PYTHON FOR DATA ANALYSIS

APROH Louise | AMINE Zayneb

Article Scientifique

VIDEO TRANSCODING TIME PREDICTION FOR PROACTIVE LOAD BALANCING

Sci-Hub | Video transcoding time prediction for proactive load balancing. 2014 IEEE International Conference on Multimedia and Expo (ICME) | 10.1109/ICME.2014.6890256 (sci-hub.do)

- Objectif 1er de l'article : présenter un modèle qui permet de prédire le temps de transcodage des vidéos en fonctions d'un flux de vidéo d'entrée et de ses paramètres de transcodage.
- Notre étude : tester des modèles de ML pour la prédiction du temps de transcodage d'une vidéo en fonction, entre autre, de ses paramètres de transcodage.

ABSTRACT

In this paper, we present a method for predicting the transcoding time of videos given an input video stream and its transcoding parameters. Video transcoding time is treated as a random variable and is statistically predicted from past observations. Our proposed method predicts the transcoding time as a function of several parameters of the input and output video streams, and does not require any detailed information about the codec used. We show the effectiveness of our method via comparing the resulting predictions with the actual transcoding times on unseen video streams. Simulation results show that our prediction method enables a significantly better load balancing of transcoding jobs than classical load balancing methods.

Données

Online Video Characteristics and Transcoding Time Dataset Data Set

- https://archive.ics.uci.edu/ml/datasets/Online+Video+Characteristics+and+Transcoding+Time+Dataset
- Le dataset présenté est composé de deux fichiers tsv nommés 'youtube_videos.tsv' et «transcoding_mesurment.tsv».
- Le premier fichier contient 10 colonnes de caractéristiques fondamentales de 1,6 million de vidéos youtube
- Le deuxième fichier de notre dataset contient 20 colonnes qui incluent les caractéristiques d'entrée et de sortie de vidéos ainsi que leur transcodage , les besoins en temps et en ressources de mémoire lors du transcodage des vidéos.

Problématique

L’objectif de notre analyse est de prédire la durée de transcodage des vidéos Youtube du dataset.

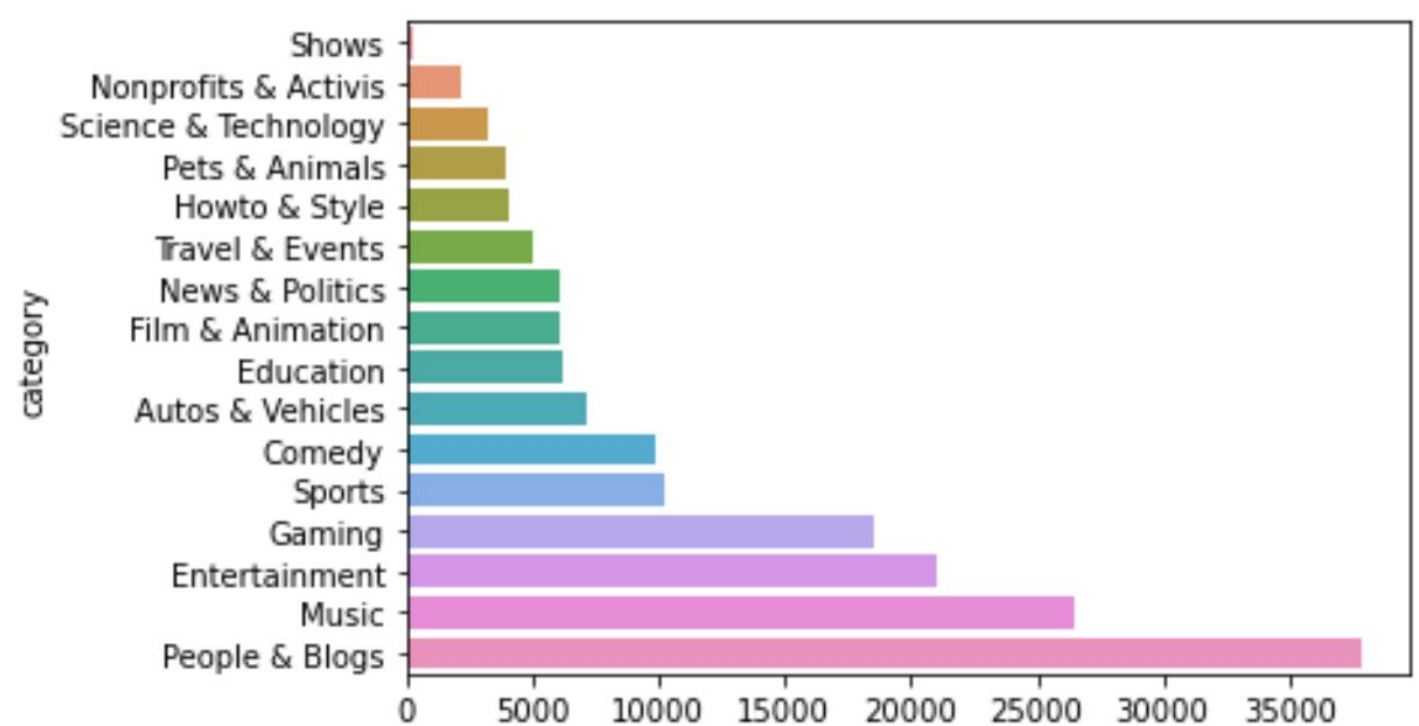
- Cette donnée correspond à la variable utime du dataset
- Notre approche est celle de résolution d’un problème de régression.

Visualisation des données et analyse

Partie 1 : Observation générale sur les vidéos du dataset 'youtube_videos.tsv'

Observation : pas de valeurs NA dans le dataset entier

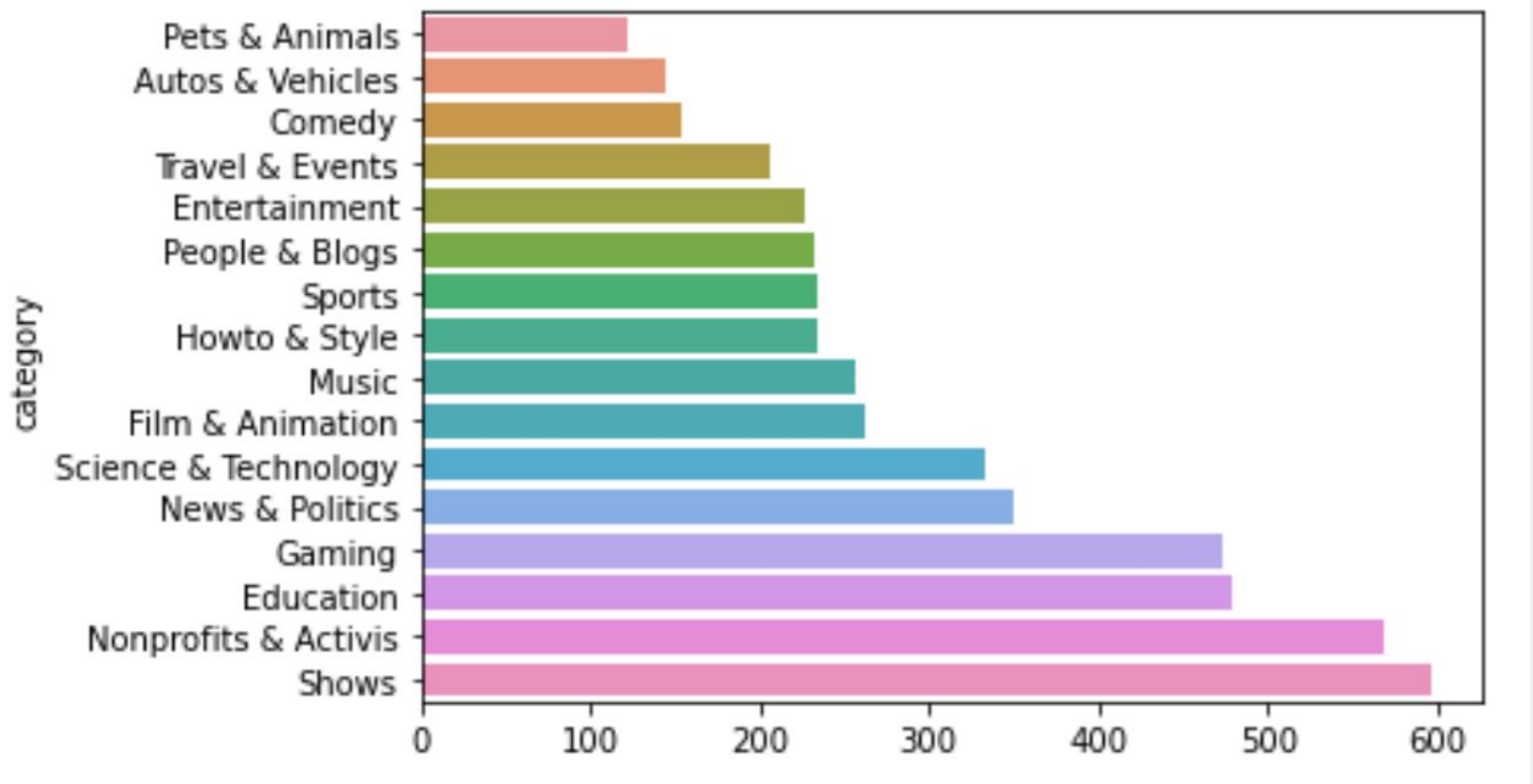
- Quelles sont les types de vidéos les plus vues et le moins vues?



Visualisation des données et analyse

Partie 1 : Observation générale sur les vidéos du dataset 'youtube_videos.tsv'

- Y a t'il des catégories de vidéos qui sont en moyenne plus ou moins longues?

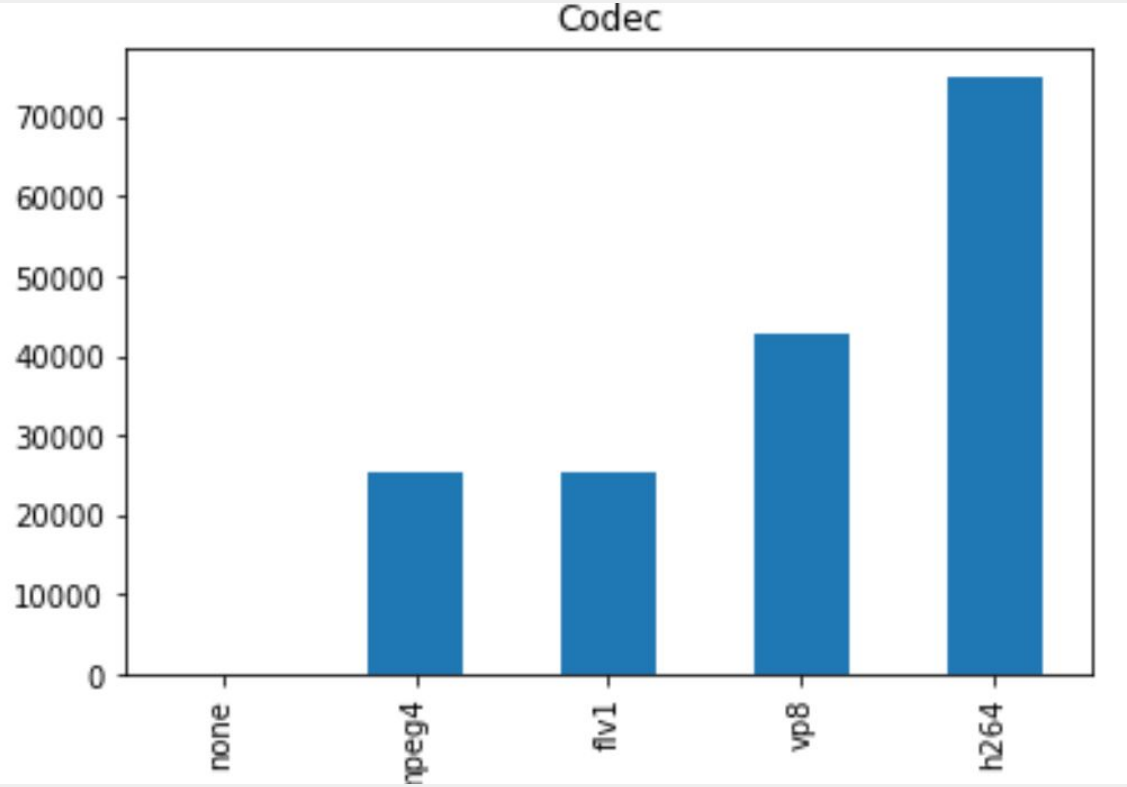


Visualisation des données et analyse

Partie 1 : Observation générale sur les vidéos du dataset 'youtube_videos.tsv'

- Quel est le type de codec le plus représenté/le moins représenté dans les vidéos Youtubes ?

  + représenté : h264



Visualisation des données et analyse

Partie 2 : Merge des 2 datasets (dfTranscoding et dfToutubeVideo)

But de la fusion : avoir des données complètes sur les vidéos étudiées. (caractéristiques et informations de transcodage)

Dataset Final enregistré dans un dataframe :

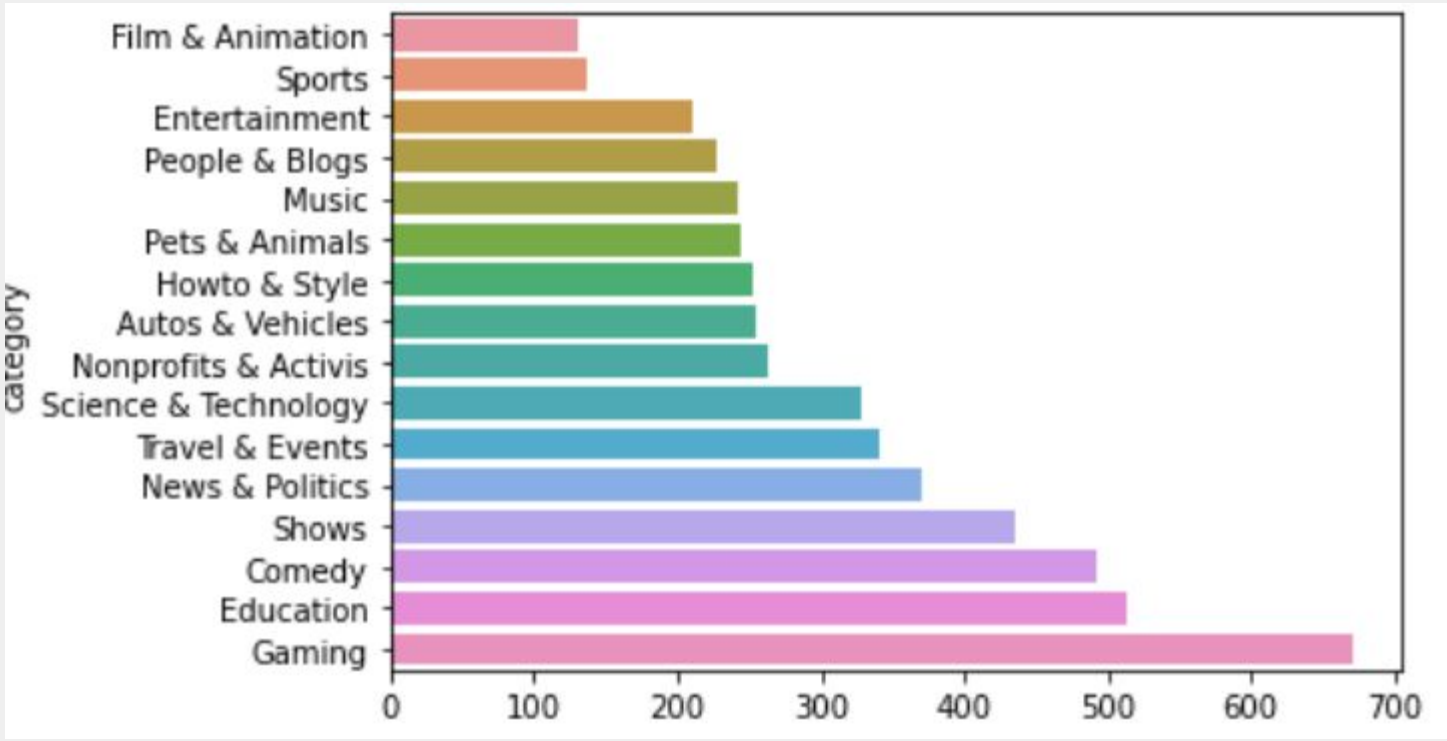
	id	duration	codec	width	height	bitrate	framerate	i	p	b	...	b_size	size	o_codec	o_bitrate	o_framerate	o_width	o_height	ume
0	04t6-jw9czg	130.35667	mpeg4	176	144	54590	12.0	27	1537	0	...	0	889537	mpeg4	56000	12.0	176	144	225(
1	04t6-jw9czg	130.35667	mpeg4	176	144	54590	12.0	27	1537	0	...	0	889537	mpeg4	56000	12.0	176	144	225(
2	04t6-jw9czg	130.35667	mpeg4	176	144	54590	12.0	27	1537	0	...	0	889537	mpeg4	56000	12.0	176	144	225(
3	04t6-jw9czg	130.35667	mpeg4	176	144	54590	12.0	27	1537	0	...	0	889537	mpeg4	56000	12.0	176	144	225(
4	04t6-jw9czg	130.35667	mpeg4	176	144	54590	12.0	27	1537	0	...	0	889537	mpeg4	56000	12.0	176	144	225(

5 rows × 23 columns

Visualisation des données et analyse

Partie 3: Observation sur les mesures de transcodage des vidéos Youtubes du dataset mergé

- Dans cet échantillon, quels sont en moyenne les catégories de vidéos les plus/moins longues?

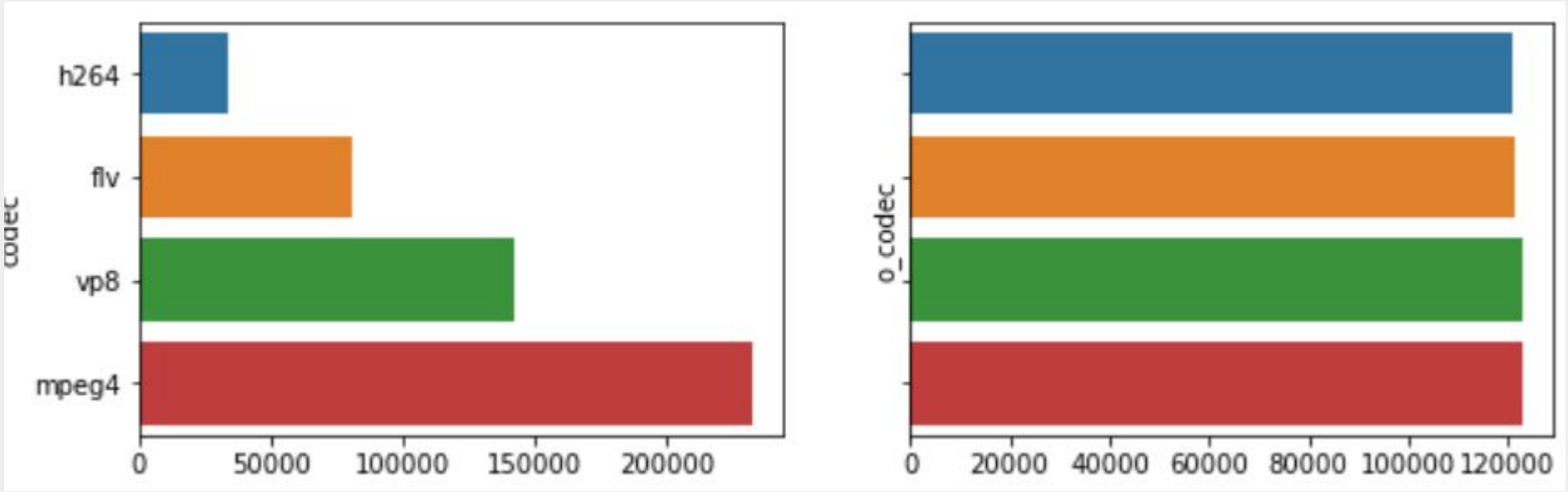


Visualisation des données et analyse

Partie 3: Observation sur les mesures de transcodage des vidéos Youtubes du dataset mergé

- Quels sont les types de codec les plus représentés/les moins représentés avant et après transcodage?

  On déduit que certaines vidéos changent de type de codec après transcodage

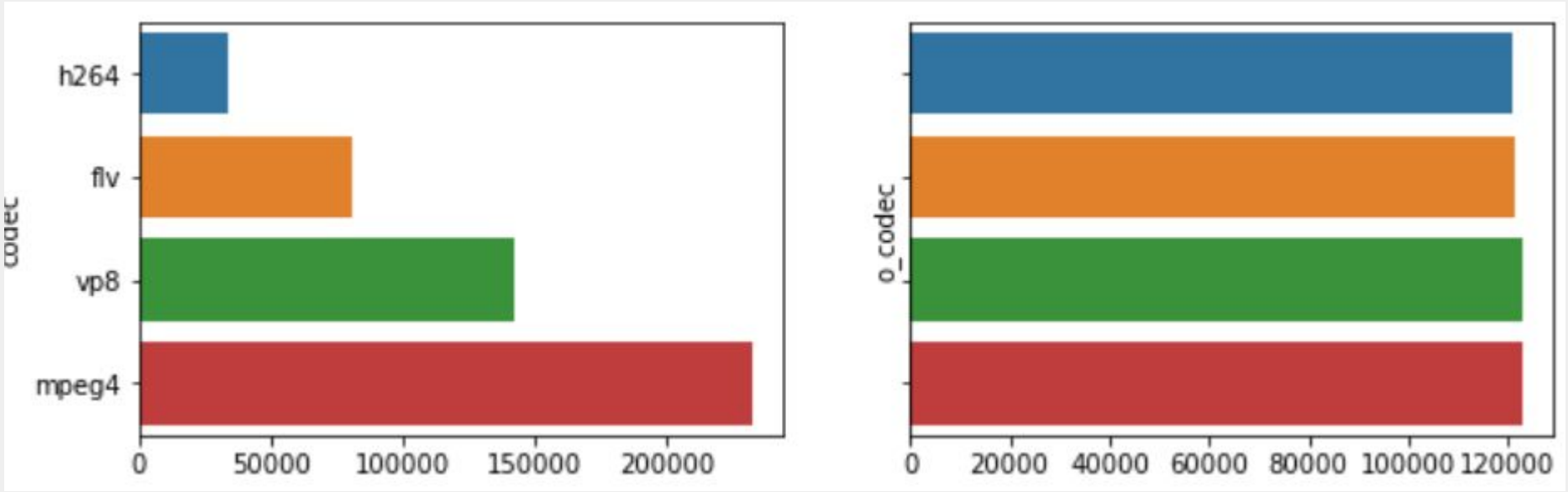


Visualisation des données et analyse

Partie 3: Observation sur les mesures de transcodage des vidéos Youtubes du dataset mergé

- Quelle est la répartition du temps de transcodage ?

  On remarque que que 75% des valeur de temps de transcodage se concentrent à moins d'une dizaine de secondes et on remarque aussi que quelques valeurs vont de 25 à peu près 230 sec.

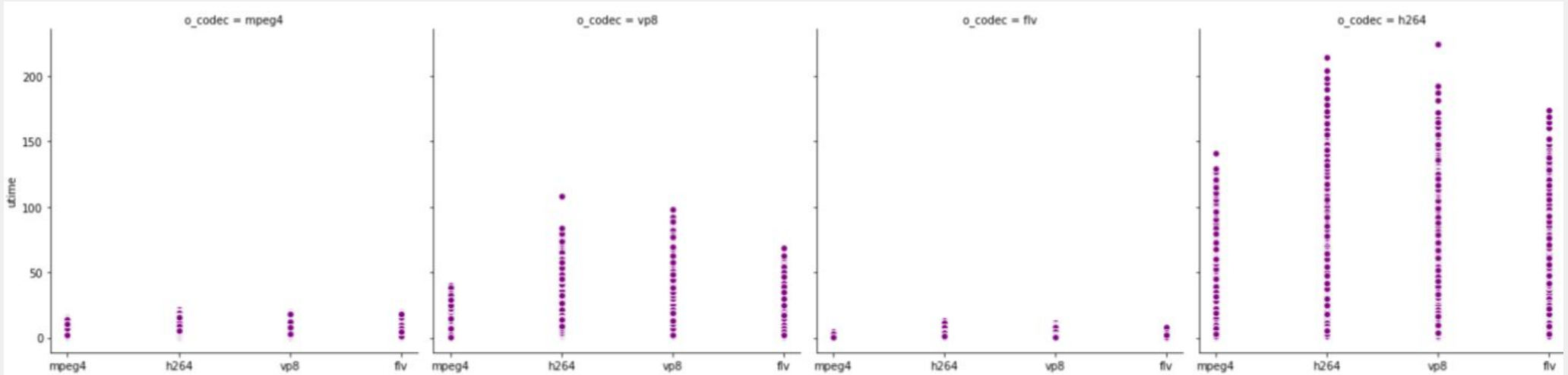


Visualisation des données et analyse

Partie 3: Observation sur les mesures de transcodage des vidéos Youtubes du dataset mergé

- Le codec d'entrée ou de sortie a t il une influence sur le temps le transcodage d'une vidéo?

  On remarque pour les vidéos étudiées, le temps de transcodage peut monter assez haut pour des vidéos avec un codec de sortie h264 quelque soit le codec d'entrée. Pour les codec de sortie flv et mpeg4 on remarque que les temps de transcodages sont assez bas quelque soit le codec d'entrée.



Visualisation des données et analyse

Partie 3: Observation sur les mesures de transcodage des vidéos Youtubes du dataset mergé

- Zoom sur la corrélation de la variable utime avec les autres variables

Les variables les plus corrélées avec Utime et qui vont sûrement jouer dans la prédiction :

Ume, O_width, O_height, bitrate, O_bitrate.

En correspondance avec l'article scientifique.

```
b              0.001768
duration       0.003637
i              0.019384
frames         0.033760
p              0.033875
i_size         0.075390
framerate      0.090951
o_framerate    0.104926
size           0.108972
p_size         0.109510
height         0.151514
width          0.152003
o_bitrate      0.154949
bitrate        0.177376
o_height       0.515267
o_width        0.518897
umem           0.658152
utime          1.000000
b_size              NaN
Name: utime, dtype: float64
```

Modèles

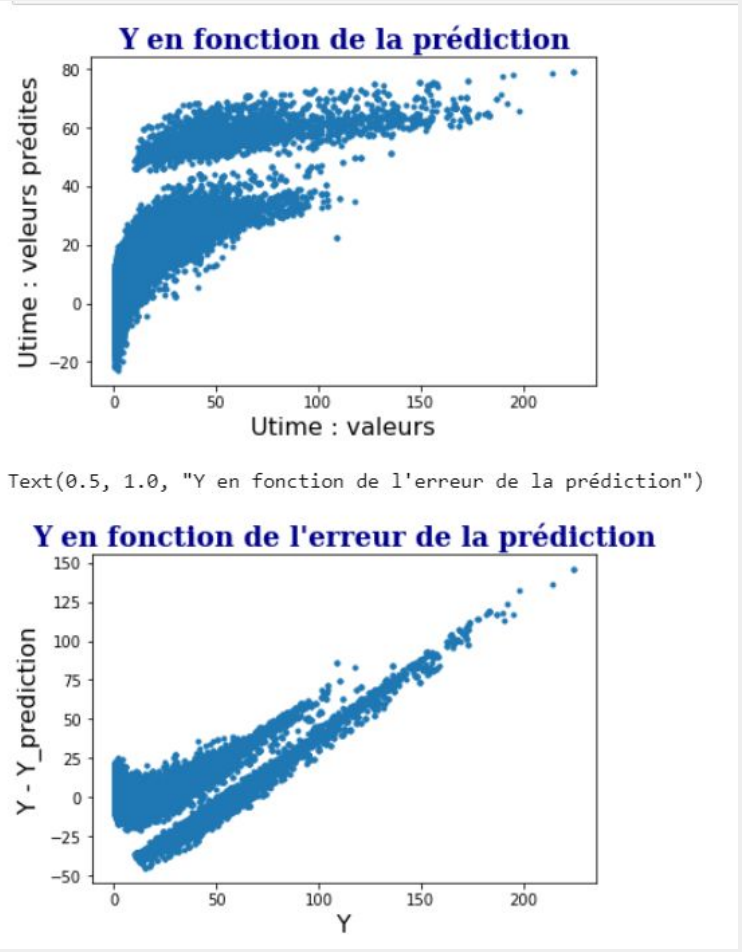
- Afin de prédire la variable de durée de transcoding Utime nous décidons d'appliquer plusieurs modèles de régression:
  - *RegressionLinéaire*
  - *RandomForest*
  - *GridSearch avec RandomForest*
  - *KNN*
  - *XGBOOST*

Préparation

- Les colonnes string sont transformé en valeurs binaire afin d'appliquer les modèles (one hot encoding)
- On sépare ensuite le dataset en deux X qui correspond au features et Y la target à prédire
- On sépare ensuite le dataset en train et test en 80% / 20%

Régression Linéaire

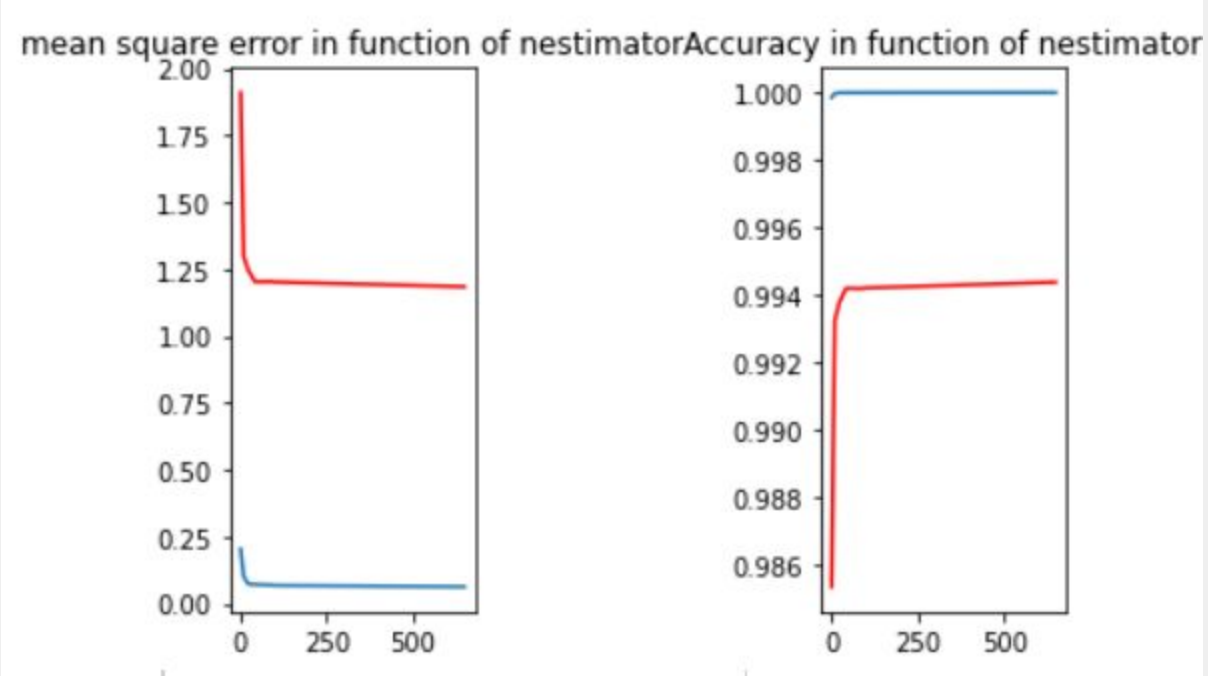
- Mean squared error: 89.66
- R^2= 0.66
- Nous remarquons que les prédictions ne sont pas excellentes même si celle-ci ne sont pas catastrophiques et même plutôt bien pour un modèle tel que celui-ci.




```
Text(0.5, 1.0, "Y en fonction de l'erreur de la prédiction")
```

RandomForest

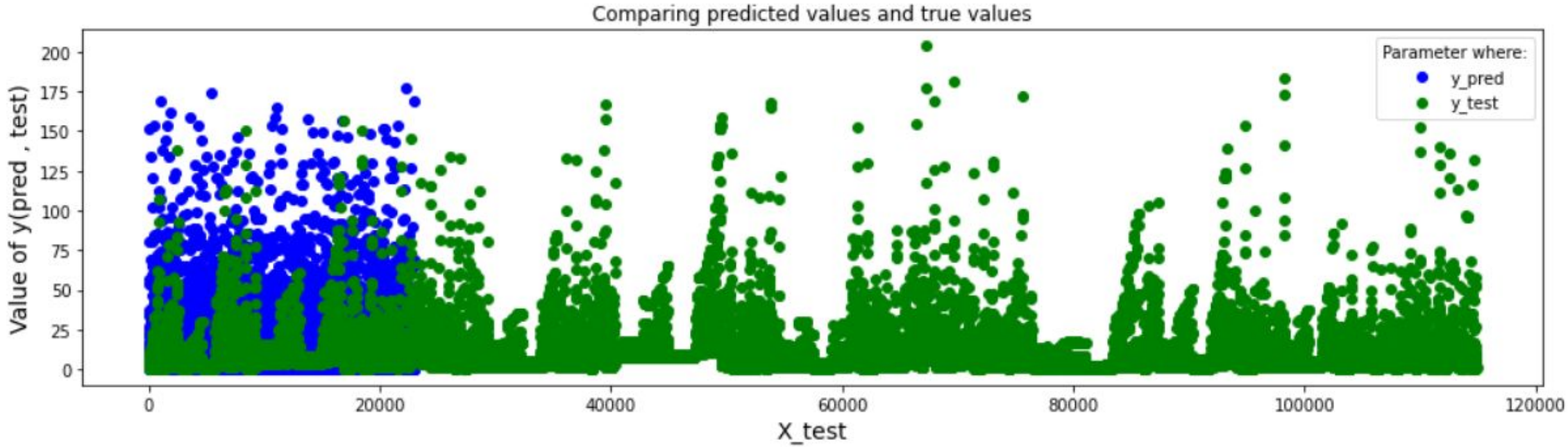
- Nous appliquons le modèle sur 7 valeurs différentes
- Nous exécutons RandomForestRegressor
- On choisi d'appliquer le modèle sans hyperparamètre puis avec.
- On calcul la mean_square_error ainsi que l'accuracy.
- Courbe rouge= avec hyper paramètres et bleu= sans hyper paramètres



Affichage des résultats

On affiche aussi les valeurs de Y_test et Y_pred afin de vérifier leur proximité

Les valeurs sont proches surtout si l'on considère qu'il s'agit de temps en seconde



Colonnes importantes

Nous déterminons les colonnes importantes au modèle afin de voir si on pourrait éventuellement améliorer encore plus le modèle.

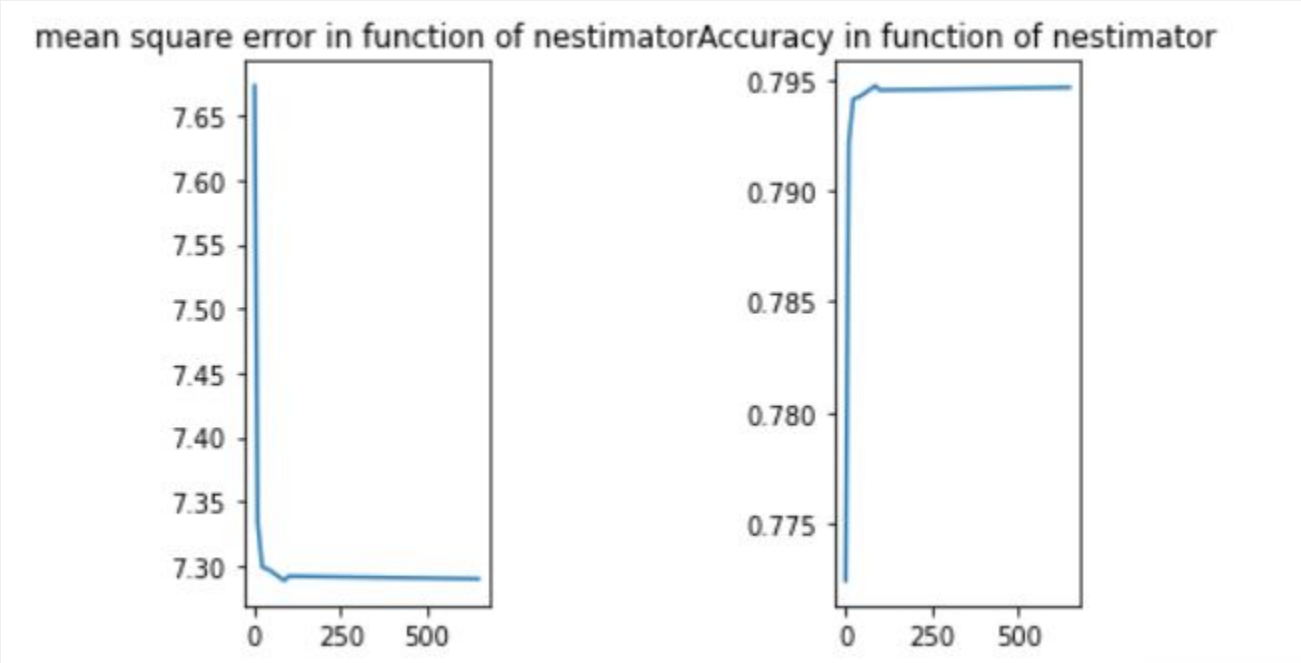
Nous n'avons pas gardé les colonnes string pour la prochaine prédiction afin de voir entre-autre si o_codec est une feature importante pour la prédiction

Nous ne garderons que les colonnes suivantes:

```
df_colonne_imp=dfTranscodingMesMerge[['umem','o_bitrate','p_size','width','b_size','size','frames','height','b']]
```

On relance le split en train et test et l'exécution du modèle pour les différents estimateurs

Affichage des résultats



- Le modèle random forest le plus efficace est le plus simple sans hyper paramètre et sans enlever les colonnes string. On peut en déduire que notamment o_codec peut avoir un impact dans la prédiction.

GridSearchCV avec RandomForest

- Dans l'objectif d'améliorer le modèle random forest nous appliquons aussi l'algorithme GridSearchCV avec l'estimateur RandomForestRegressor.

- Les paramètres sont :

```
param_grid = {
        "n_estimators"      : [10,20,30],
        "max_features"      : ["auto", "sqrt", "log2"],
        "min_samples_split" : [2,4,8],
        "bootstrap": [True, False],
        }
```

- Nous appliquons le modèle 5 fois et calculons l'erreur avec MeanSquareError et nous affichons aussi l'accuracy.

```
15:52:04
Loop:  0
--------------
ok
mean square error score: 1.2672849274494338
Best Score: 0.9935543611072077
Best params: {'bootstrap': True, 'max_features': 'auto', 'min_samples_split': 2, 'n_estimators': 30}
Loop:  1
--------------
ok
mean square error score: 1.2748322650684127
Best Score: 0.9936409308624319
Best params: {'bootstrap': True, 'max_features': 'auto', 'min_samples_split': 2, 'n_estimators': 30}
Loop:  2
--------------
ok
mean square error score: 1.35705248118415
Best Score: 0.9937369401830137
Best params: {'bootstrap': True, 'max_features': 'auto', 'min_samples_split': 2, 'n_estimators': 30}
Loop:  3
--------------
ok
mean square error score: 1.329050971367915
Best Score: 0.9935942529399309
Best params: {'bootstrap': True, 'max_features': 'auto', 'min_samples_split': 2, 'n_estimators': 30}
Loop:  4
--------------
ok
mean square error score: 1.2650850817355737
Best Score: 0.9936191542901828
Best params: {'bootstrap': True, 'max_features': 'auto', 'min_samples_split': 2, 'n_estimators': 30}
17:37:10

Duration time : 6305.202346086502
```

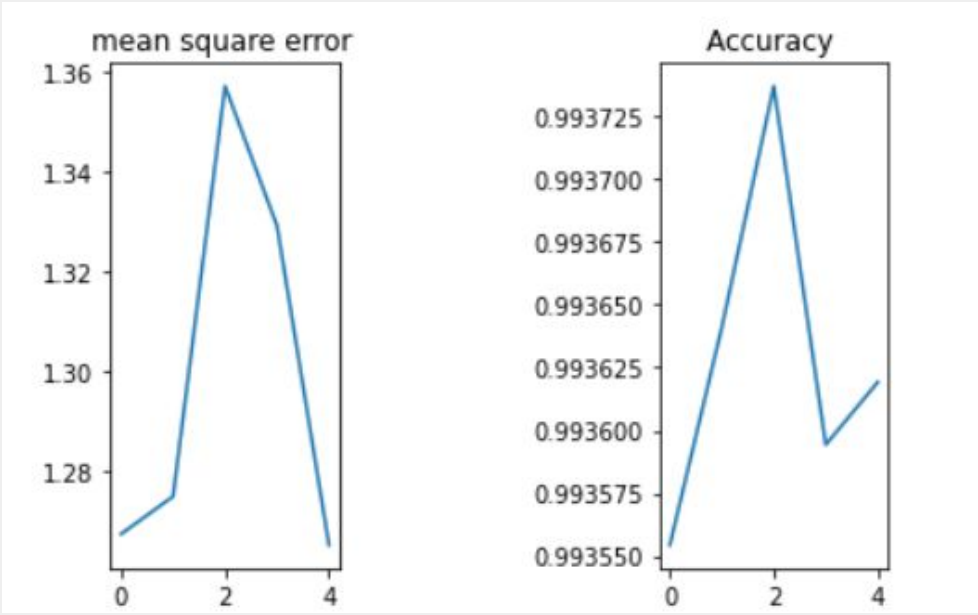
Résultat

- Les résultats ne sont pas les plus performants
- Meilleure solution obtenue avec un score

R^2 = 0.9936 et un MSE= 1.2651 avec les paramètres suivants : bootstrap':

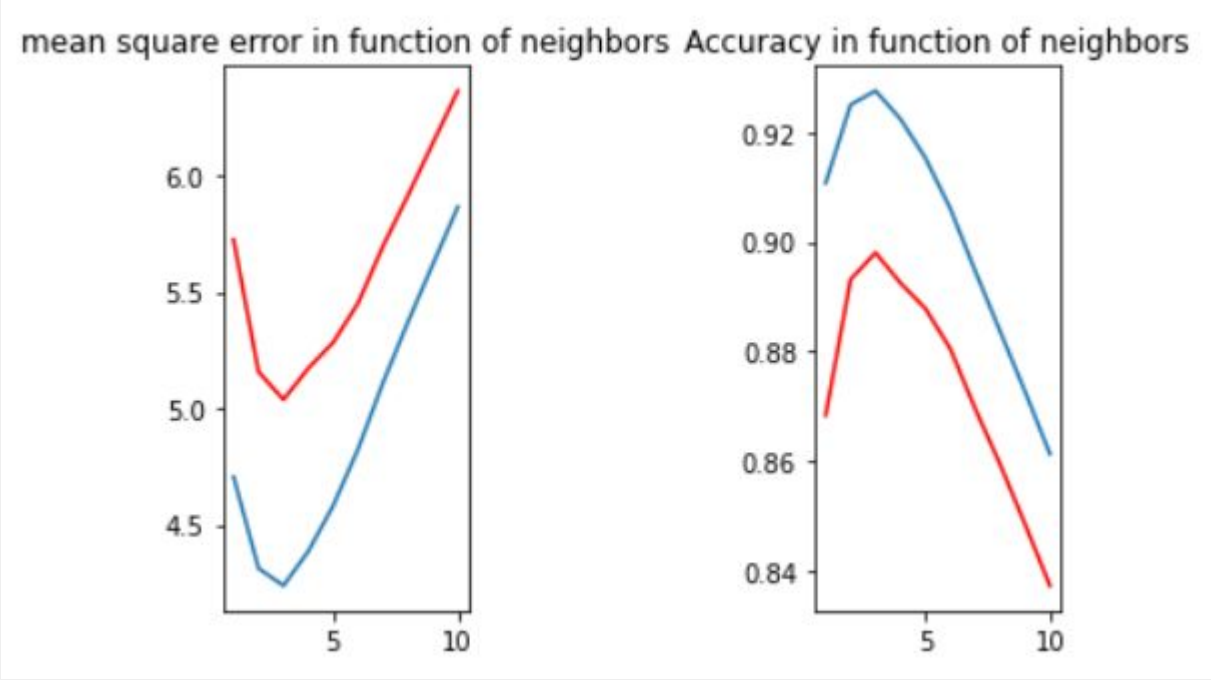
- True, '
- max_features': 'auto',
- 'min_samples_split': 2,
- 'n_estimators': 30

- On remarque que ce socre est asssez proche du Random Forest de départ. La grid search n'était peut-être pas nécessaire car les résultats de base étaient déjà très concluants.



KNN

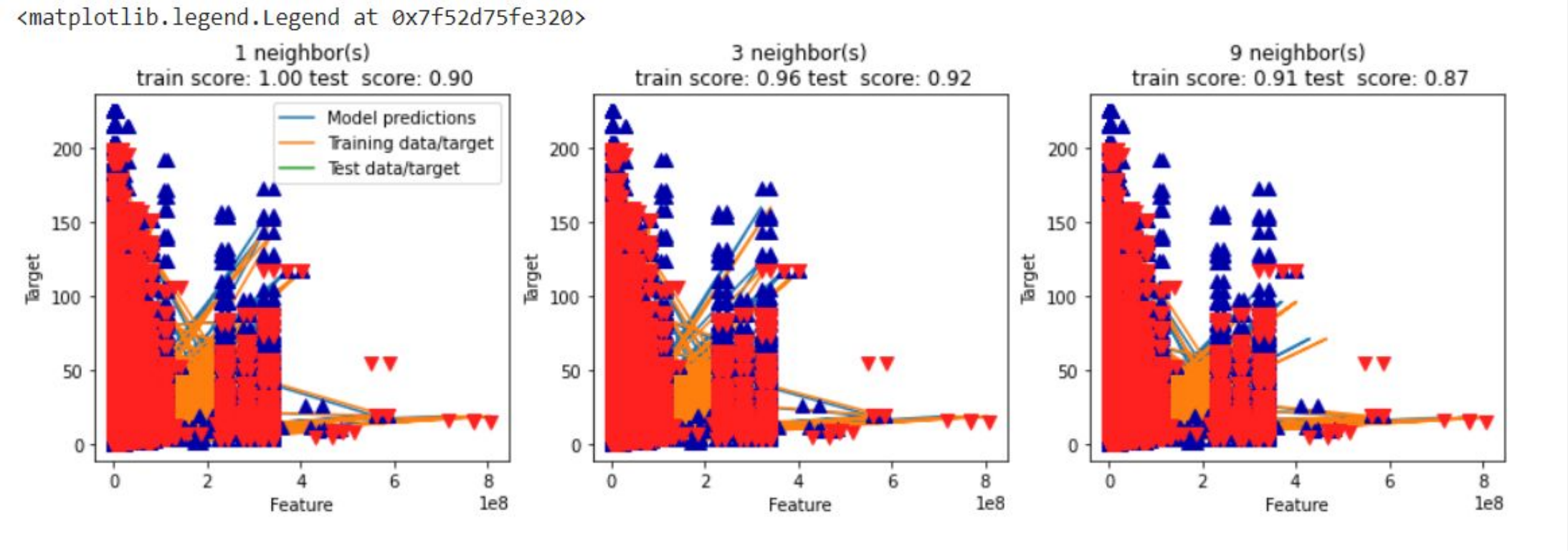
- On choisi d'appliquer le modèle K Neighbors aux données. On fait varier le nombre de voisin de 1 à 10.
- On choisi d'appliquer le modèle sans hyperparamètre puis avec.
- On calcul l'accuracy et la mean_square_error



- Le meilleur modèle est celui sans hyper paramètre

Affichage de KNN

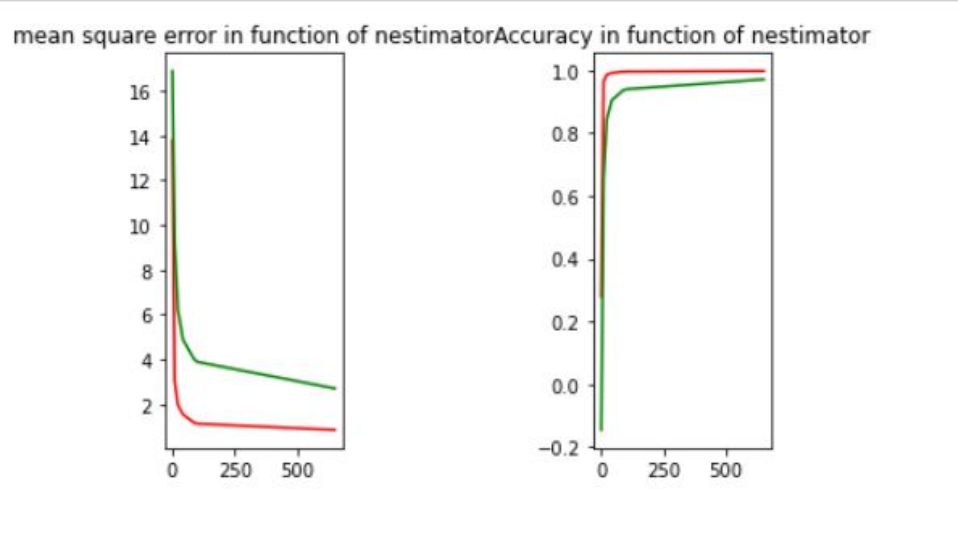
- On choisi aussi d'afficher pour trois prédictions différentes les données train les données test et les prédictions.



XGBOOST

- Le modèle XGBOOST s'applique de la même manière que random forest. On choisi donc pour estimateurs les mêmes que ceux du random forest afin de pouvoir les comparer plus simplement.

- On exécute le modèle avec et sans hyperparamètre, on affiche les deux afin de les comparer.



- On voit que le modèle avec les hyperparamètres est très proche de celui sans . Les résultats sont très proche de ceux du Random Forest.

Comparaison des modèles

- Nous créons un dataframe qui regroupe les accuracy et mean square error de chaque modèle afin de les comparer.

	MSE_KNN	MSE_KNN_IMP	MSE_RF	MSE_RF_IMP	MSE_XGBOOST	MSE_XGBOOST_IMP	ACC_REG_LIN	ACC_KNN	ACC_KNN_IMP	ACC_RF	ACC_RF_IMP	ACC_XGBOOST	ACC_XGBOOST_IMP
0	4.705379	5.723793	0.203098	7.673617	16.883403	13.785029	0.660006	0.910760	0.868355	0.999845	0.772435	-0.145400	0.279399
1	4.312428	5.156466	0.099969	7.333466	9.304007	3.052728	NaN	0.925042	0.893158	0.999962	0.792163	0.652162	0.964661
2	4.238274	5.037453	0.072433	7.298822	6.220572	1.961938	NaN	0.927598	0.898033	0.999980	0.794122	0.844512	0.985403
3	4.386395	5.172113	0.070412	7.296592	4.883553	1.540756	NaN	0.922449	0.892509	0.999981	0.794248	0.904168	0.990998
4	4.582750	5.283112	0.068211	7.288132	4.019961	1.170765	NaN	0.915351	0.887845	0.999983	0.794725	0.935065	0.994802
5	4.826513	5.452988	0.066921	7.291618	3.885782	1.121773	NaN	0.906106	0.880517	0.999983	0.794528	0.939327	0.995228
6	5.111353	5.697504	0.061915	7.289496	2.698567	0.846935	NaN	0.894696	0.869561	0.999986	0.794648	0.970738	0.997280
7	5.371951	5.915111	NaN	NaN	NaN	NaN	NaN	0.883685	0.859407	NaN	NaN	NaN	NaN
8	5.619719	6.140544	NaN	NaN	NaN	NaN	NaN	0.872708	0.848487	NaN	NaN	NaN	NaN
9	5.865743	6.363895	NaN	NaN	NaN	NaN	NaN	0.861319	0.837264	NaN	NaN	NaN	NaN

Temps d'exécution

- On choisit aussi d’afficher les temps d'exécution de chaque modèle car c’est un paramètre important au choix du meilleur modèle.

```
tmp_LinRg=0.0334985256195068 36
tmp_rf=1023.0794982910156
tmp_rf_imp=143.7336151599884
tmp_rf_gridsearch=6305.202346086502 # Pour 5 valeur seulement
tmp_rf_knn=81.12048649787903
tmp_xgboost=180.3040008544922
```

- Le modèle le plus rapide est régression linéaire puis suit KNN

Conclusion

- Bon modèle pour notre exercice : Bonne prédiction + Temps d'exécution raisonnable.
- Meilleurs modèles testés : **XGBoost et le RandomForest** sans hyper paramètres.
- Axe d’amélioration en fonction de l'article : retirer la colonne Umem à l’entrée du modèle. L'article ne prend pas cette feature en compte dans son modèle (SVM) en entré.
- La problématique de départ tirée de l'article a été menée à bien dans notre projet