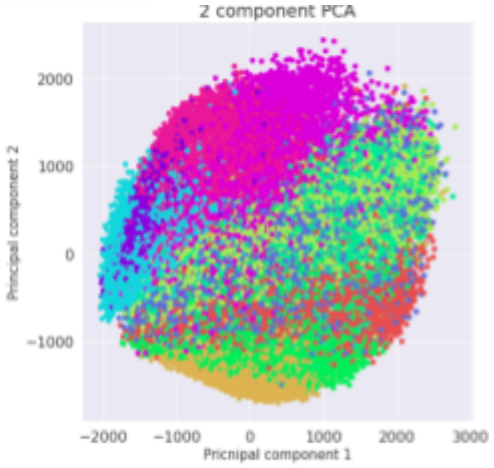


Figure A

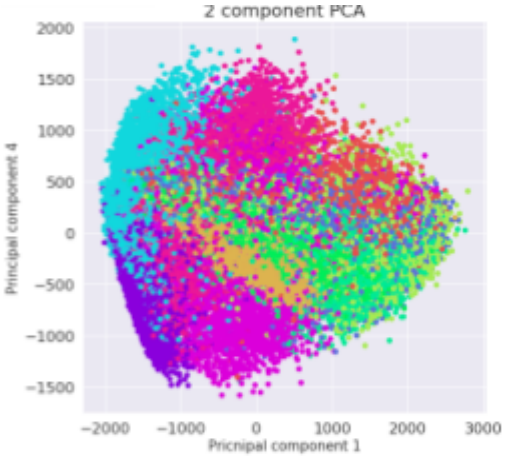


Figure B

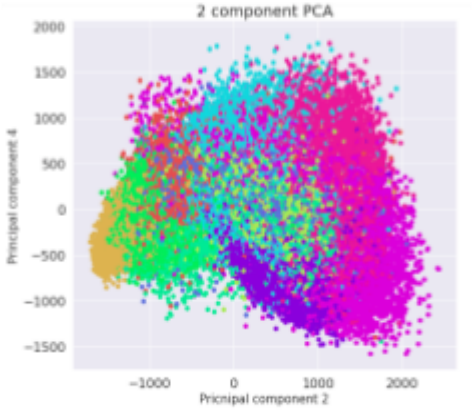


Figure C

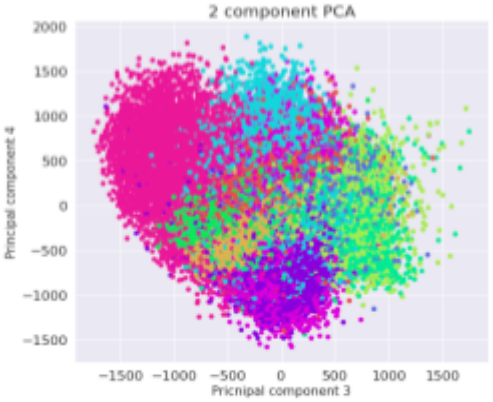


Figure D

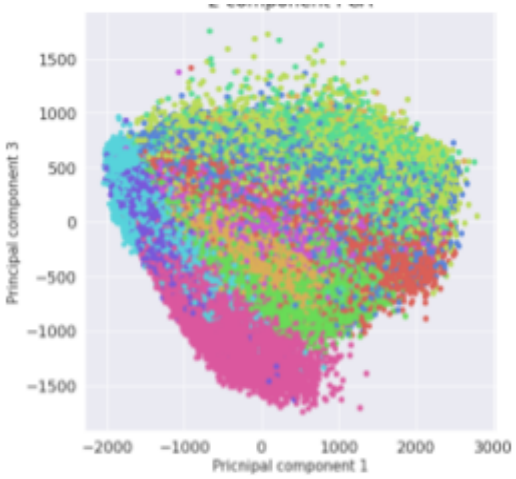


Figure E

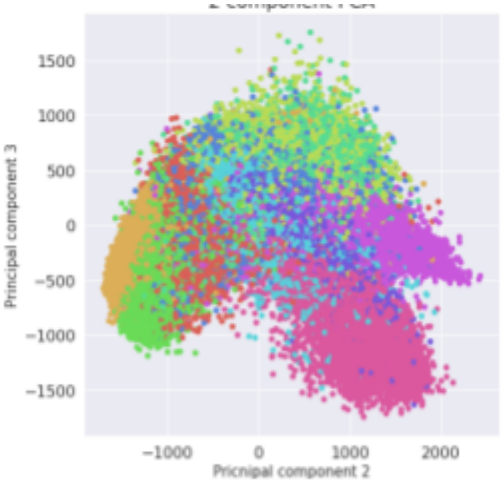


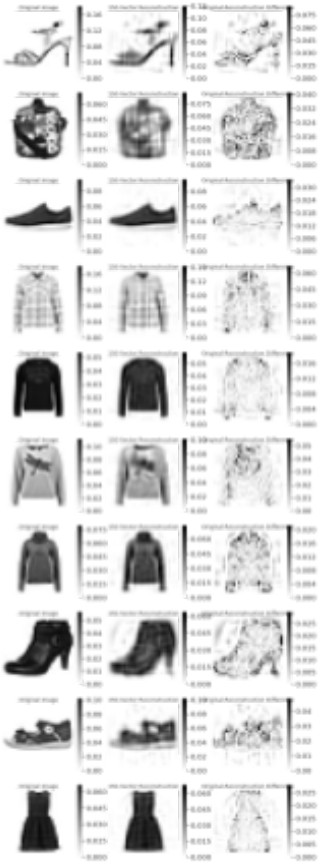
Figure F

Même si le jeu de données est supervisé et que toutes les images appartiennent à des classes données au départ, on constate que la visualisation de ces données avec une sur les différentes composantes n'arrive pas à séparer le jeu de données en des clusters bien défini, Cela est dû à la faible explicabilité de la variance par les composantes. Le graphe de la figure A est le plus compacte

des six car les points d’une même classe sont moins éparpillées par rapport aux autres graphes, ce constat est logique car nous utilisons la PCA1 et PCA2 pour la visualisation. Cependant, elles n’arrivent pas à définir des frontières entre les différents clusters, d'où la visualisation sous une forme de boule.

L’éparpillement des points pour les autres graphes est beaucoup trop important et aucuns d’eux n’arrivent à séparer clairement les points en clusters. Notre conclusion est que la pca n’arrive pas a faire une séparation de nos points en cluster, ceci est dû au faite que cette méthode est une technique de réduction de dimensionnalité linéaire qui ne convient pas à la nature complexe du jeu de données fashion-MNIST, cependant il serait intéressant d’inclure la pca1 et pca2 comme attributs dans le jeu de données pour faire de la classification.

Afin d’estimer la qualité de reconstruction des images a l’aide des composantes principales de la pca, nous appliquons cette fois ci une pca avec n=150. Nous choisissons ensuite 10 exemples de chaque classe du jeu de données, la figure ci-dessous montre pour chaque exemple son image originale, sa reconstruction en utilisant les composantes de la pca et la différence entre les deux :

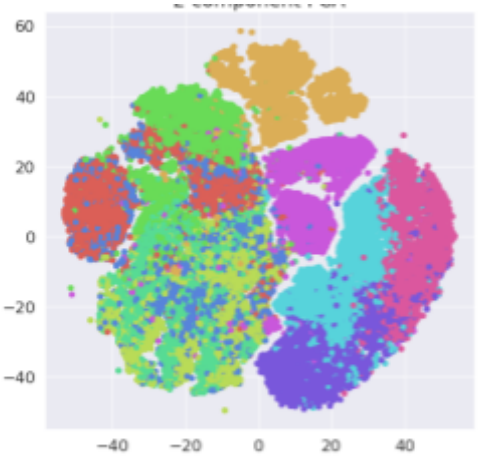


Reconstruction des images avec 150 composantes de l’ACP

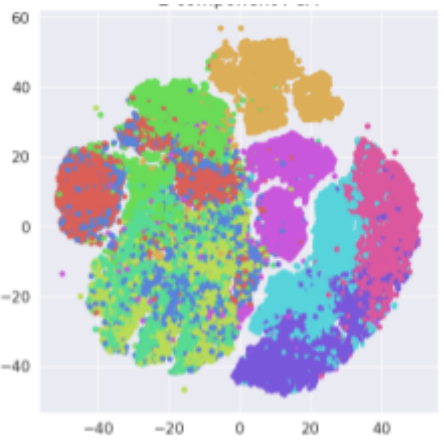
Réduction de dimension pour Fashion-MNIST avec t-SNE :

On s'intéresse maintenant a une technique de réduction dimensionnelle qui est plus adaptée à notre jeu de données. L'algorithme t-SNE (**t-distributed stochastic neighbor embedding**) est une technique de réduction de dimension pour la visualisation de données développée par Geoffrey Hinton et Laurens van der Maaten. Il s'agit d'une méthode non-linéaire permettant de représenter un ensemble de points d'un espace à grande dimension dans un espace de deux ou trois dimensions, les données peuvent ensuite être visualisées avec un nuage de points. Nous avons appliqué une T-SNE en laissant les paramètres par défaut à l'exception du paramètre perplexity que l'on a fait varier.

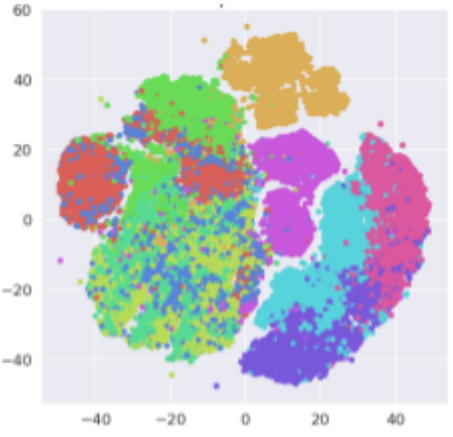
Les figures ci-dessous représente les nuages de points des différentes exécutions de la T-SNE pour différentes valeurs du paramètre perplexity :



Perplexity = 30 (default)



Perplexity = 40



Perplexity = 50

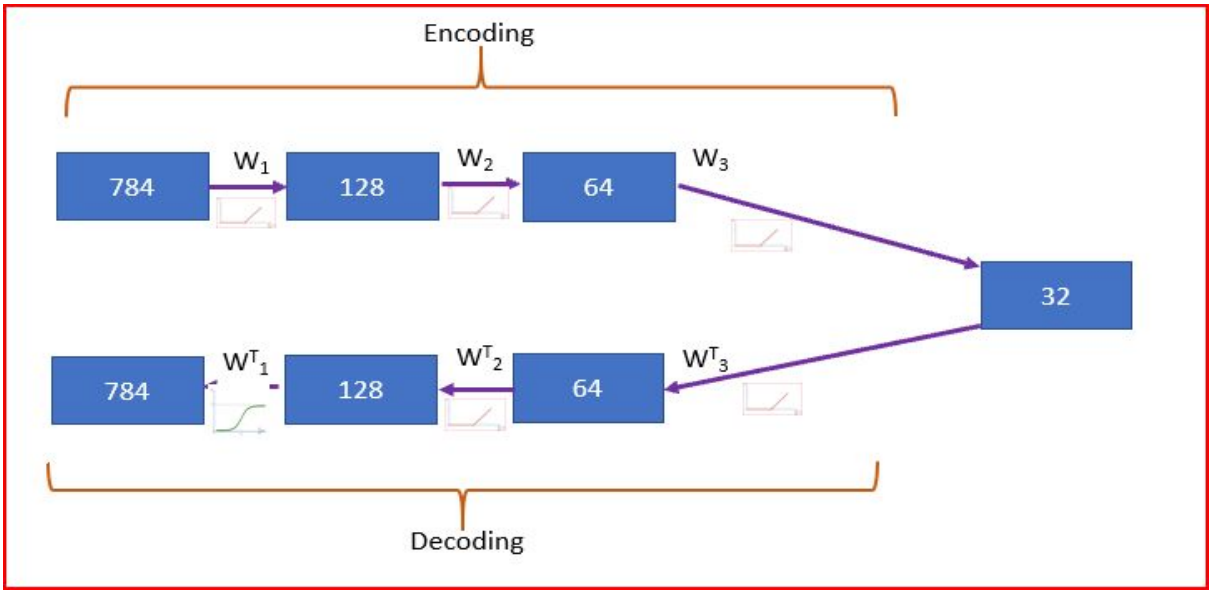
Le clustering de la t-NSE est bien meilleure par rapport à celle de la pca, ceci est dû au fait qu'elle soit une technique de réduction de dimension non linéaire. On remarque que le clustering est

légèrement meilleur lorsque l'on a augmenter la valeur du paramètre perplexity, l'intuition derrière ce paramètre et qu'il aide l'algorithme à trouver le nombre des plus proches voisins pour chaques points. En effet pour perplexity = 40 et perplexity = 50 on remarque un meilleur regroupement pour les clusters, notamment pour le cluster orange qui devient de plus en plus compacte.

Réduction de dimension pour Fashion-MNIST avec deep autoencoder

Les auto-encodeurs sont un type spécifique de feedforward neural networks où l'entrée est identique à la sortie. Ils compressent l'entrée dans pour obtenir un code de dimension inférieure, puis reconstruisent la sortie à partir de cette représentation. Le code est une "compression" compact de l'entrée, également appelé représentation en espace latent.

Un autoencoder est fait à partir de 3 composants: encodeur, code et décodeur " comme le montre la figure en bas". Le codeur compresse l'entrée et produit le code, le décodeur reconstruit ensuite l'entrée uniquement à l'aide de ce code.



Les autoencodeurs sont principalement des algorithmes de réduction de dimensionnalité (ou compression).

Architecture :

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 784)	0
dense_1 (Dense)	(None, 128)	100480
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 32)	2080

```
dense_4 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_5 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_6 (Dense)              (None, 784)               101136
=================================================================
```

Partie codeur:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input 1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
=================================================================
```

Partie décodeur:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input 2 (2nd InputLayer)     (None, 32)                0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_5 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_6 (Dense)              (None, 784)               101136
=================================================================
```

On a opté pour une architecture de 5 hidden layers avec l'input layer et l'output layer.

Model design:.

- **nombre de noeuds**

- input layer avec 784 noeuds.
- un premier dense layer de 128 noeuds.
- un second dense layer de 64 noeuds.
- un troisième dense layer (output pour le codeur) de 32 noeuds.
- on a la partie décodeur qui est la symétrie de ce qu'on a fait. des layers 64 , 128 puis 784.

- **Fonction d'activation**

- Pour les layers 1 jusqu'au 5 on a utilisé la fonction d'activation ReLU
- Pour la sortie ou on a utilisé la fonction sigmoid.

- **Loss**

- Pour le loss on a utilisé la binary_crossentropy car on a normalisé notre canale /255.

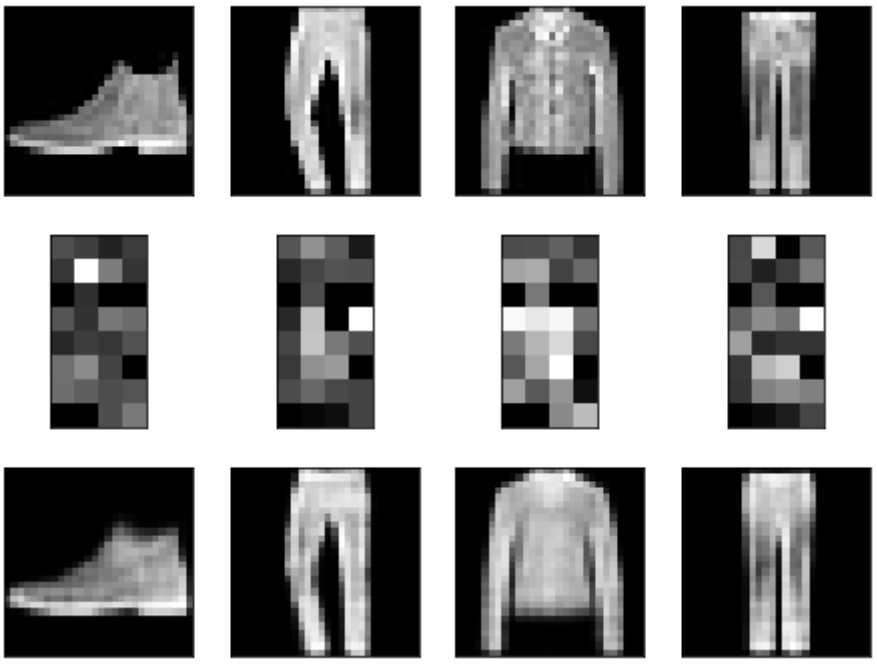
- **Optimizer**

- Pour l'optimizer on a utilisé ADAM optimizer.

Hyperparamètres

- **Learning rate :** pour le learning rate on a utilisé deux learning rate , lr=0.001 , lr= 0.01.
- **Batch size :** on a utilsé un batch de 64.

Images avant et après encodage :

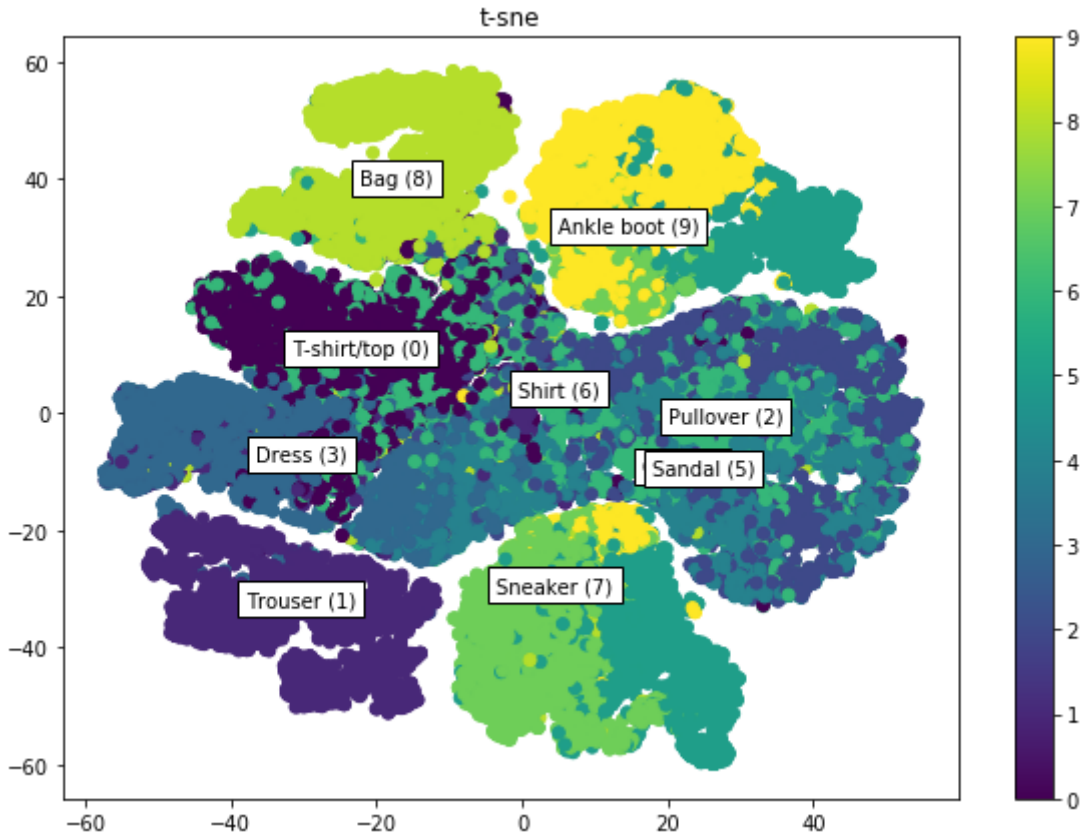


Clustering dans un espace latent de petite dimension

Après l'encodage des donnés on a appliqué sur les donnés réduites l'algorithme K-means avec 10 clusters. la performance de ce dernier était de 53% de précision.

Pour la visualisation 2D en particulier, le t-SNE est probablement le meilleur algorithme, mais il nécessite généralement des données de dimension relativement réduite. Donc, une bonne stratégie pour visualiser les relations de similarité dans les données de grande dimension consiste à commencer par utiliser un auto-codeur pour compresser vos données dans un espace à basse dimension (par exemple, 32 dimensions), puis à utiliser t-SNE pour mapper les données compressées dans un plan 2D.

Visualisation du t-sne



On remarque que:

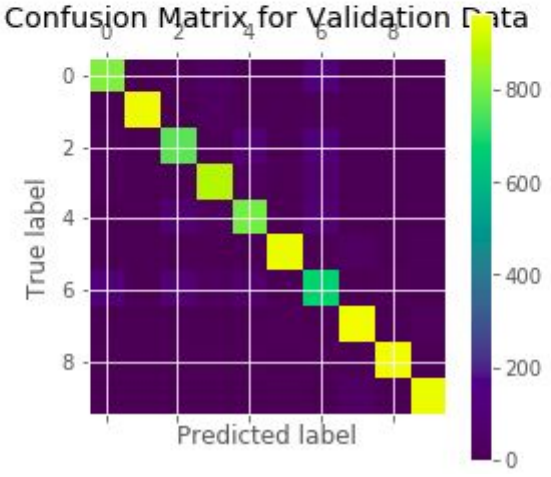
- Les clusters 2 4 6 0 3 qui représentes les hauts sont regroupés.
- Les clusters 7 9 qui représentent des chaussures sont séparé des autres.
- Le cluster 8 (bag) est tout seule et le cluster 1 (trouser) est seule aussi.
- le cluster 5 (sandal) a été confondu avec les cluster qui représentes des hauts.

SVM :
Afin d'appliquer l'algorithme de classification SVM sur le dataset Fashion mnist de Keras, on a utilisé plusieurs fonction de transition "KERNEL" comme LINEAR, POLY ou encore RBF.
on a décidé de laissé le cout C à 1 en pensant que ce dernier n'a pas de grande influence, et aussi on risque de tomber sur un overfitting ou bien sur un underfitting.

Resultat pour les 3 fonctions kernel :

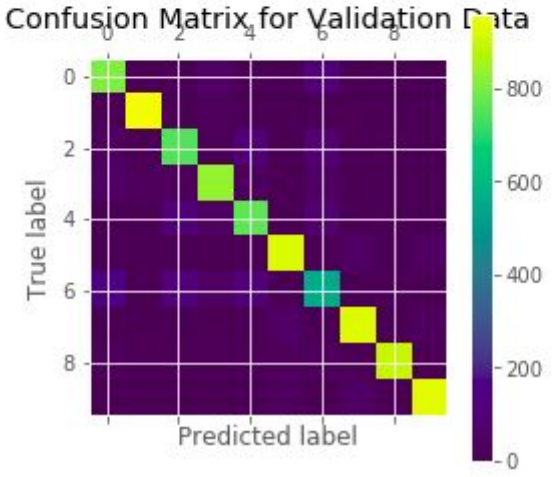
Kernel	Accuracy
Linear	0.8405
Polynomial (degree=3)	0.8755
RBF	0.8711

```
Confusion Matrix:
 [[819   1   8  40   2   1 123   0   6   0]
 [  1 956   2  30   2   0   9   0   0   0]
 [ 17   0 768   9  92   0 113   0   1   0]
 [ 15   2   8 883  27   0  64   0   1   0]
 [  1   0  76  29 804   0  89   0   1   0]
 [  0   0   0   0   0 952   4  31   2  11]
 [116   3  89  29  60   0 690   0  13   0]
 [  0   0   0   0   0  12   0 967   0  21]
 [  4   0   4   7   4   2  14   4 961   0]
 [  0   0   0   0   0   7   2  36   0 955]]
```



-Matrice de confusion pour la fonction polynomial comme kernel-

```
Confusion Matrix:
 [[810   5  11  51   5   1 111   0   6   0]
 [  8 961   4  21   2   0   2   0   2   0]
 [ 32   7 752   8 108   0  87   0   6   0]
 [ 47  22  25 834  27   0  41   0   3   1]
 [  1   3 115  40 766   0  71   0   4   0]
 [  0   0   0   0   0 924   0  39   3  34]
 [165   2 120  36 105   0 559   0  13   0]
 [  0   0   0   0   0  43   0 929   1  27]
 [ 18   1  14  12   8  17  28   4 898   0]
 [  0   0   0   0   0  27   1  36   0 936]]
```



-Matrice de confusion pour la fonction linear comme kernel-

Ensuite, on a utilisé svm avec des features encodé avec un autoencoder et on obtenu des resultats moins importants que les précédents, cela peut etre à cause de la perte d'information lors de l'encodage mais bien évidemment avec des features réduites vu que l'autoencoder joue le role de la réduction dimensionnel.

Kernel	Accuracy
Linear	0.8045

LDA (Linear Discriminant Analysis) : a pour but la réduction dimensionnel :
On a essayé pour plusieurs nombre de composants et le nombre de composant égale à 10 nous a donné la meilleure accuracy en utilisant le classifier logistic regression de SKlearn :
On remarque que certaine classes ont été bien classées tandis que d'autres non, concernant la perte de donnée, on a constaté qu'en utilisant LDA on a gagné en rapidité en préservant assez de features afin d'avoir de bonnes résultats.
Résultat : logisitic regression as a classifier and 10 LDA composants as input features !

```
[[795   3  15  70   8   8  81   0  20   0]
 [  5 953   5  28   5   0   2   0   2   0]
 [ 32   1 697  14 161   5  79   0  11   0]
 [ 30  15  17 852  40   7  34   0   5   0]
 [  0   5 120  37 739   0  91   0   8   0]
 [  0   0   0   1   0 866   2  70  16  45]
 [146   3 131  55 111   7 511   1  35   0]
 [  0   0   0   0   0  39   0 905   1  55]
 [  5   0   6  12   5  12  22   4 932   2]
 [  0   0   0   0   0  29   1  50   0 920]]
Accuracy0.817
```

-Matrice de confusion + l'accuracy-