

# Rapport : Projet Digit Recognizer

- Rapport : Projet Digit Recognizer
  - **I. Introduction**
    - Présentation du projet
      - *Contexte du projet*
      - *Objectifs du Projet*
    - Présentation des données
      - Présentation des données fournies
      - Description des images et des pixels
  - **II. Reconnaissance des chiffres (Digit Recognizer)**
    - Qu'est-ce qu'un OCR (Optical Character Recognition) ?
  - **III. Implémentation**
    - Préparation des données
    - Construction & Entraînement du modèle
    - Prédiction des échantillons inconnus
  - **IV. Résultats**
    - Analyse de la matrice de confusion sans PCA
      - *Pour*  $C=1e-2$
      - *Pour*  $C=1$
      - *Pour*  $C=1e2$
    - Bilan
- Conclusion
  - Synthèse des résultats
  - Amélioration

# I. Introduction

## Présentation du projet

### Contexte du projet

Le projet **Digit Recognition** est une introduction au domaine de la vision par ordinateur.

Proposé par Kaggle ([lien vers la compétition](#)), il demande de prédire correctement des chiffres manuscrits représentés dans des images.

Ce projet se base sur le jeu de données MNIST (*Modified National Institute of Standards and Technology*). Cette base de données contient 70 000 images en niveaux de gris représentant des chiffres manuscrits (de 0 à 9).

Chaque image mesure 28x28 pixels, avec des valeurs de pixels allant de 0 (blanc) à 255 (noir).

MNIST est largement utilisé comme référence dans la vision par ordinateur pour évaluer et comparer des algorithmes de classification, notamment des modèles de machine learning comme les SVM ou des réseaux de neurones.

### Objectifs du Projet

Comme mentionné dans l'introduction, l'objectif de ce projet est de réaliser un modèle capable de prédire des chiffres inscrit manuscritement dans des images.

Ces chiffres, allant de 0 à 9, sont extraits de données standardisées en niveaux de gris.

En tant qu'exercice, ce projet constitue un point d'entrée vers des sujets plus complexes en reconnaissance optique de caractères (**OCR**) et en apprentissage automatique.

Nous serons amené à :

- Utiliser un PCA pour réduire les dimensions des données.
- Entraîner un SVM avec une partie des données d'entraînements.
- Tester le modèle avec l'autre partie des données d'entraînements.
- Ajuster les paramètres de la SVM pour réduire le plus possible les erreurs de classification.
- Utiliser le modèle pour prédire la valeur d'échantillons sans label.

Nous cherchons à obtenir un modèle avec une précision optimale.

Pour y parvenir, nous testerons différentes valeurs pour le compromis entre marges et erreurs. En bonus, nous examinerons les performances du modèle avec et sans l'utilisation du PCA.

## Présentation des données

### Présentation des données fournies

Les données sont fournies sous la forme de deux fichiers : **train.csv** et **test.csv**.

- **train.csv** :
  - Contient 42 000 exemples d'entraînement.
  - Chaque ligne correspond à une image de 28 x 28 pixels, soit un total de 784 pixels par image.
  - La première colonne, appelée **label**, contient le chiffre manuscrit représenté (0 à 9).
  - Les colonnes suivantes, nommées `pixel0` à `pixel783`, contiennent les valeurs de luminance des pixels.
- **test.csv** :

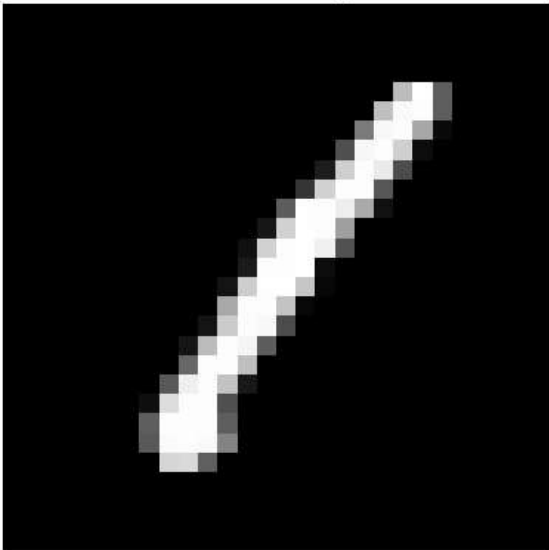
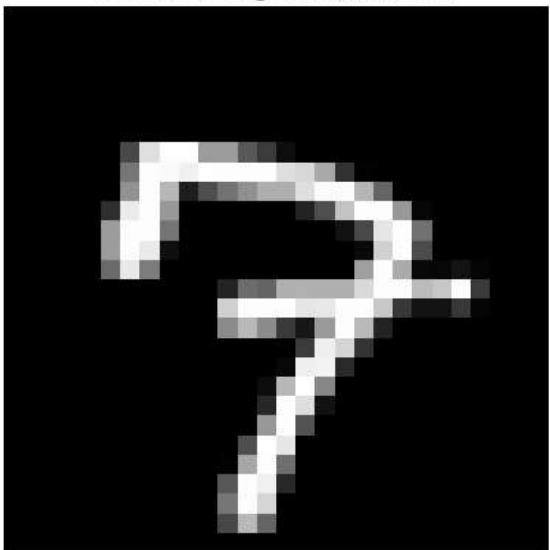
- Contient 28 000 exemples sans étiquette.
- Les données sont structurées de la même manière que `train.csv` , à l'exception de la colonne `label` .

## Description des images et des pixels

Chaque image est une matrice de 28 x 28 pixels, où chaque pixel est représenté par une valeur entière comprise entre **0** et **255**, correspondant à la luminosité :

- **0** représente un pixel totalement blanc.
- **255** représente un pixel totalement noir.

Les pixels sont ordonnés de gauche à droite et de haut en bas. Par exemple, un pixel dans la colonne `pixel131` correspondra à un pixel situé dans la 20<sup>e</sup> ligne et la 4<sup>e</sup> colonne de l'image.

Image d'Entrainement	Image de Test
<p>Train Data - Digit: 1</p> 	<p>Test Data - Digit: À déterminer</p> 

## II. Reconnaissance des chiffres (Digit Recognizer)

Ce projet fait office de porte d'entrée aux disciplines de vision par ordinateur assisté par intelligence artificiel.

Dans cette partie du rapport, nous allons développer l'un cas concret de cette discipline : la reconnaissance optique de caractère, aussi appelé OCR.

### Qu'est-ce qu'un OCR (Optical Character Recognition) ?

L'Optical Character Recognition (OCR) est une technologie permettant de convertir des images de texte manuscrit, dactylographié ou imprimé en données textuelles exploitables par des machines.

Cette méthode est couramment utilisée dans divers domaines tels que la numérisation de documents, la reconnaissance de plaques d'immatriculation, ou encore l'analyse de formulaires et de tickets de caisse.

Un OCR fonctionne en plusieurs étapes :

**1. Acquisition de l'image :**

Une image contenant du texte est capturée (par exemple, une photo ou un scan).

**2. Prétraitement de l'image :**

Les données brutes de l'image sont nettoyées pour réduire le bruit, ajuster la luminosité (à l'aide de processus tel que l'uniformisation) ou améliorer les contrastes.

**3. Segmentation :**

L'image est divisée en blocs plus petits, généralement des lignes, des mots, puis des caractères individuels.

**4. Extraction des caractéristiques :**

Les caractéristiques clés des formes et contours des caractères sont extraites pour les rendre exploitables par un algorithme de classification.

**5. Classification :**

Un modèle de machine learning ou d'intelligence artificielle est utilisé pour prédire à quel caractère correspond chaque segment. C'est ici qu'interviennent des algorithmes comme les SVM (Support Vector Machines) ou les réseaux de neurones dans des systèmes modernes.

**6. Post-traitement :**

Les prédictions brutes sont assemblées pour reformer le texte, en appliquant éventuellement des corrections orthographiques ou grammaticales basé sur des modèles linguistiques. Ainsi, si des lettres ou des mots n'ont pas été reconnu, alors il est possible de prédire ce qui était écrit à partir du sens général de la phrase.

# III. Implémentation

## Préparation des données

Première étape de notre projet, nous avons du préparé les données pour qu'elles puissent être utilisées par le SVM.

Nous avons travaillé avec le dataset **train.csv**, qui contient les données d'entrainement du modèle.

Pour rappel, la première colonne contient le label du chiffre (de 0 à 9), tandis que les 784 colonnes suivantes correspondent aux valeurs de luminance des pixels de l'image (28 x 28 pixels).

Vu le nombre trop important de variables, nous avons utilisé un PCA pour réduire les dimensions sans quoi le programme pourrait prendre des heures pour construire le modèle. Nous avons choisi un nombre de dimensions tel que au moins **95%** de la variance totale soit conservée. Cette méthode garantie que la majeure partie de l'information dans les données est retenue.

Nous avons utilisé les étapes suivantes :

- **Charger les données dans deux matrices :**

Une première matrice contient les labels des données et une seconde qui contient.

- **Utiliser une PCA pour réduire les dimensions :**

Les données transitent par une PCA pour réduire leurs dimensions.

- **Diviser les données en deux sous-ensembles :**

Pour nous assurer de la précision du modèle, nous avons divisé les données d'entraînements en deux sous-échantillons. Un premier qui sera utilisé pour entraîner le modèle, et un second pour évaluer les performances du modèle sur des données d'entraînement non utilisées.

- **Standardisation des données :**

Les données brutes d'entraînements ont été standardisées. Cette étape permet de centrer les données autour de 0 et de les normaliser, ce qui améliore la convergence et la performance de la SVM.

## Construction & Entraînement du modèle

Maintenant que les données d'entraînement prêtes, nous avons entraîné une SVM linéaire avec le noyau `'linear'` en utilisant la bibliothèque `scikit-learn`. Nous avons fait les étapes suivantes :

### 1. Définition du modèle :

Une instance de `SVC` a été créée avec les paramètres `kernel='linear'` et plusieurs `C` différentes.

Le paramètre  $C$  est l'un des paramètres clés d'une **SVM (Support Vector Machine)**. Il contrôle le compromis entre ces deux éléments :

#### i. Maximiser la marge entre les classes :

- Un  $C$  faible favorise la maximisation de la marge. Le modèle est alors plus tolérant aux erreurs de classification dans l'ensemble d'entraînement.
- Une marge large est synonyme d'un modèle plus généralisé, moins sensible aux fluctuations mineures des données d'entraînement.

#### ii. Minimiser les erreurs de classification :

- Un  $C$  élevé pénalise fortement les erreurs de classification. Le modèle essaiera alors de classer correctement tous les points de l'ensemble d'entraînement, quitte à avoir une marge plus petite.
- Cela peut entraîner un surapprentissage (overfitting), où le modèle est trop adapté aux données d'entraînement et perd en capacité de généralisation.

Nous avons utilisé la manière empirique pour déterminer notre  $C$  :

- **$C$  élevé ( $1e^2$ ) :**
  - Privilégie une erreur minimale dans l'ensemble d'entraînement.
  - Risque accru de surapprentissage, surtout si les données d'entraînement contiennent du bruit ou des erreurs. Nous ne devrions pas avoir de problème puisque les images ont déjà été traitées pour retirer bruits et erreurs.
- **$C$  faible ( $1e^{-2}$ ) :**
  - Privilégie une marge maximale et tolère davantage d'erreurs.
  - Modèle plus généralisé, mais potentiellement sous-adapté aux données complexes.
- **$C$  équilibré ( $1e^0$ ) :**
  - Cherche un compromis entre les deux.

Pour ce projet, un compromis équilibré serait le plus pertinent : nous voulons le moins d'erreur de classification tout en conservant des classes distinctes pour chaque chiffre.

Nous avons limité le compris à des valeurs inférieure à  $1e^2$  et supérieur à  $1e^{-2}$  sans quoi la compilation était trop longue (plus de 3 heures).

#### 1. Entraînement :

Le modèle a été entraîné sur l'ensemble d'entraînement.

#### 2. Prédiction sur l'ensemble de validation :

Une fois entraîné, le modèle a été utilisé pour prédire les labels des données de validation.

#### 3. Évaluation :

La précision globale a été calculée en comparant les prédictions aux labels réels de l'ensemble de validation.

Un score de précision et une matrice de confusion nous permettent d'analyser les performances du modèle en détail.

## Prédiction des échantillons inconnus

Une fois le modèle entraîné et avec ses performances mesurée, nous l'avons utilisé pour prédire les labels des données du fichier **test.csv**.

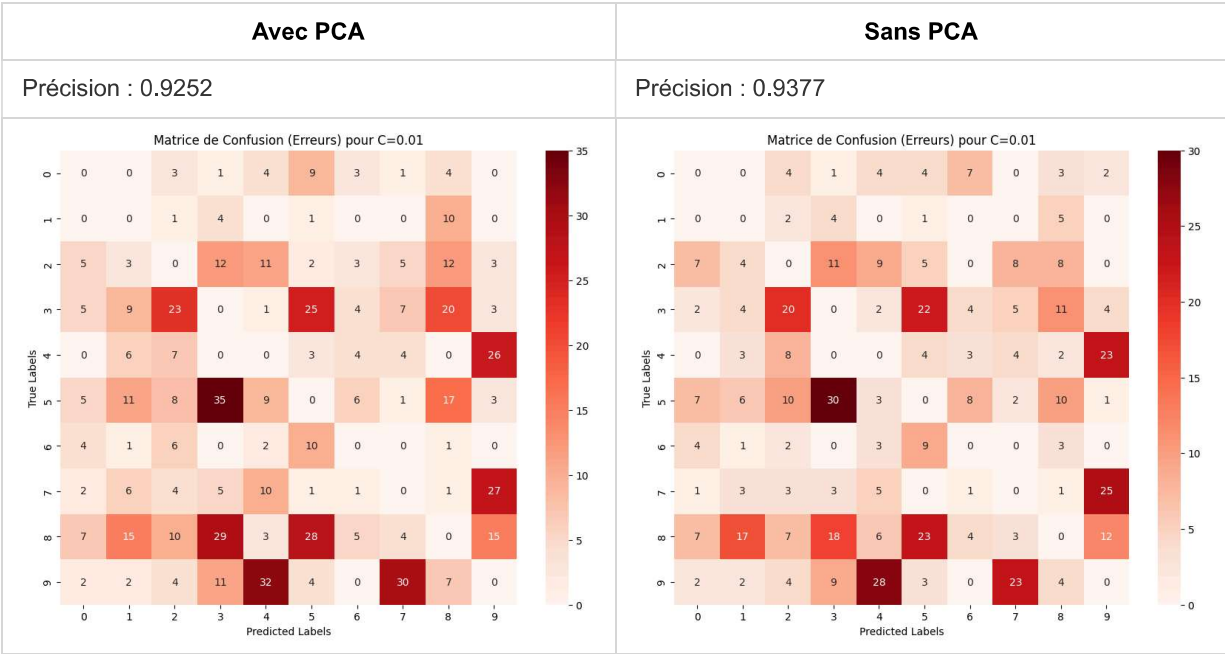
Le but est le suivant : prédire quel valeur est chaque échantillon.

Ces valeurs sont ensuite écrite dans le fichier **prediction\_svm.data**.

# IV. Résultats

## Analyse de la matrice de confusion sans PCA

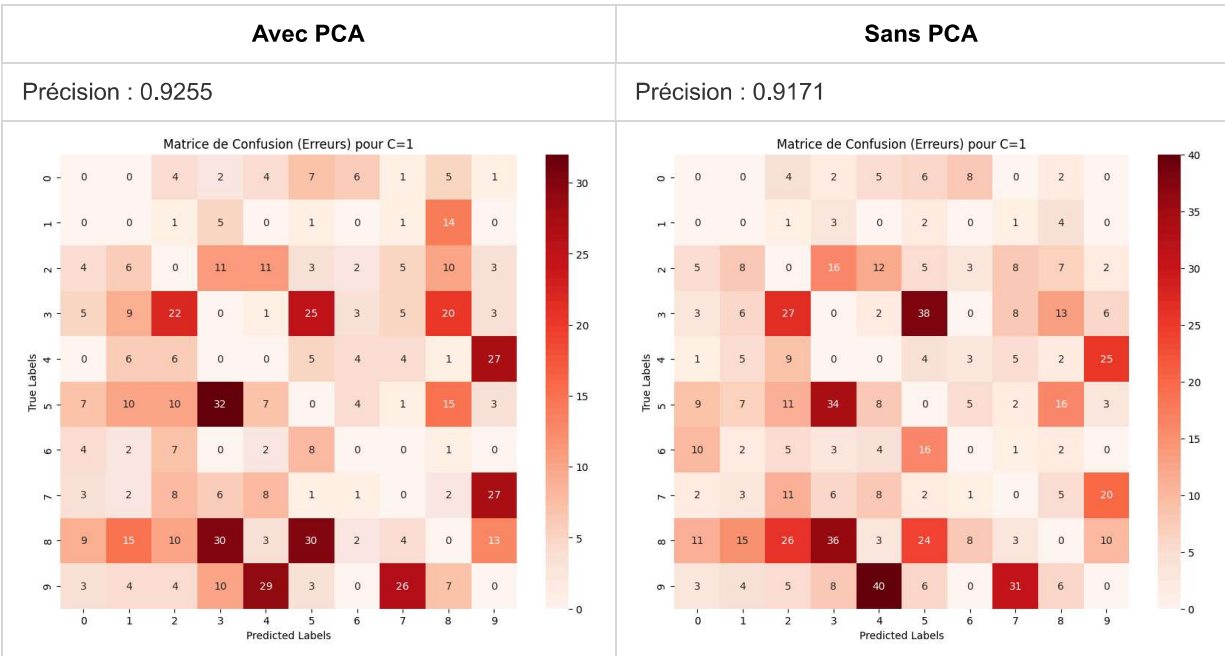
Pour  $C=1e-2$



Le modèle n'utilisant pas de PCA a un précision supérieure.

Pour  $C=1$

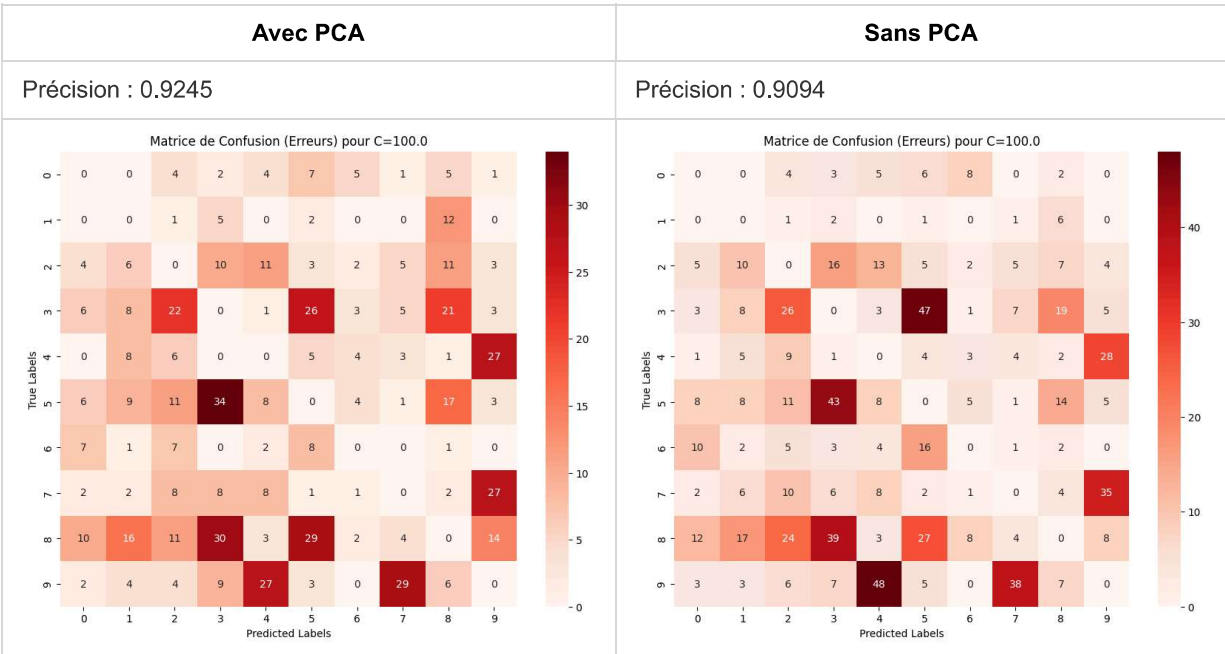
Pour un  $C$  égal à 1, nous obtenons ces matrices de confusion :



On peut voir sur cette matrice que le modèle se trompe souvent. Le modèle avec PDA a une précision légèrement supérieure.

**Pour  $C=1e2$**

Ci-dessous la matrice de confusion pour un  $C$  égal à  $1e2$ .



Le score de précision du modèle avec PDA ne bouge quasiment pas. Celui sans PDA cependant diminue comparé aux précédentes valeurs de  $C$ .  
Le temps de compilation devient très long.

**Bilan**

En lisant les matrices de confusion, on peut observer que le modèle a du mal à discerner la différence entre le 7 et les 9, les 4 et les 9, et 5 et les 3.

Quand on utilise un PCA, le taux de précision du modèle change très peu qu'importe la valeur de  $C$ . Cette dernière ne dépasse pas **92.5%**.

Quand on n'utilise pas de PCA, le taux de précision change bien plus : on passe de **90%** pour un  $C$  égal à 100 à **93.5%** quand  $C$  égal à 0.01.

En conclusion, une valeur équilibrée pour  $C$  est la marche à suivre quand on utilise un PCA. Il y a très peu de différence entre les trois valeurs de  $C$ , mais le temps de compilation est cependant complètement différent (plus le  $C$  s'éloigne de 1, plus il faut du temps).

Si l'on ne souhaite pas utiliser de PCA, alors il vaut mieux utiliser un  $C$  petit. La précision est bien plus élevée.



# Conclusion

## Synthèse des résultats

En conclusion, notre SVM est capable de prédire la valeur d'un chiffre écrit manuscritement. Le modèle est assez précis : au mieux, nous obtenons une précision de **93%**.

Il n'y a peu (voir pas) de différence de précision entre l'utilisation ou non du PCA. On observe par contre une différence dans le temps de compilation : l'utilisation du PCA rend la compilation significative plus rapide.

## Amélioration

Si nous avons utilisé une SVM pour ce projet, cette technique n'est plus beaucoup utilisée dans le domaine de la reconnaissance d'image. Des techniques modernes, bien plus complexes et spécialisées, sont maintenant utilisées, notamment :

- **Réseaux de neurones convolutifs (CNN)** : Particulièrement efficaces pour les images, ils apprennent automatiquement les caractéristiques pertinentes sans étape d'extraction manuelle.
- **Apprentissage par transfert** : Utilisation de modèles pré-entraînés comme ResNet ou VGG pour des tâches spécifiques.
- **Vision Transformers (ViT)** : Inspirés des techniques utilisées dans le traitement du langage, ces modèles divisent l'image en blocs et analysent les relations entre eux. Ils offrent une compréhension globale de l'image et montrent un fort potentiel pour des applications complexes.

Pour ce premier projet, nous avons voulu voir si une SVM était suffisante puisque les données étaient simples (images de chiffres bien isolés) et le prétraitement des données avait déjà été fait.

Cependant, le nombre d'erreur très important et le temps de compilation du modèle font qu'il serait préférable d'utiliser des méthodes plus complexes, comme par exemple un réseau de neurones.