

Prédiction de popularité

Un Projet Python for data analysis

Sanaâ DECHANE & Wail LATIF



Introduction

Ce projet a pour but d'étudier le "Online News Popularity Data Set » et mettre en place un modèle de machine learning permettant de prédire la popularité (nombre de partages) d'un article donné.

Pour atteindre notre objectif nous avons dû passer par plusieurs étapes intermédiaire :

- Exploration des données
- Nettoyage des données
- Visualisation des données
- Mise en place des modèles
- Exposition du modèle (API)



► Exploration des données

Description des données

Une fois que nous avons récupéré le jeu de données qui nous a été attribué, nous avons décidé de faire des recherches et essayer de mieux comprendre sa structure et son contenu avant de nous lancer dans la phase de machine learning.

Notre jeu de données est le « Online News Popularity Data Set » fournis par l'« UCI Machine Learning Repository ». Grâce aux informations récupérées via l'UCI, nous avons découvert qu'il s'agit d'une base de données qui référence l'ensemble des articles publiés par un site d'actualité pour une période de 2 ans. Le site web en question s'appelle « Marshable », et a été fondé par Pete Cashmore en juillet 2005.

Afin de connaître de quelle période date nos données nous avons utilisé la colonne *timedelta* correspondant au delta entre la date de publication de l'article et la date d'acquisition du jeu de données par l'UCI (8 Janvier 2015).

```
# On calcule la date de l'article le plus récent et le plus ancien du data set
start_date = (acquisition_date - timedelta(days=max_timedelta)).strftime("%d-%m-%Y")
end_date = (acquisition_date - timedelta(days=min_timedelta)).strftime("%d-%m-%Y")
```

```
print( f"Le data set contient {data_set.shape[0]} articles datant du {start_date} jusqu'au {end_date}" )
```

Le data set contient 39644 articles datant du 07-01-2013 jusqu'au 31-12-2014

Étude des colonnes

Notre jeu de données est composé de 61 colonnes dont 58 prédictives, 1 cible qui correspond au nombre de partages (shares) et 2 non prédictives qui sont *timedelta* (colonnes utilisées précédemment) et url correspondant à l'url de l'article. Ces informations nous ont été données par l'UCI.

Après quelques recherches sur les colonnes composant nos données, nous avons pu remarquer que certaines colonnes telles que *weekdays*, ou encore *data_channel* semble avoir subi un encodage one hot, de plus nous avons pu remarquer qu'un travail de NLP ainsi que de clustering (utilisant LDA) a été effectué sur ces données.

Grâce à toutes ces informations nous avons pu classer les colonnes en 2 groupes :

- Statistique d'une étude NLP (n_tokens_title, n_tokens_content, LDA_00 ...)
- Analytics web (num_keywords, data_channels, self_reference_avg_shares ...)

Discrétisation de la cible

L'objectif de ce projet étant de déterminer la popularité d'un article selon les données qui ont été mises à notre disposition. Ce problème est donc un problème de classification, il nous a fallu donc déterminer les seuils permettant de décrire un article comme 'populaire' / 'non populaire'.

Plusieurs découpages sont possibles, d'après nos tests plus le découpage est « facile » meilleurs sont les résultats des modèles (ex: populaire / non populaire) . Cependant au vu de la répartition de nos données, nous avons jugé que pour être précis il fallait mettre en place un minimum de niveaux de popularité :

- Non populaire : < 708 partages (10% des articles)
- Peu populaire : $708 \leq \< 1100$ partages (25% des articles)
- Neutre : $1100 \leq \< 2000$ partages (30% des articles)
- Populaire : $2000 \leq \< 6100$ partages (25% des articles)
- Très populaire : ≥ 6100 partages (10% des articles)

Cette information a été stockée dans une nouvelle colonne « popularity ».



► Visualisation des données

Préparation des données

Avant de démarrer la visualisation des données, nous avons voulu créer une copie de notre jeu de données que nous avons appelée `data_vis`.

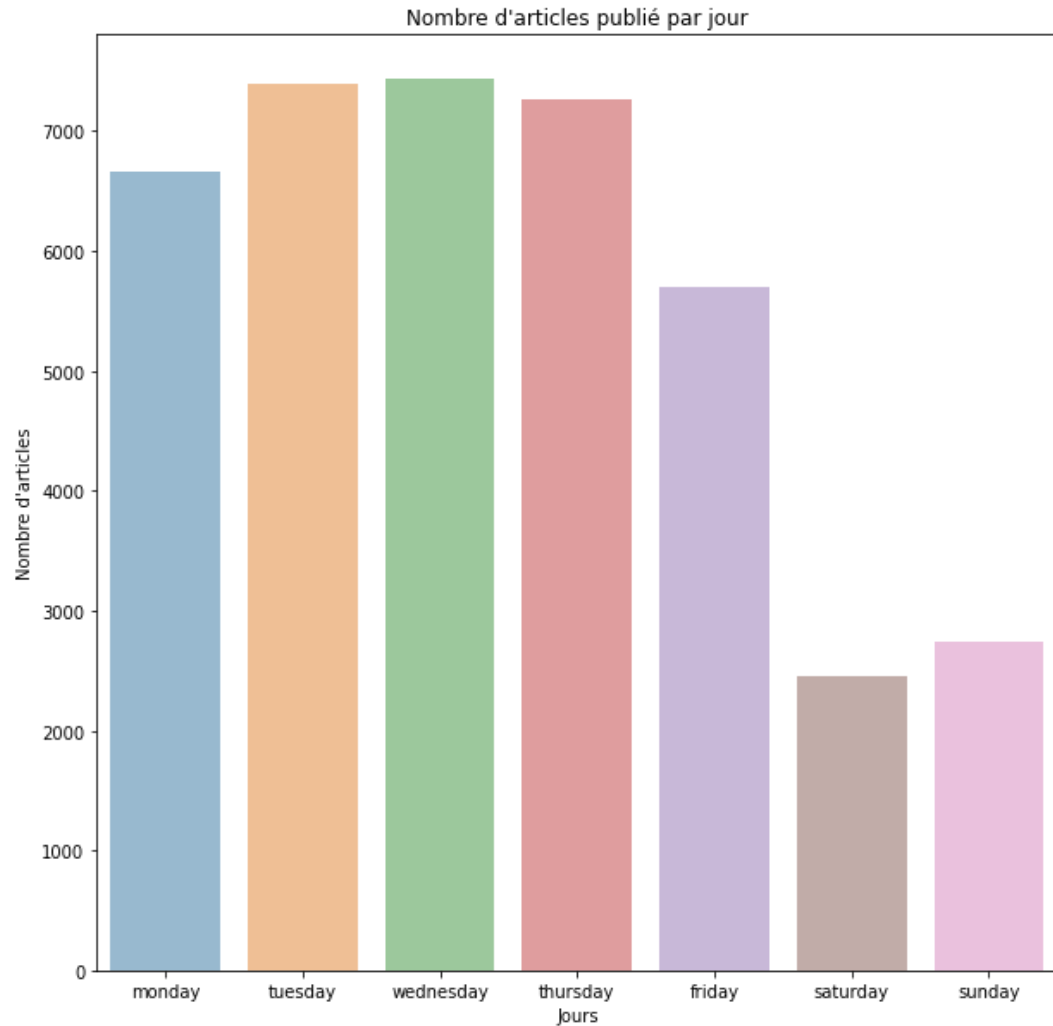
Dans ce nouveau jeu de données nous avons regroupé les colonnes ayant subi un encodage one hot en une seule colonne « `publish_days` » pour le jour de publication, et « `category` » pour les colonnes `data_channels`.

Pour chacune de ces deux colonnes un sanity check a été effectué afin de vérifier les résultats avant la création des colonnes.

```
# Mise à jour du data set
data_vis.insert(loc=12, column='category', value=_)
data_vis.drop(columns = channel_columns, axis = 1, inplace=True)
data_vis['category']
```

```
0      Entertainment
1          Business
2          Business
3      Entertainment
4          Tech
...
39639          Tech
39640      Social Media
39641          Others
39642          World
39643      Entertainment
Name: category, Length: 39644, dtype: object
```

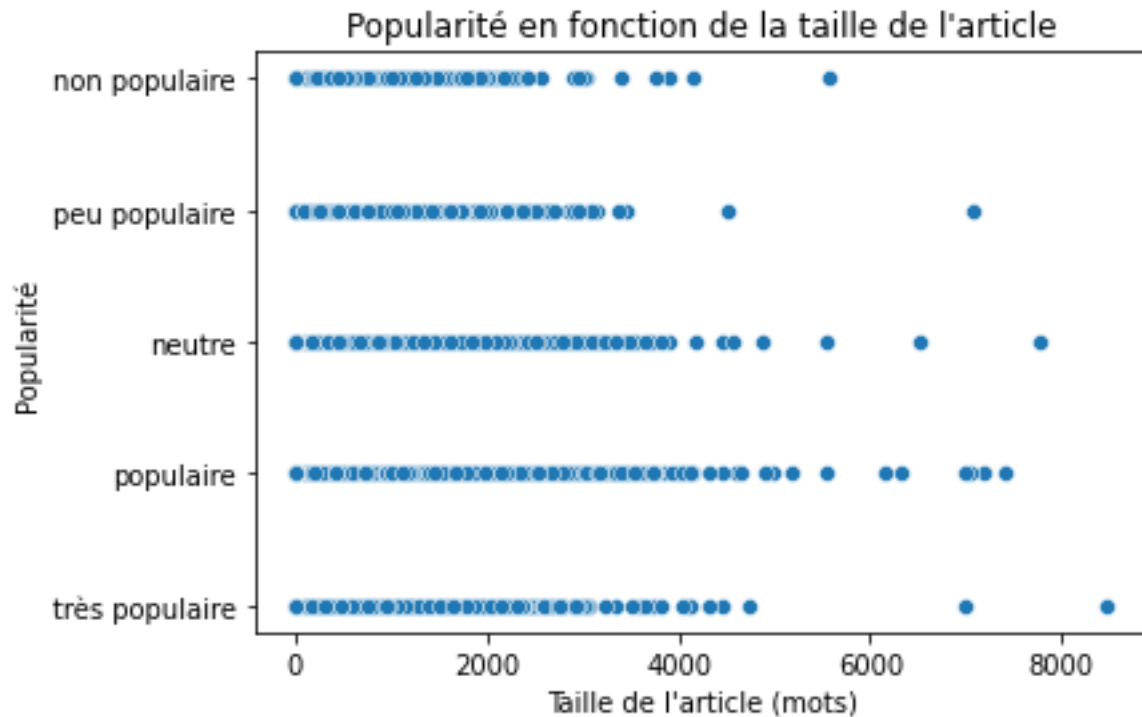

Nombre d'articles publiés en fonction du jour



Dans un premier temps nous avons voulu voir le nombre d'articles publiés par jour afin de voir si ce nombre changeait d'un jour à l'autre.

Il semblerait qu'en semaine il y ait plus d'articles publiés que durant le week-end ce qui semble logique, de plus il y a un pic en milieu de semaine (mercredi)

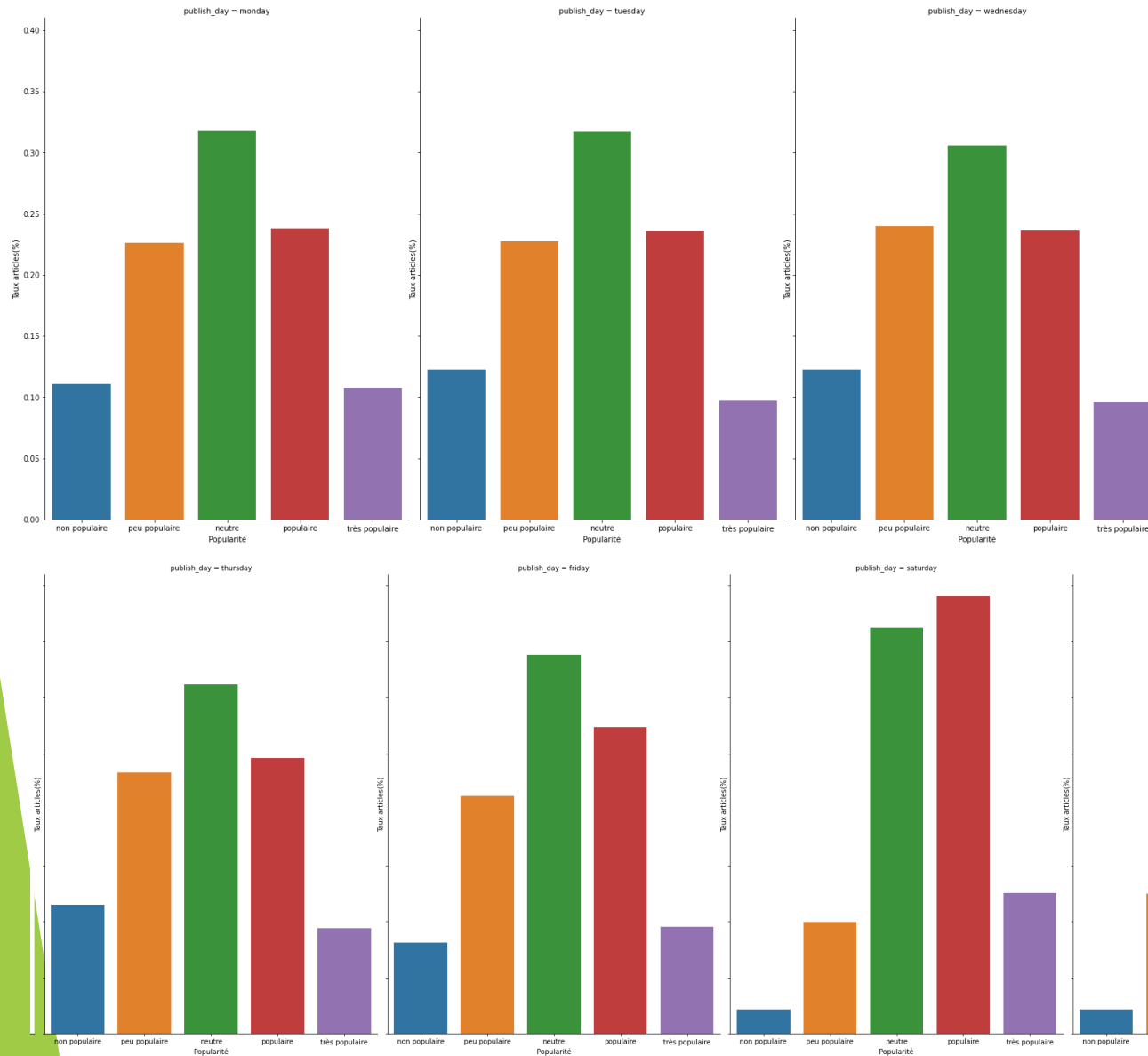
Popularité des articles en fonction de leur taille



Nous avons pensé que la taille de l'article pouvait avoir un impact sur sa popularité nous avons donc voulu vérifier.

Selon cette figure les articles les plus populaires contiennent moins de 3000 mots.

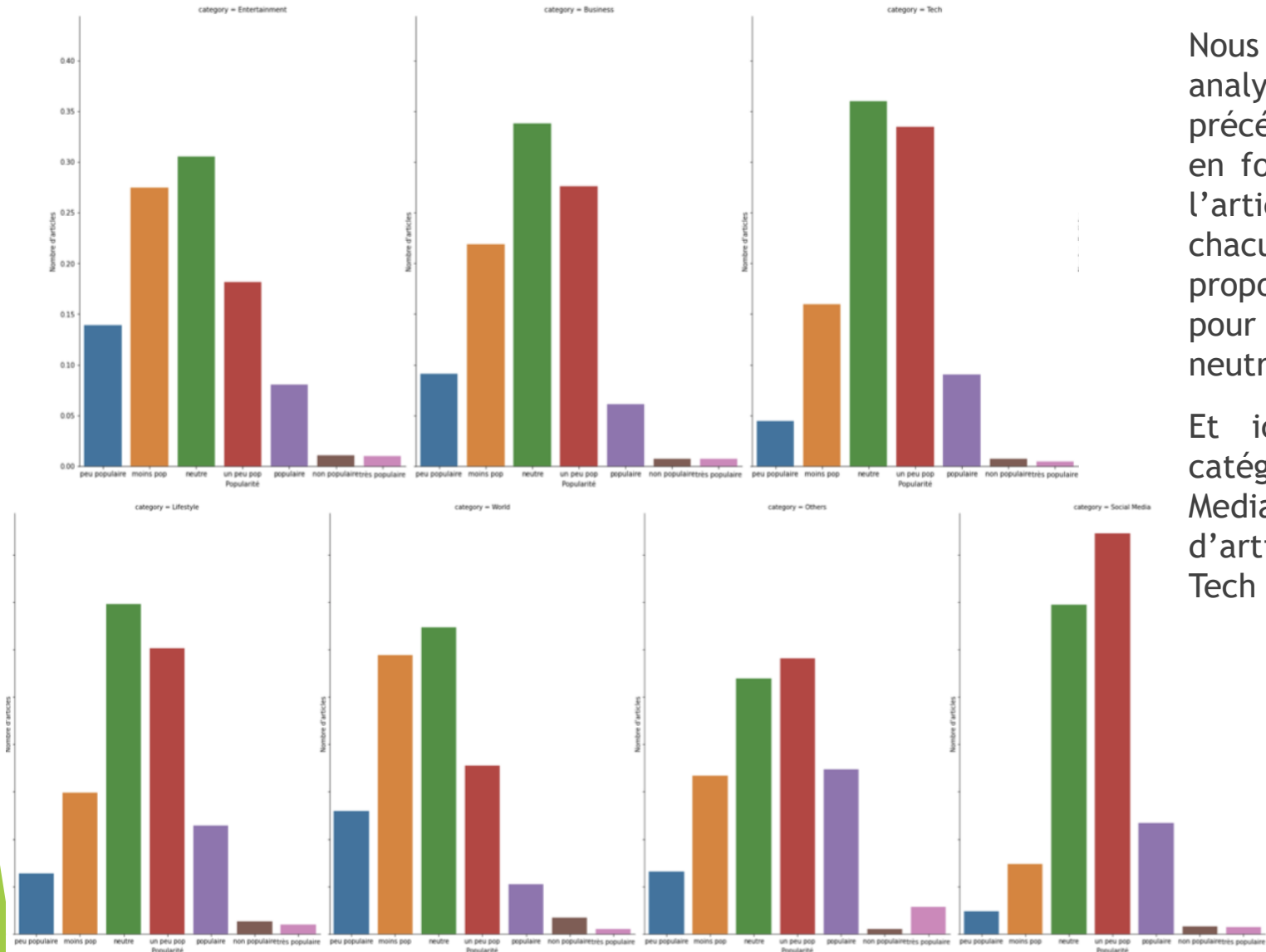
Distribution des articles en fonction de leur popularité et jour de publication



Nous avons pu voir précédemment qu'en semaine le nombre d'articles publiés est plus important que durant le week-end, nous voulions donc voir la proportion des articles publié pour chacune des classes (populaire, neutre non populaire ...).

Il semblerait que la probabilité qu'un article soit populaire ait bien plus grande en week-end, en effet près de 40% des articles publié le samedi sont populaires.

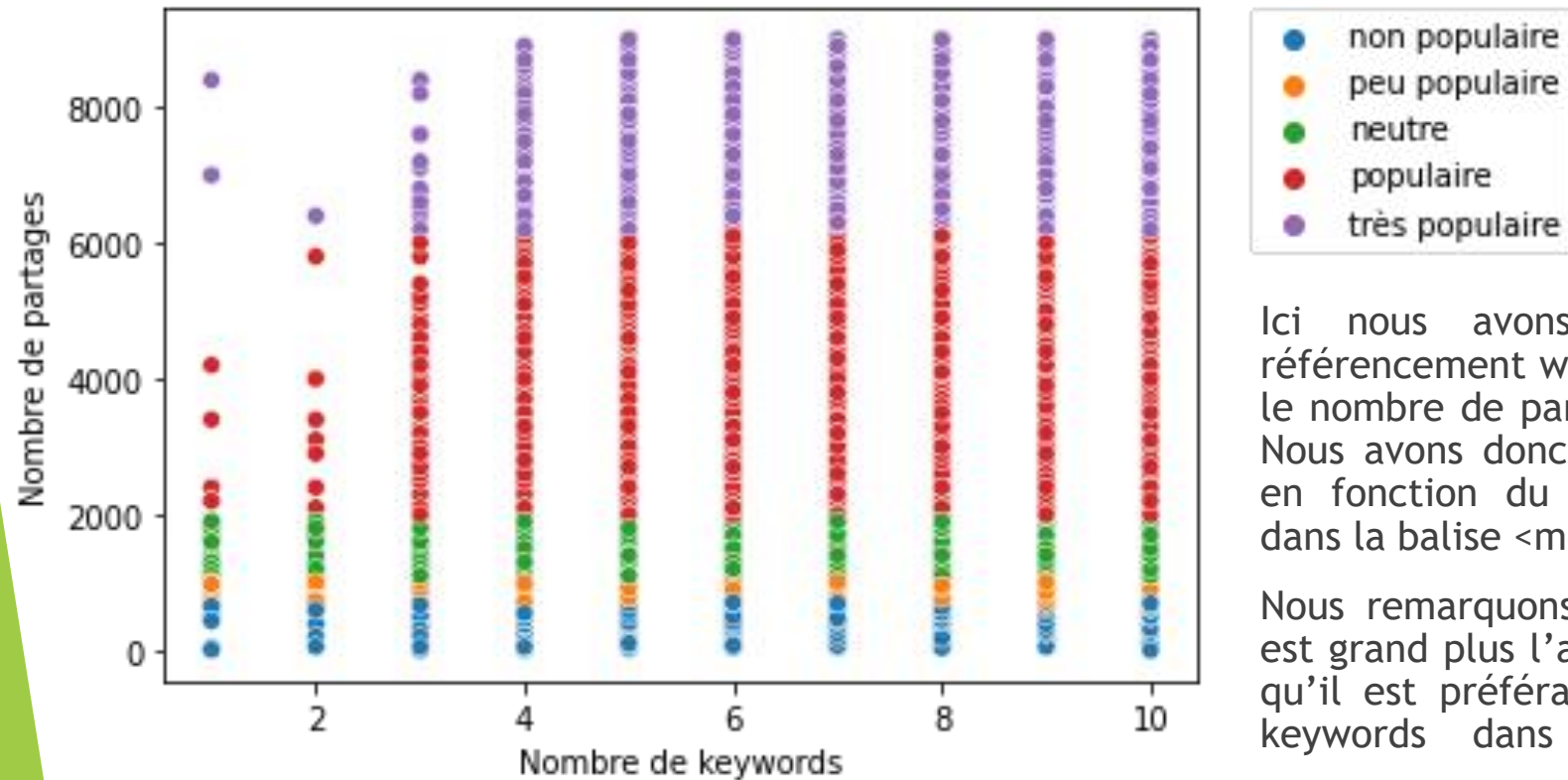
Distribution des articles en fonction de leur popularité et catégorie



Nous avons effectué une analyse similaire à la précédente mais cette fois-ci en fonction de la catégorie de l'article afin d'avoir pour chacune des catégories la proportion des articles publiés pour chaque classe (populaire, neutre ...)

Et ici on remarque que 3 catégories ressortent, Social Media avec près de 40% d'articles populaires, suivi par Tech et World.

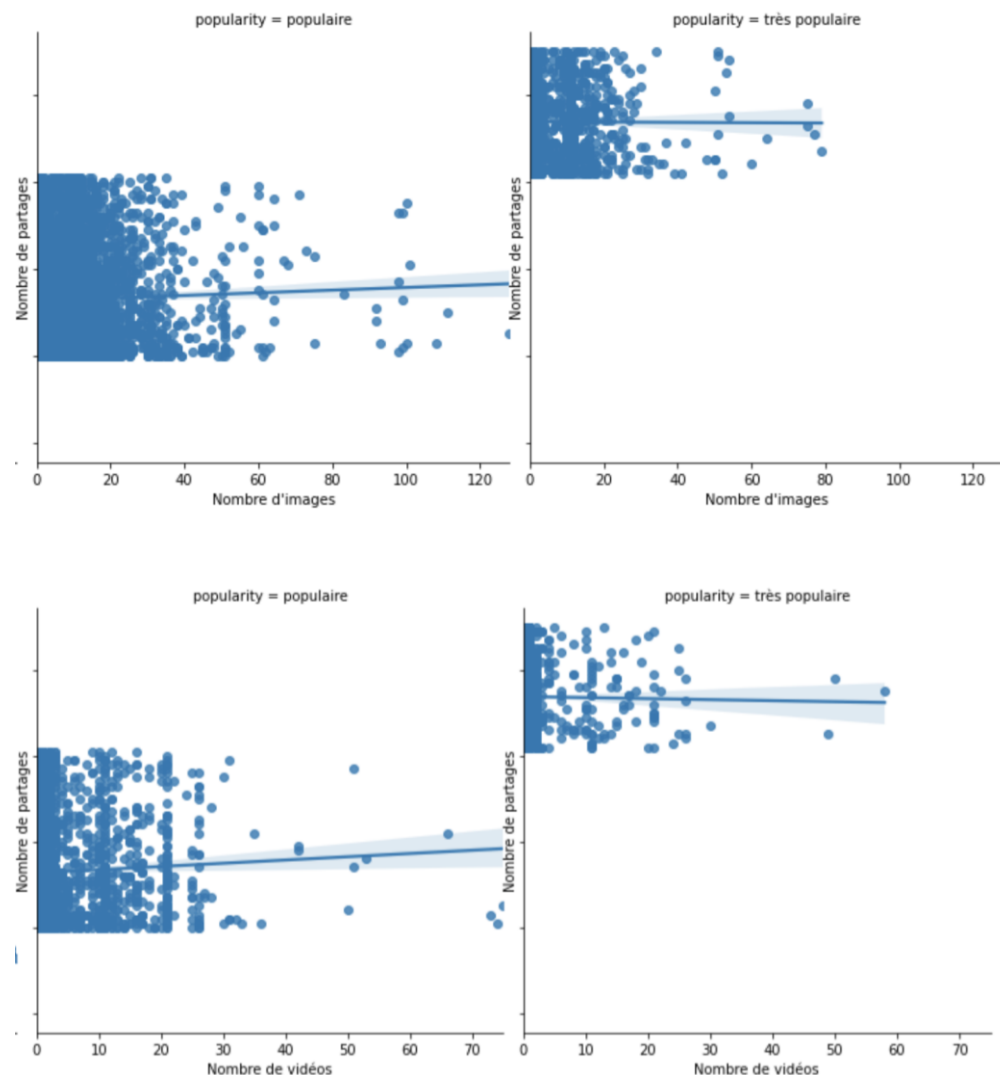
Popularité des articles en fonction du nombre de keywords dans la balise html <meta>



Ici nous avons voulu voir si le référencement web avait un impact sur le nombre de partages pour un article. Nous avons donc étudié leur évolution en fonction du nombre de keywords dans la balise <meta> de l'article.

Nous remarquons que plus ce nombre est grand plus l'article est populaire et qu'il est préférable d'avoir plus de 3 keywords dans la balise <meta>.

Popularité des articles en fonction du nombre d'images et vidéos



Pour finir nous avons voulu voir si le nombre d'images et vidéos à un impact sur la popularité de l'article.

Nous remarquons que plus ce nombre est petit plus l'article est populaire et qu'il est préférable d'avoir moins de 20 images et 3 vidéos dans l'article.

Conclusion – Visualisation des données

Nos recommandations afin de maximiser la popularité d'un article :

- L'article doit contenir moins de 3000 mots
- Le titre doit contenir entre 5 et 16 mots
- Le nombre d'images utilisé doit être inférieur à 20
- Utiliser un maximum de 3 vidéos
- Mettre moins de 40 liens externes dans l'article
- La balise <meta> de l'article doit contenir un minimum de 3 keywords
- Publié l'article de préférence le week-end
- De préférence publié un article dans les catégories Social media, Tech ou encore world



► Standardisation des données

Standardisation des données

Avant de démarrer la mise en place des modèles nous devons tout d'abord standardiser nos données afin de s'assurer d'avoir les meilleures performances possibles. Cependant toutes les colonnes ne sont pas à standardiser, en effet les colonnes issues de l'encodage one hot tel que `week_days` ou encore `data_channels`, mais aussi la `target` car cela pourrait perturber les résultats, sachant que ces colonnes ne sont pas numériques.

Nous avons donc utilisé le `StandardScaler` fournis par `Sklearn` afin d'effectuer cette standardisation, et par la même occasion créer un nouveau data frame `data_set_std`.



► Modèles de prédiction

Préparation des données

Avant de démarrer la mise en place des modèles nous avons fait un split de nos des données. Pour cela nous avons utilisé notre variable `data_set` et non pas la copie (`data_vis`) utilisé pour l'étape précédente. Utilisé `data_set` nous a permis de garder l'encodage one hot pour les colonnes `week_days` et `data_channels`.

Ce data frame a été split en 2 data frames :

- X : contenant les 58 colonnes prédictives
- Y : contenant la nouvelle colonne popularity

Par la suite ces deux data frames ont été splitter à leur tour en 2, afin de créer une partie train et test pour les futurs modèles. Ce split a été effectué en suivant la ration 67% pour le jeu d'entraînement et 33% pour celui de test.

Les modèles

Pour ce problème de classification, nous avons choisi de mettre en place 4 modèles de ML avec 3 approches différentes:

- Decision Tree : Un modèle simple très pratique pour les problèmes de classification, facile à visualiser et à comprendre pratique pour le feature engineering.
- Random Forest : Permet de voir si nous pouvons améliorer les performances des Decision Tree, et de limiter l'over-fitting
- XGBoost : Afin de voir si le boosting nous offre de meilleure performance que les Decision Tree
- K-Nearest Neighbor : Nous voulions voir si une approche de clustérisations serait plus performante que les Decision Tree

Nous avons délibérément mis de côté les régressions linéaire et logistique, car au vue de la distribution des données, ces modèles risquent de ne pas converger ou mal converger.

Premiers résultats

Dans un premier temps, nous avons voulu tester les performances de chacun de ces modèles avec les hyper-paramètres par défaut afin d'avoir un premier aperçu des résultats que l'on peut espérer.

	DecisionTreeClassifier	RandomForestClassifier	XGBClassifier	KNeighborsClassifier	type
0	0.285197	0.365792	0.358387	0.311589	default

D'après ces premiers résultats, nous pouvons d'ores et déjà voir que 2 modèles sont au coude à coude, Random Forest et XGBoost avec des performances très similaire.

Bien que les modèles Decision Tree et KNN sont moins performant que les autres nous allons tout de même essayer de les améliorer.

Decision tree tuning

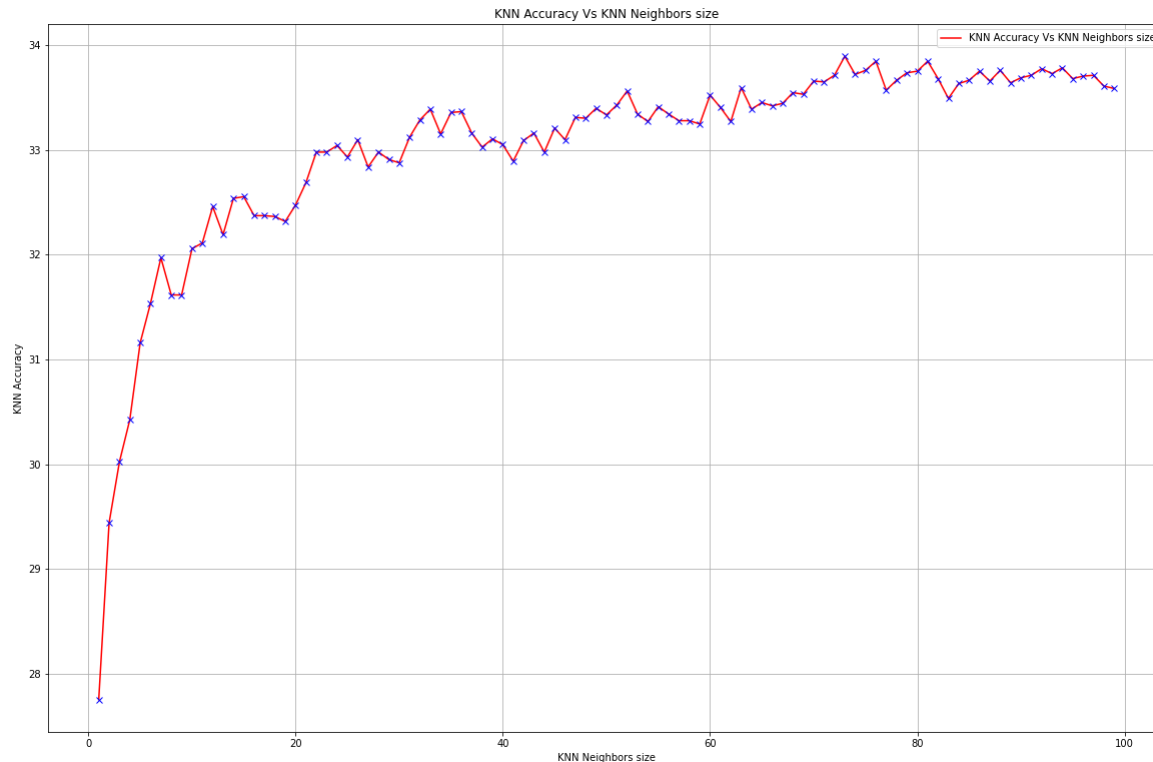
Dans un second temps, nous sommes passés à essayer d'améliorer les modèles en modifiant les hyper paramètres. Nous avons commencé par le Decision Tree classifier, nous avons utilisé la classe GridSearchCV afin de mettre en place une grille de paramètres et tester un maximum de valeurs possibles, afin de maximiser nos chances d'améliorer le modèle.

Grâce à cette grille nous avons réussi à monter notre modèle à 36,28% d'accuracy, en utilisant les paramètres suivants :

```
{'min_impurity_decrease': 0.001,  
 'max_depth': 8,  
 'max_features': None,  
 'criterion': 'entropy'}
```

Nous allons maintenant faire la même chose avec les autres modèles.

KNN hyper-parameter tuning

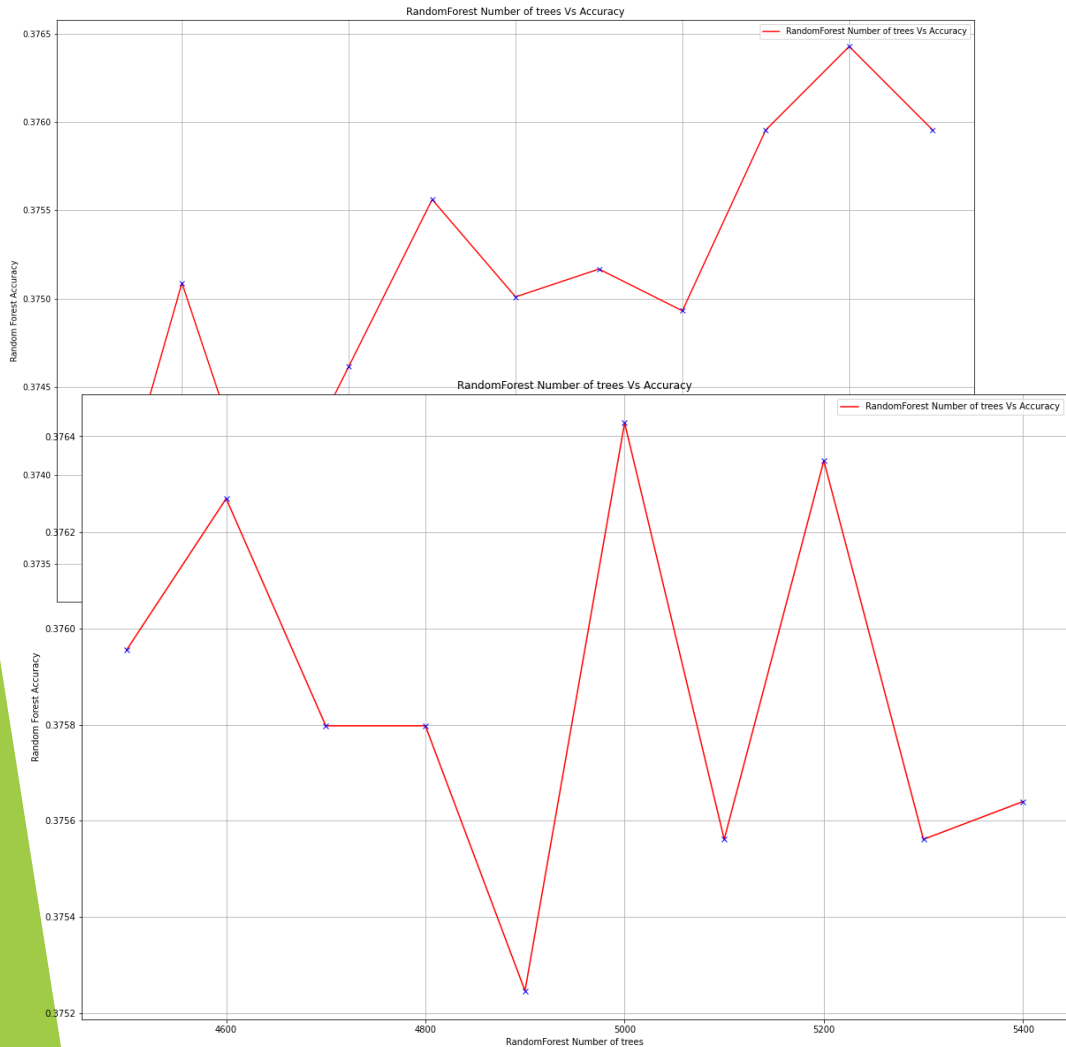


Dans le cas du K-Nearest Neighbor modèle, nous avons décidé de jouer sur la taille du modèle afin de pouvoir améliorer les performances.

Nous pouvons voir que le maximum d'accuracy que nous avons réussi à atteindre avec ce modèle est de 33,89 %.

Bien qu'au départ il semblait être plus performant que le Decision Tree finalement avec les bons paramètres le Decision Tree à de meilleures performances.

Random Forest hyper-parameter tuning



Contrairement au Decision Tree, pour le Random Forest nous avons pris la décision d'optimiser seulement 1 paramètre :

- Nombre d'arbres : 4980

Ce choix est du au faite qu'en essayant de modifier les autres paramètre nous nous somme rendu compte que l'accuracy du modèle baissait. En fixant ce paramètre à 4980 nous avons réussi à atteindre une précision de 37,65 %.

XGBoost hyper-parameter tuning

Pour le modèle XGBoost nous avons voulu jouer sur 4 paramètres :

- Learning_rate : 0.1
- Max_depth : 8
- Min_child_weight : 3
- colsample_bytree : 0.6

Ce qui nous a permis d'amener notre modèle à 37,49% d'accuracy ce qui est juste un peu plus faible que le modèle Random Forest.

Conclusion – Modèles de prédiction

Nos résultats:

	DecisionTreeClassifier	RandomForestClassifier	XGBClassifier	KNeighborsClassifier	type
0	0.285197	0.365792	0.358387	0.311589	default
1	0.362877	0.376586	0.374931	33.8927	optimisé

Bien que les résultats ne sont pas satisfaisants, on peut voir qu'après optimisation la précision de nos modèles c'est nettement amélioré, jusqu'à 7% d'amélioration pour le Decision Tree. Cependant ce n'est pas assez suffisant, pour que ces modèles soient utilisables.

Pour les améliorer il faudrait approfondir notre travail sur la compréhension des données et leur traitement en amont, comme par exemple essayer de supprimer les valeurs qui semble aberrante. Cependant aux vues du nombre de projets que nous avons à finaliser, nous n'avons pas pu accorder autant de temps que nous le voulions à cet aspect du projet.



► Exposition du Modèle (API)

Mise en place de l'API

Une fois les modèles optimisés il faut les rendre facilement utilisables, accessibles. Pour ce faire nous avons mis en place une api permettant d'accéder au modèle avec la meilleure accuracy.

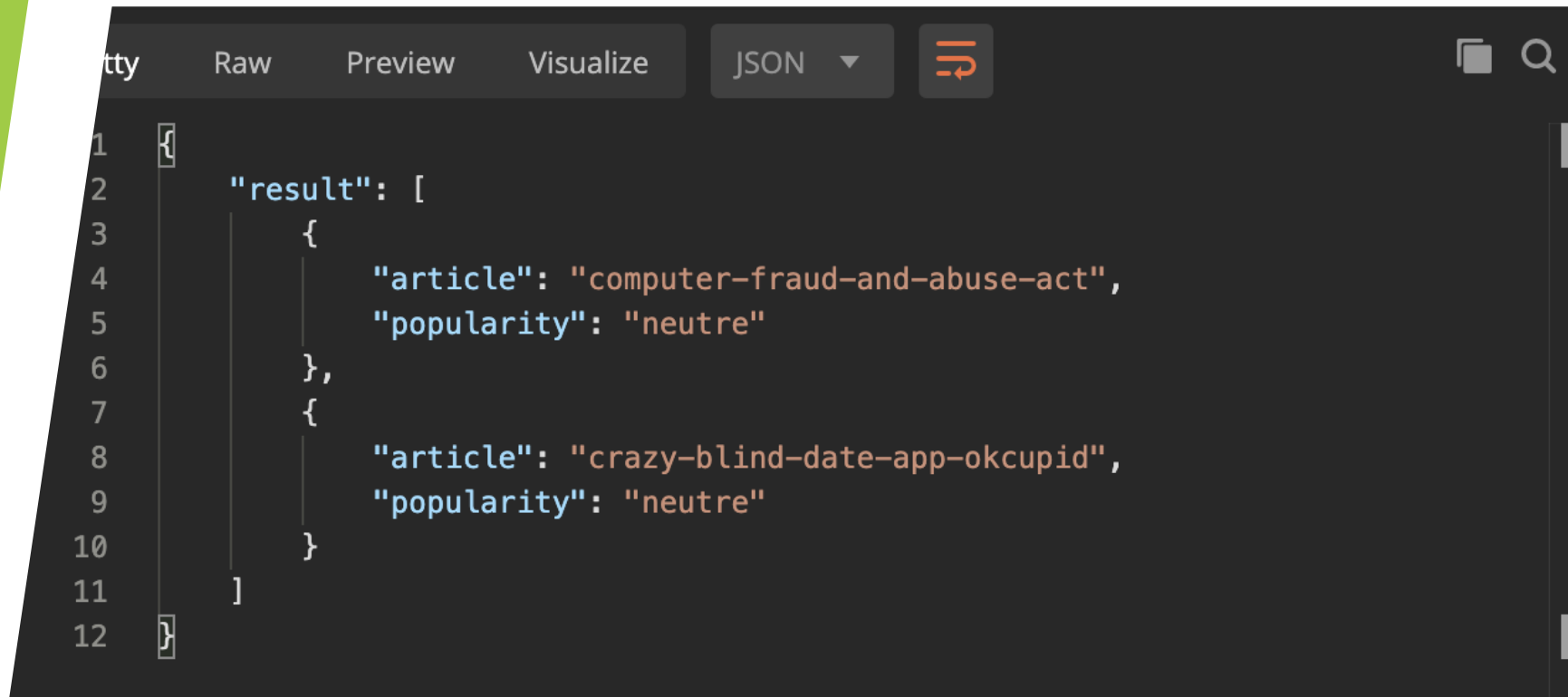
Cette API permet de récupérer la liste des articles disponibles dans notre jeu de données. L'utilisateur recevra la liste des titres et il devra choisir l'un d'entre eux ainsi le modèle prédira sa popularité. La liste des titres a été créée à partir de l'url de chacun des articles en gardant la dernière partie du path.

L'API qui a été mise en place en utilisant flask est donc constituée de 3 End-point :

- / : permet de vérifier que l'api est accessible
- /articles : permet de récupérer la liste des articles via une requête GET
- /popularity : permet de prédire la popularité d'un article ou une liste d'articles

Nous avons par la suite testé l'api à l'aide de l'outil PostMan.

Nous avons également pensé à une évolution de l'api, celle-ci permettrait de prendre en paramètre le lien d'un article Marshable, afin de pouvoir le scraper, effectuer l'analyse NLP afin de pouvoir récupérer les champs nécessaires à la prédiction. Cependant comme on peut l'imaginer la mise en place d'une telle évolution aurait demandé un temps conséquent.

A screenshot of a web application's JSON viewer. The interface has a dark theme with a top bar containing tabs for 'Raw', 'Preview', and 'Visualize', a 'JSON' dropdown menu, and a refresh icon. The main area displays a JSON object with a 'result' array containing two items. Each item is an object with 'article' and 'popularity' fields. The first item has 'computer-fraud-and-abuse-act' and 'neutre'. The second item has 'crazy-blind-date-app-okcupid' and 'neutre'. Line numbers 1 through 12 are visible on the left side of the code editor.

```
1  {}  
2    "result": [  
3      {  
4        "article": "computer-fraud-and-abuse-act",  
5        "popularity": "neutre"  
6      },  
7      {  
8        "article": "crazy-blind-date-app-okcupid",  
9        "popularity": "neutre"  
10     }  
11   ]  
12 }
```

Résultats de l'API



► Conclusion

Conclusion

Après notre étude, nous avons noté plusieurs points qui permettent d'améliorer la popularité d'un article, notamment sa taille, le nombre de keywords dans la balise <meta> ou encore la catégorie de l'article (social media, Tech ...).

Bien que ces informations nous est été très utile, l'ensemble des modèles que nous avons mis en place n'était pas très performant même après optimisation. Afin de les améliorer, il faudrait passer plus de temps sur le traitement de nos données en amont afin d'optimiser nos résultats, notamment grâce à la suppression des lignes produisant du bruit, et qui dérange la prédiction (valeur trop grande, pas assez).

Et finalement pour rendre notre modèle utilisable nous l'avons exposé à l'aide d'une API Flask permettant de prédire la popularité d'un ou plusieurs article de notre jeu de données.