Reconnaître des chifres manuscrits grâce au Machine Learning

Eric Simon
Arina Bolkunova
Antoine Meyer

RAPPORT
DIGIT - RECOGNIZER

LINK DU NOTEBOOK

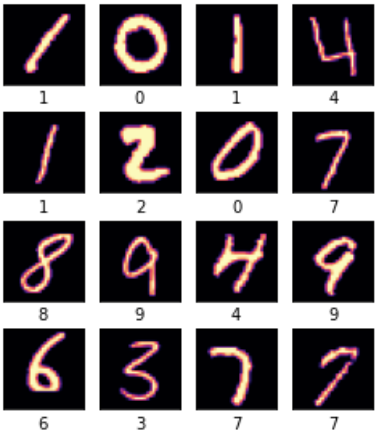
SIMPLON.CO

INTRODUCTION

OBJECTIF

A partir des valeurs contenues dans les features, produire un algorithme de Machine Learning étant capable d'en déterminer le label correspondant.

Les features sont ici **des niveaux de gris contenue dans des pixels**, et la target **des chiffres écrits de manière manuscrite**. Il s'agit ainsi de déterminer à partir d'une image le chiffre correspondant.



ANALYSE DES DONNÉES

Données :

https://www.kaggle.com/c/digit-recognizer/data

Structure :

2 jeux de données :

- 1 jeu de donnée "train.csv" destiné à l'entrainement du modèle.
  - 42000 lignes
  - 785 colonnes (784 features, 1 target ("label"))
- 1 jeu de donnée "test.csv" destiné à l'analyse.
  - 28000 lignes
  - 784 colonnes

Caractéristiques :

- Les jeux de donnée sont complets (aucune valeur manquante).
- Toutes les données sont de type "int64".
- Les features ont des valeurs entières comprises entre 0 et 255.
- Le label a des valeurs entières comprises entre 0 et 9.
- Les classes du label sont présentes à environ 10%.

PRE-PROCESSING

STANDARDISATION

La standardisation de nos données n'a donné aucun changement significatif de résultat. Cela est logique, dans le sens où **les valeurs des composantes sont faibles** (toutes comprises entre [0,255]) et **de même échelle** (toutes traduisent une valeur de gris dans un pixel). Standardiser n'est pas donc pas nécessaire.

ECHANTILLONAGE

Le volume du dataset pose problème au niveau du temps d'exécution. Il est possible de réduire ce temps en découpant le dataset plusieurs fois par la fonction ***train_test_split*** de Sklearn.

Etape 1 : On indique la taille du test_size à 0.8, ce qui veut dire qu'on prend que 20% du dataset.

Cette étape permet de réduire le dataset initial de 80%. Avec les 20% de données restantes, en veillant à ce qu'elles gardent la bonne proportion d'éléments avec stratify=y, on a un échantillon fonctionnel pour pouvoir essayer des modèles.

Etape 2 : On sépare alors cet échantillon en échantillons de test et d'entrainement avec une répartition de 0.2 (80% entrainement, 20% test), en conservant le proportion de y..

PRE-PROCESSING

ACP

L'Analyse en Composante Principale (ACP) est une méthode de pre-processing utilisant des opérations matricielles simples à partir de l'algèbre linéaire et des statistiques pour calculer une projection des données d'origine dans le même nombre ou moins de dimensions. Dans un cas où un nombre de features est important, **elle permet de réduire le nombre de dimension/features, et donc de réduire également les temps de calcul sans toutefois en affecter significativement la précision.**

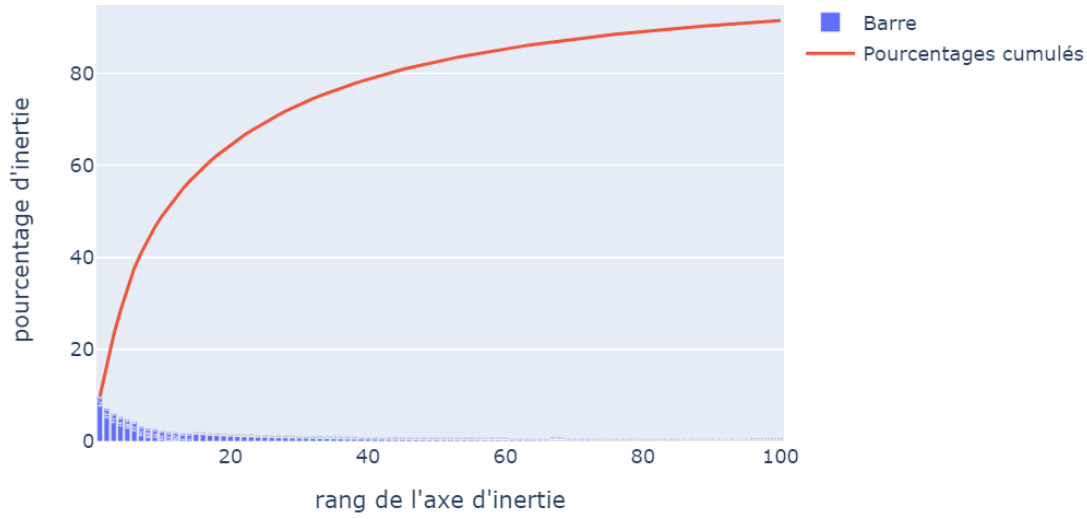
Raisons d'utiliser l'ACP :

- Décorrèle les variables du dataset.
- Résume les données avec le moins d'information possible.
- Permet de surmonter la redondance des fonctionnalités dans l'ensemble de données.
- Vise à capturer des informations précieuses expliquant une variance élevée, ce qui donne la meilleure précision. Cela rend les visualisations de données faciles à gérer.
- Diminue la complexité du modèle et augmente l'efficacité du calcul.

Pour utiliser l'ACP :

- Le dataset doit avoir beaucoup de features.
- Les données doivent être mises à l'échelle.

ON OBTIENT 90% DE LA VARIANCE EXPLIQUÉE AVEC 86 COMPOSANTES



COMPARAISONS DES MODÈLES

SUPPORT VECTOR CLASSIFIER

Le modèle SVC (Support Vector Classifier) était pertinent d'utilisation pour sa précision dans le traitement des données de type spatiale. Nous avons ainsi comparer 9 modèles disposant chacun de paramètres différents pour nous assurer les prédictions les plus précises relativement à un temps de travail le plus court.

MODÈLES	TEMPS	PRÉCISION
kernel : *linear* **train :** *80%* **ACP :** *oui*	*617s*	*0.932*
kernel : *linear* **train :** *16%* **ACP :** *non*	*5.6s*	*0.916*
kernel : *linear* **train :** *16%* **ACP :** *oui*	*NaN*	*NaN*
kernel : *poly* **train :** *80%* **ACP :** *oui*	*NaN*	*0.973*
kernel : *poly* **train :** *16%* **ACP :** *non*	*18.8s*	*0.942*
kernel : *poly* **train :** *16%* **ACP :** *oui*	*4.2s*	*0.967*
kernel : *rbf* **train :** *80%* **ACP :** *oui*	*288s*	*0.977*
kernel : *rbf* **train :** *16%* **ACP :** *non*	*20.7s*	*0.962*
kernel : *rbf* **train :** *16%* **ACP :** *oui*	*3.9s*	*0.97*

CONCLUSION

OPTIMISATION

Le meilleur rapport précision/temps de traitement parmi nos modèles a été obtenu avec un kernel RBF, un échantillon d'entrainement à 14% de nos données totales, et une réduction des features par ACP. L'outil GridSearchCV nous a permis par la suite de trouver les hyper-paramètres les plus adaptés afin de gagner encore plus de précision.

Les meilleurs hyper-paramètres déterminer par l'outil GridSearchCV :

- kernel : 'rbf'
- C : 101
- gamma : 1

PRÉCISION GAGNÉE : 0.97 -> 0.971

CONCLUSION

Notre modèle de SVC est capable de déterminer un chiffre avec 97,1% de chance de succès en analysant les niveaux de gris contenue dans une matrice de pixels.

Il a l'avantage d'être rapide et de nécessiter une quantité limitée de donnée.