Projet de reconnaissance de lettre

Python for Data Analysis

Jules Gaussens - Marine Mézin

Professeur: Romain Jouin

09.02.2019

Sommaire

Objectif du projet

- Identifier l'une des 26 lettres majuscules de l'alphabet latin parmis des affichages de pixels rectangulaires en noir et blanc

- Les caractères présents sur les affichages sont basés sur 20 polices différentes et chacun de ces caractères a été déformé de manière aléatoire.
- Le fichier complet comporte 20 000 lignes avec 16 attributs par ligne.

Présentation du jeu de données

- 20 000 lignes
- 17 colonnes

	letter	rec_1	rec_2	rec_3	rec_4	rec_5	rec_6	rec_7	rec_8	rec_9	rec_10	rec_11	rec_12	rec_13	rec_14	rec_15	rec_16
19995	D	2	2	3	3	2	7	7	7	6	6	6	4	2	8	3	7
19996	C	7	10	8	8	4	4	8	6	9	12	9	13	2	9	3	7
19997	T	6	9	6	7	5	6	11	3	7	11	9	5	2	12	2	4
19998	S	2	3	4	2	1	8	7	2	6	10	6	8	1	9	5	8
19999	A	4	9	6	6	2	9	5	3	1	8	1	8	2	7	2	8

Présentation du jeu de données

- Aucune valeur manquante, aucune valeur nulle

- Col. 1 : Lettre majuscule
  - Variable cible
  - Type : object (typage automatique de Python pour l'importation d'une variable chaîne de caractères)

- Col.2 à Col. 16 :
  - Variables statistiques
  - Type : integer

Présentation du jeu de données

- **Col.2 : position horizontale**, en nombre de pixels, calculée à partir du bord gauche de l'image jusqu'au bord du plus petit rectangle, ou boîte rectangulaire, pouvant être dessiné autour de la lettre à prédire
- **Col.3** : **position verticale**, en nombre de pixels, calculée du bas de l'image au bord de la boîte rectangulaire
- **Col.4** : **largeur** de la boîte rectangulaire
- **Col.5** : **hauteur** de la boîte rectangulaire
- **Col.6** : **nombre total de pixels** sur l'image

Présentation du jeu de données

- **Col.7** : **position horizontale moyenne** des pixels de la boîte rectangulaire (centre de la boîte divisé par sa largeur). Cette caractéristique a une valeur négative si l'image est "trop à gauche" (exemple : lettre L).

- **Col.8** : **position verticale moyenne** des pixels de la boîte rectangulaire (centre de la boîte divisé par sa hauteur).

Présentation du jeu de données

- **Col.9** : **valeur quadratique moyenne des distances horizontales** en pixels (mesurées en Col.7). Cet attribut aura une valeur plus élevée pour les images dont les pixels sont plus largement séparés dans le sens horizontal, comme ce serait le cas pour les lettres W ou M.
- **Col.10** : **valeur quadratique moyenne des distances verticales** en pixels (mesurées en Col.8).
- **Col.11** : **produit moyen des distances** horizontales et verticales (mesurées en Col.7 et Col.8) pour chaque pixel. Cet attribut a une valeur positive pour les lignes diagonales allant du bas à gauche au haut à droite et une valeur négative pour les lignes diagonales de haut à gauche au bas à droite.

Présentation du jeu de données

- **Col.12** : **valeur moyenne de la distance horizontale au carré multipliée par la distance verticale** de chaque pixel (mesure la corrélation de la variance horizontale avec la position verticale).
  Formule : $x^2 * y$

- **Col.13** : **valeur moyenne de la distance verticale au carré multipliée par la distance horizontale** de chaque pixel (mesure la corrélation de la variance verticale avec la position horizontale).
  Formule : $x * y^2$

Présentation du jeu de données

- **Col.14** : **nombre moyen de bords** rencontrés lors de la numérisation de **gauche à droite** à toutes les positions verticales de la boîte. Cette mesure fait la distinction entre des lettres comme "W" et "M" ou des lettres comme "T" et "L".
- **Col.15** : **somme des positions verticales des bords** rencontrés (mesurées en Col.14). Cette fonctionnalité donnera une valeur plus élevée s'il y a plus de bords en haut de la boîte, comme dans la lettre "Y.".
- **Col.16** : **nombre moyen de bords** rencontrés lors de la numérisation de **bas en haut** à toutes les positions horizontales de la boîte.
- **Col.17** : **somme des positions horizontales des bords** rencontrés (mesurées en Col.16).

Emplacement des parties du projet

- Scraping des données : *scraping.ipynb*
- Visualisation des données : *data_viz.ipynb*
- Modélisation et réglage des paramètres : *modelisation.ipynb*

Scraping des données

Méthode utilisée :

- Accès aux données sur leur url via **Selenium** et un **webdriver Chrome**
- Clic sur la localisation des données via le xpath + click()
- Téléchargement des données

Difficulté rencontrée :
Lors de l'utilisation du driver Firefox, problème pour enlever la popup Firefox au moment du téléchargement
=> Problème réglé en utilisant Chrome.

Visualisation des données

Utilisation de :

- **Pandas** pour manipuler les DataFrames
- **Matplotlib** et **Seaborn** pour tracer les graphiques

Objectif :

- Comprendre la signification des données

Visualisation des données

- Graphe A : Nombre d'occurrences de chaque lettre par ordre décroissant
- Graphe B : Position moyenne horizontale de la boîte pour chaque lettre
- Graphe C : Fréquence d'apparition des valeurs des positions horizontales des boîtes
- Graphe D : Position moyenne verticale de la boîte pour chaque lettre
- Graphe E : Fréquence d'apparition des valeurs des positions verticales des boîtes
- Graphe F : Fréquence du nombre total de pixels sur les images
- Graphe G : Fréquence du nombre total moyen de pixels sur les images par lettre
- Graphe H : Nombre total moyen de pixels sur les images par lettre

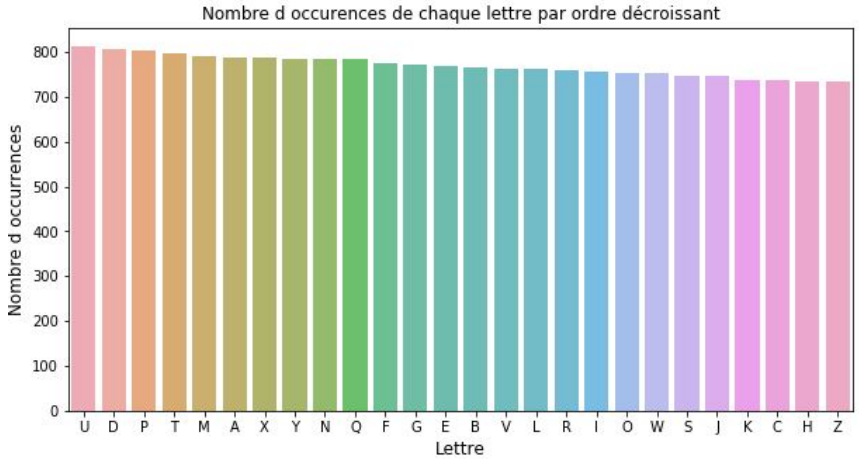
Visualisation des données

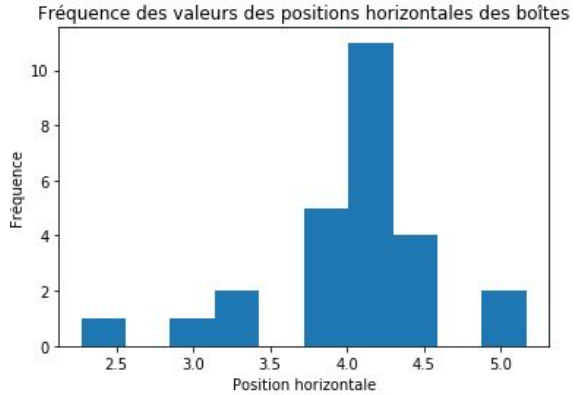
- Graphe I : Fréquence de la valeur quadratique moyenne des distances horizontales
- Graphe J : Fréquence de la valeur quadratique moyenne des distances horizontales par lettre
- Graphe K : Valeur quadratique moyenne des distances horizontales par lettre
- Graphe L : Fréquence de la valeur quadratique moyenne des distances verticales
- Graphe M : Fréquence de la valeur quadratique moyenne des distances verticales par lettre
- Graphe N : Valeur quadratique moyenne des distances verticales

Visualisation des données

- Graphe O : Fréquence du nombre moyen de bords rencontrés aux positions verticales par lettre
- Graphe P : Nombre moyen de bords rencontrés aux positions verticales par lettre
- Graphe Q : Fréquence du nombre moyen de bords rencontrés aux positions horizontales par lettre
- Graphe R : Nombre moyen de bords rencontrés aux positions horizontales par lettre

Exemples de nos visualisations





Modélisation et optimisation - RandomForest

Librairie : **Scikit Learn**

Méthode utilisée :

1. Premier modèle **RandomForest** sans réglage des hyperparamètres
2. Optimisation avec variation des hyperparamètres grâce à **grid Search**
    - min_samples_split *(nombre minimal d'échantillons requis pour scinder un noeud)*
    - n_estimators *(nombre d'arbres)*
    - max_depth *(profondeur des arbres)*
    - max_features *(nombre de colonnes à considérer pour trouver le meilleur split)*

Modélisation et optimisation - RandomForest

- 1er test :

```
param_grid = {  'min_samples_split' : [5, 10, 15], 'n_estimators': [20, 40, 60], 'max_depth'
:range(10)[1:], 'max_features':[2, 8, 14]}
{'max_depth': 9, 'max_features': 2, 'min_samples_split': 5, 'n_estimators': 60}
0.8249241964827168
```

- 2ème test :

```
param_grid = {  'min_samples_split' : [3, 5, 7], 'n_estimators': [50, 60, 70], 'max_depth' :[8.5,
9, 9.5], 'max_features':[1, 2, 3]}
{'max_depth': 9, 'max_features': 3, 'min_samples_split': 3, 'n_estimators': 50}
0.8382049727107338
```

Modélisation et optimisation - RandomForest

- 3ème test :

```
param_grid = {  'min_samples_split' : [2, 3, 4], 'n_estimators': [45, 50, 55], 'max_depth' : [9],
'max_features':[2, 3, 4]}
{'max_depth': 9, 'max_features': 3, 'min_samples_split': 4, 'n_estimators': 50}
0.8351728320194057
```

- 4ème test :

```
param_grid = {  'min_samples_split' : [3, 4, 5], 'n_estimators': [45, 50, 55, 60], 'max_depth' :
[9], 'max_features':[3]}
{'max_depth': 9, 'max_features': 3, 'min_samples_split': 4, 'n_estimators': 50}
0.8367495451788963
```

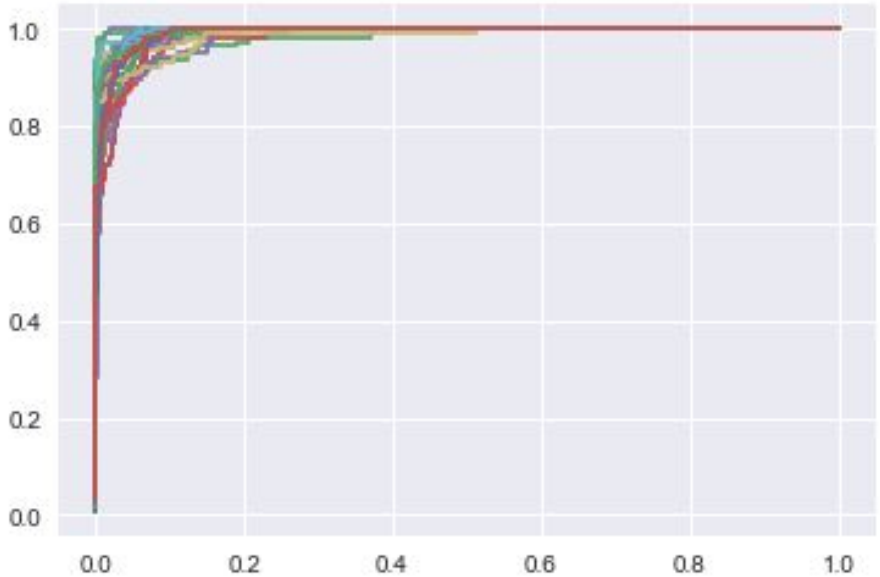
Modélisation et optimisation - RandomForest

Tous les résultats :

```
[mean: 0.81783, std: 0.00650, params: {'max_depth': 9, 'max_features': 3, 'min_samples_split': 3, 'n_estimators': 45}
, mean: 0.81777, std: 0.00656, params: {'max_depth': 9, 'max_features': 3, 'min_samples_split': 3, 'n_estimators': 50
}, mean: 0.82171, std: 0.00747, params: {'max_depth': 9, 'max_features': 3, 'min_samples_split': 3, 'n_estimators': 5
5}, mean: 0.82444, std: 0.01177, params: {'max_depth': 9, 'max_features': 3, 'min_samples_split': 3, 'n_estimators':
60}, mean: 0.82141, std: 0.00195, params: {'max_depth': 9, 'max_features': 3, 'min_samples_split': 4, 'n_estimators':
45}, mean: 0.81965, std: 0.00630, params: {'max_depth': 9, 'max_features': 3, 'min_samples_split': 4, 'n_estimators':
50}, mean: 0.82074, std: 0.00966, params: {'max_depth': 9, 'max_features': 3, 'min_samples_split': 4, 'n_estimators':
55}, mean: 0.82347, std: 0.01660, params: {'max_depth': 9, 'max_features': 3, 'min_samples_split': 4, 'n_estimators':
60}, mean: 0.81995, std: 0.00939, params: {'max_depth': 9, 'max_features': 3, 'min_samples_split': 5, 'n_estimators':
45}, mean: 0.82044, std: 0.00861, params: {'max_depth': 9, 'max_features': 3, 'min_samples_split': 5, 'n_estimators':
50}, mean: 0.82383, std: 0.00860, params: {'max_depth': 9, 'max_features': 3, 'min_samples_split': 5, 'n_estimators':
55}, mean: 0.82923, std: 0.00848, params: {'max_depth': 9, 'max_features': 3, 'min_samples_split': 5, 'n_estimators':
60}]
```

Visualisation des performances

Meilleur classifieur Random Forest
Courbe ROC pour chaque classe :



Modélisation et optimisation - Xgboost

Librairie : **Scikit Learn**

Méthode utilisée :

1. Premier modèle **Xgboost** sans réglage des hyperparamètres
2. Optimisation avec variation des hyperparamètres grâce à **grid Search**
    - n_estimators *(nombre d'arbres)*
    - max_depth *(profondeur des arbres)*
    - learning_rate

Modélisation et optimisation - Xgboost

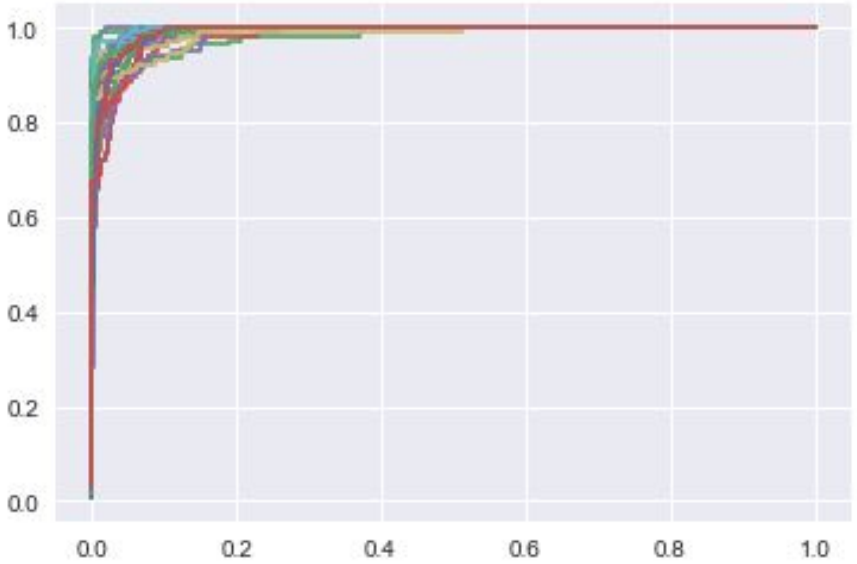
15 premiers résultats :

```
print(grid_search.grid_scores_[1:15])
```

```
[mean: 0.89090, std: 0.00149, params: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 40}, mean: 0.90922, std:
0.00259, params: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 60}, mean: 0.92201, std: 0.00175, params: {'l
earning_rate': 0.1, 'max_depth': 5, 'n_estimators': 80}, mean: 0.93202, std: 0.00186, params: {'learning_rate': 0.1,
'max_depth': 5, 'n_estimators': 100}, mean: 0.93687, std: 0.00109, params: {'learning_rate': 0.1, 'max_depth': 5, 'n_
estimators': 120}, mean: 0.91195, std: 0.00052, params: {'learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 20},
mean: 0.92984, std: 0.00050, params: {'learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 40}, mean: 0.93881, std:
0.00066, params: {'learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 60}, mean: 0.94281, std: 0.00146, params: {'
learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 80}, mean: 0.94560, std: 0.00101, params: {'learning_rate': 0.1
, 'max_depth': 10, 'n_estimators': 100}, mean: 0.94724, std: 0.00074, params: {'learning_rate': 0.1, 'max_depth': 10,
'n_estimators': 120}, mean: 0.91892, std: 0.00100, params: {'learning_rate': 0.1, 'max_depth': 15, 'n_estimators': 20
}, mean: 0.93366, std: 0.00144, params: {'learning_rate': 0.1, 'max_depth': 15, 'n_estimators': 40}, mean: 0.94033, s
td: 0.00170, params: {'learning_rate': 0.1, 'max_depth': 15, 'n_estimators': 60}]
```

Visualisation des performances

Meilleur classifieur XGBoost
Courbe ROC pour chaque classe :



Comparaison des performances

- Random Forest - Meilleur score: **0.8295936931473621**
- XGBoost - Meilleure score: **0.952152819890842 9**

=> XGBoost plus performant

Conclusion

- Scraping des données : Sélénium et ChromeDriver
- Data Visualisation : Matplotlib et Seaborn
- Modélisation : Random Forest et XGboost
    - Optimisation d'hyperparamètres (n_estimators, max_depth, etc)
- Meilleur modèle dans notre cas => **XGboost**
- AUC correct pour nos classifications