Projet de reconnaissance automatisée de chiffres manuscrits

La Poste cherche à automatiser la reconnaissance de codes postaux manuscrits. Notre projet consistait à coder un algorithme qui identifie automatiquement un chiffre manuscrit à partir d'une image à l'aide d'un modèle de machine learning.

Nous avons identifié trois étapes classiques d'un projet de machine learning :

- Récupérer, explorer et visualiser les données
- Choisir et entraîner un ou plusieurs modèles de machine learning et les évaluer
- Optimiser le modèle de plus efficace sur nos données

Au sein de l'équipe nous avions tous des compétences complémentaires. La répartitions des tâches s'est faite naturellement selon les appétences de chacun.

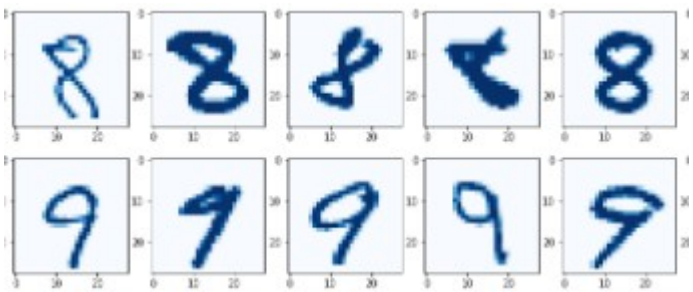
- Clément : exploration, visualisation et études statistiques des données
- Alexandre : entraînement et optimisation des hyper-paramètres du modèle
- Matthieu : évaluation du modèle

I\ Exploration des données et preprocessing

A\ Exploration et visualisation des données

Les jeux de donnée nous ont été fournis sous format csv depuis le site Kaggle. Nous n'avons pas eu à nous soucier de la problématique de récupérer les données depuis une API ou par scrapping. Nous avons tout d'abord regardé la taille de notre jeu de données. Nous avons constaté que nous avons 42000 ligne et 785 colonnes. Chaque ligne correspond à une image et chaque colonne correspond à un pixel au sein de l'image. Chaque image est composée de 783 pixels. Les données du fichier train.csv sont labellisées. La colonne Label contient le chiffre de l'image en valeur numérique. En revanche les données du fichier test.csv ne sont pas labellisées. Elles ne pourront nous servir à tester notre modèle mais uniquement par contrôle visuel en comparant le chiffre prédit à celui écrit dans l'image.

Nous avons observé visuellement un échantillon de 30 observations par chiffre. Les chiffres 8 et 9 nous ont semblé être ceux qui allaient poser le plus de problème à notre algorithme. Ils peuvent être confondus acec d'autres chiffres selon la qualité du tracé. Ces observations seront confirmés par le modèle.



Puis nous avons constaté que notre dataset ne comprenait aucune valeur manquante. Toute les données sont du type integer. Les valeurs au sein de chaque pixel peuvent aller de 0 à 254. Nous avons constaté que certains pixels ont une valeur fixe à 0 pour toutes les images.

Puis nous avons observé la fréquence de chaque chiffre au sein de notre dataset. Le nombre d'observations pour chaque chiffre varie de 3795 pour le chiffre 5 à 4684 pour le nombre 1. L'écart existe mais il est faible. Il ne nous a pas semblé justifier un travail de rééquilibrage.

```
train.stb.freq(['label'])
```

	label	count	percent	cumulative_count	cumulative_percent
0	1	4684	11.152381	4684	11.152381
1	7	4401	10.478571	9085	21.630952
2	3	4351	10.359524	13436	31.990476
3	9	4188	9.971429	17624	41.961905
4	2	4177	9.945238	21801	51.907143
5	6	4137	9.850000	25938	61.757143
6	0	4132	9.838095	30070	71.595238
7	4	4072	9.695238	34142	81.290476
8	8	4063	9.673810	38205	90.964286
9	5	3795	9.035714	42000	100.000000

B\ Preprocessing

Notre dataset de comprend aucune valeur manquante. Nous n'avons pas eu à supprimer des lignes ou à réaliser une imputation sur certaines valeurs.

Nous avons affiché la matrice de corrélation. Nous avons constaté que certains pixels sont très corrélés les une aux autres. Soit ce sont des pixels qui sont très utilisés dans l'ensemble des iméges, soit ce sont des pixels qui ne le sont jamais (valeur constante ou quasi-constante à 0). De plus, ls valeurs de chaque pixels peuvent varier de 0 à 254. Mais tous les pixels ne prennent pas tous la valeur maximale dans notre dataset. Ainsi, même si théoriquement nos données sont à la même échelle, en pratique dans notre dataset ce n'est pas le cas. Il nous a donc semblé important de standardiser nos données.

Enfin, nous avons divisé notre jeu de données en train_set et test_set avec la méthode train_test_split de sklearn.

II\ Entraînement et optimisation du modèle

A\ Recherche des hyper-paramètres optimaux

L'utilisation d'un algorithme SVM était dans la consigne de l'exercice. Nous sommes donc directement partis sur ce type d'algorithme et nous n'en avons pas testé d'autre.
Nous avons commencé par entraîner un modèle pour chaque kernel disponible et nous avons observé le l'accuracy_score pour chacun d'entre eux. Les résultats par ordre croissant de score ont été les suivants : Linear, Poly et rbf. Le kernel rbf était le plus performant sur notre jeu de données.

Puis nous avons utilisé GridSearchCV pour identifier les hyper-paramètres optimaux pour notre modèle SVM. Nous avons constaté que la meilleure valeurs pour le gamma était « scaled », soit la valeur par défaut. Un C avec une valeur de 10 nous donnait les meilleures performances.
Nous avons entraîné notre modèle SVM avec un kernel rbf avec C = 10 et gamma = « scaled » sur l'ensemble de notre train_set. Nous avons testé notre modèle sur notre set d'entraînement. Nous avons obtenu un accuracy score de 0,98.

```
clf = svm.SVC(C=10)
clf.fit(X_train, y_train)
preds = clf.predict(X_test)
accuracy_score(y_test, preds)
```

```
0.9801190476190477
```

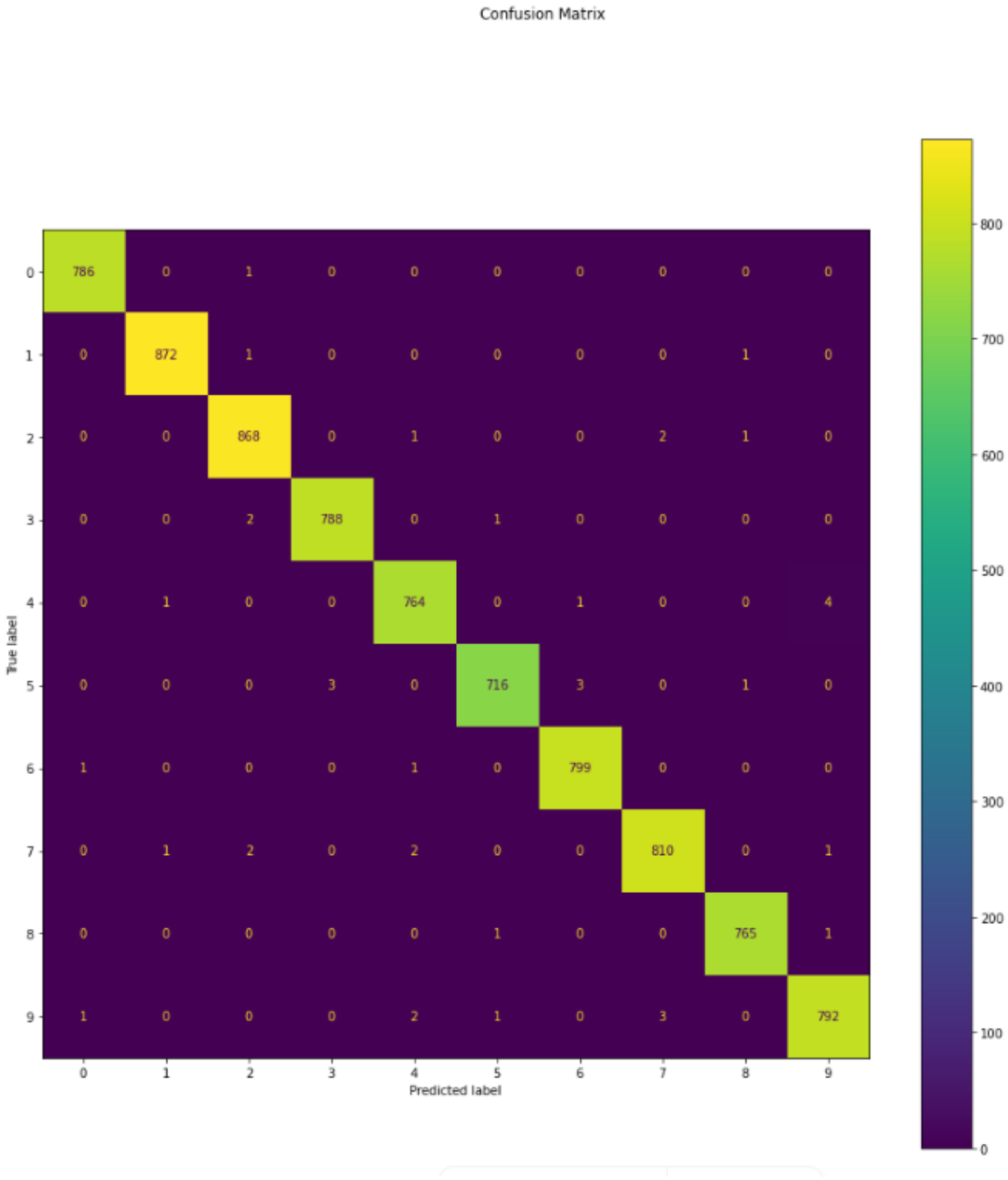
B\ Evaluation du modèle

Le rapport de classification a révélé que les trois chiffres pour lesquels notre modèle fait le plus d'erreurs de prédiction sont 2, 8 et 9. La matrice de confusion a révélé que le chiffre 9 est parfois prédit comme 0, 4 ou 7. Cette constatation vient confirmer des observations de la phase exploratoire. En effet, certains chiffres sont très mal écrits. Dans le cas du chiffre 9 les mauvaises classification viennent d'une boucle mal ou trop formée.

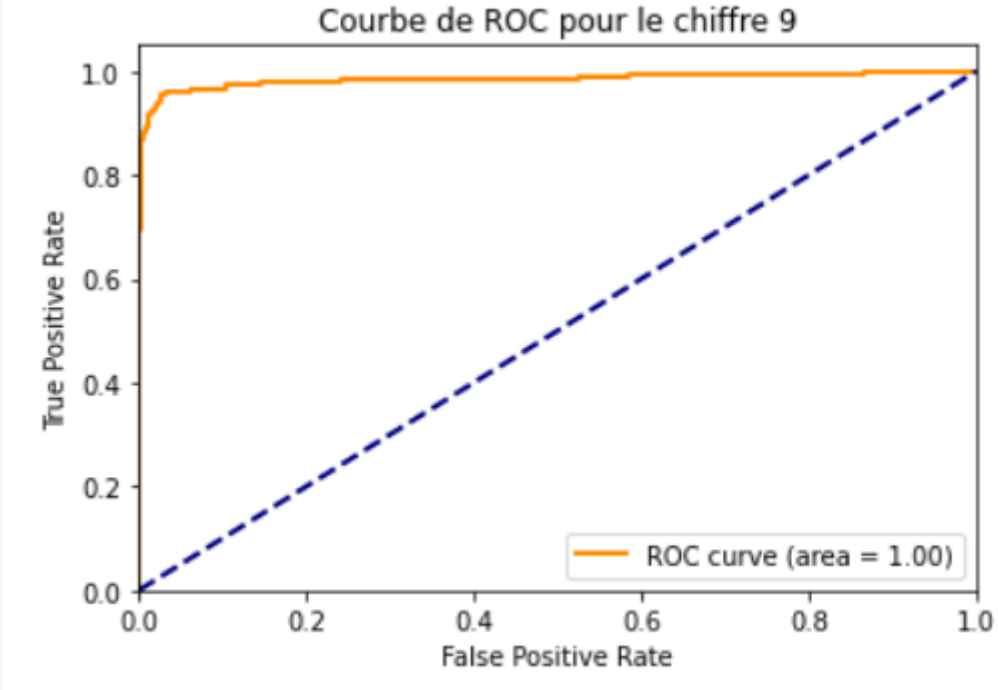
```
Classification report for classifier :
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       204
           1       0.98      0.99      0.99       221
           2       0.94      0.94      0.94       173
           3       0.97      0.93      0.95       218
           4       0.95      0.98      0.96       197
           5       0.97      0.96      0.96       183
           6       0.95      0.98      0.96       202
           7       0.97      0.96      0.96       223
           8       0.94      0.93      0.94       183
           9       0.93      0.92      0.93       196

    accuracy                           0.96      2000
   macro avg       0.96      0.96      0.96      2000
weighted avg       0.96      0.96      0.96      2000
```



Nous avons tracé la courbe de ROC pour chaque chiffre. Les chiffres pour lesquels il y a le plus de faux positifs sont 8 et 9. Encore une fois, ces deux chiffres sont visuellement très similaires et la capacité à les distinguer est dépendante de la qualité du tracé.



Enfin, nous avons effectué une prédictions sur des données du test.csv. Nous avons affiché les images afin de les comparer avec les résultats prédits. Nous avons constaté que le modèle est globalement performant, même si il reste quelques fausses prédictions.