Projet FDA:
Video popularity prediction

Rida ENNAMIRI
Yebabah N'GORAN
Vanessa VANDENBERGHE

Sommaire

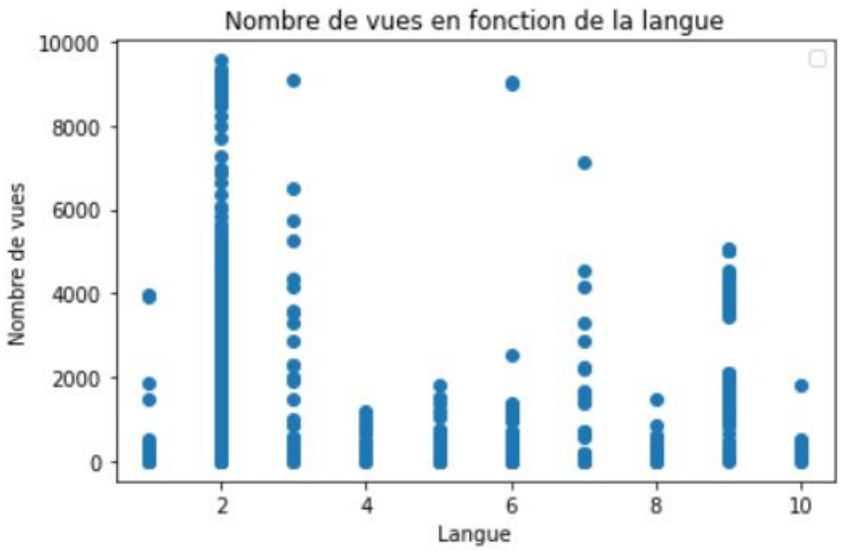
- Présentation du challenge
- Quelques visualisations des données
- Traitement des données
- Modèles classiques
- Autres modèles
- Résultats
- Conclusion

Présentation du Challenge

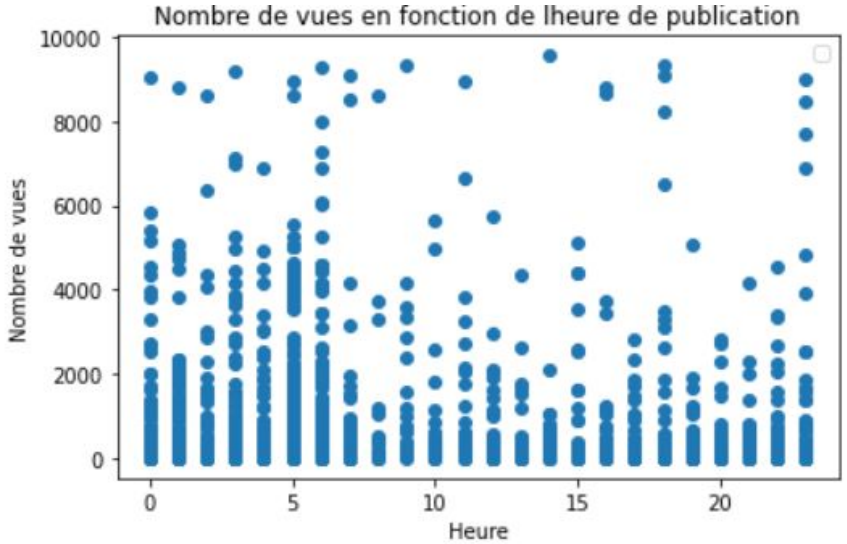
- Prédire le nombre de vues d'une vidéo

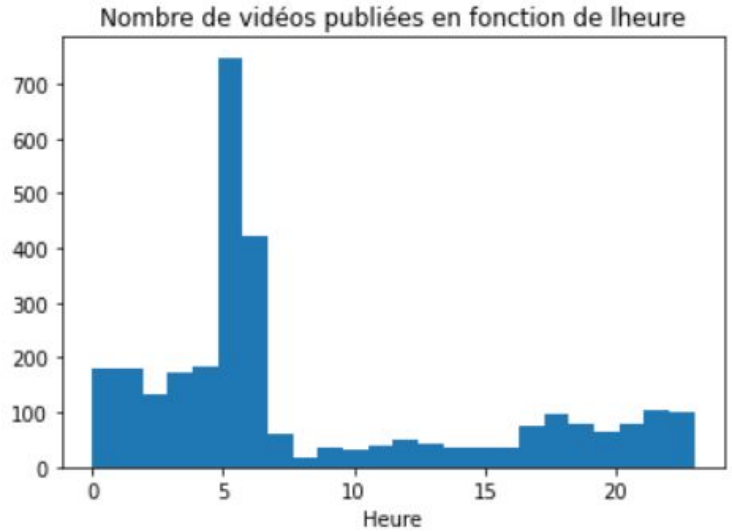
- 4 jeux de données
  - meta_data
  - image_df
  - title_df
  - desc_df

Quelques visualisations des données

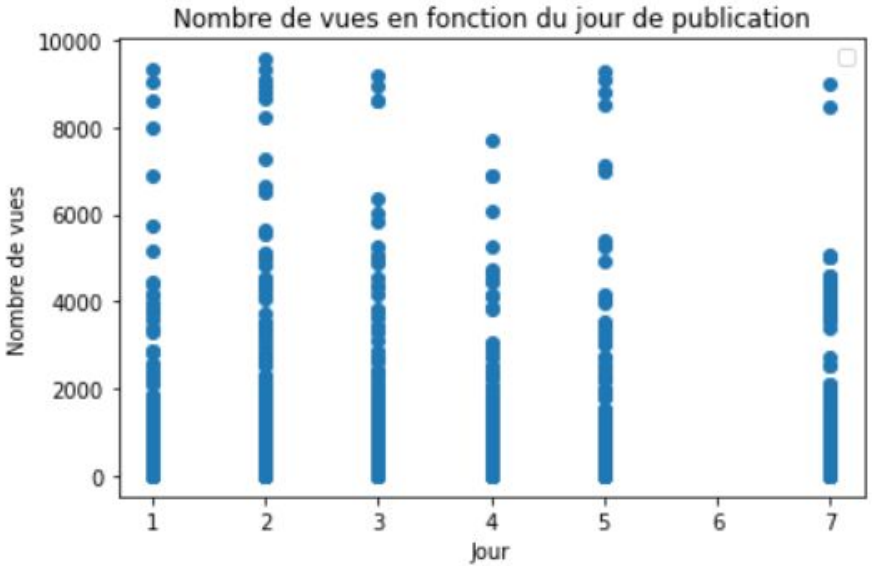


Quelques visualisations des données

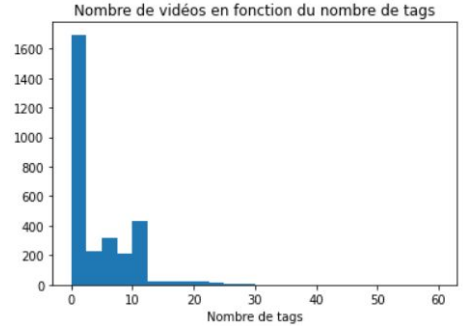


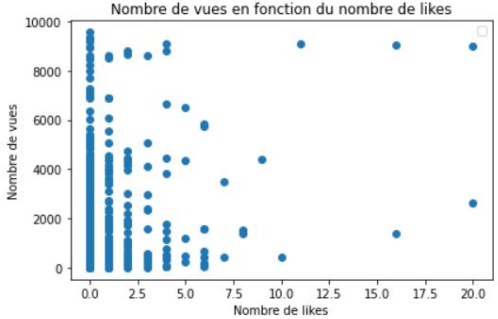


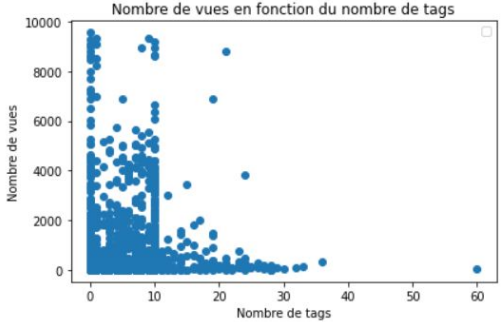
Quelques visualisations des données



Quelques visualisations des données







Traitement des données

- One hot encoding

emb_False	emb_True	prt_False	prt_True	prt_a_False	prt_a_True	Ing_1	Ing_2	Ing_3	Ing_4	Ing_5	Ing_6	Ing_7	Ing_8	Ing_9	Ing_10
0	1	0	1	1	0	0	0	1	0	0	0	0	0	0	0
0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0
0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0
0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0
0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0

Traitement des données

- Cyclical feature encoding

sin_hour	cos_hour	cos_day	sin_day
1.0	6.123234e-17	-0.222521	-0.974928
1.0	6.123234e-17	-0.222521	-0.974928
1.0	6.123234e-17	-0.222521	-0.974928
1.0	6.123234e-17	-0.222521	-0.974928
1.0	6.123234e-17	-0.222521	-0.974928

$$x_{sin} = \sin\left(\frac{2 * \pi * x}{\max(x)}\right)$$

$$x_{cos} = \cos\left(\frac{2 * \pi * x}{\max(x)}\right)$$

Traitement des données

- Feature selection avec Lasso regularization

Entrée [88]:
```
y = metadata['views']
x = image.loc[:, image.columns != 'comp_id']
```

Entrée [66]:
```
model = SelectFromModel(LogisticRegression(C=1, penalty='l1', solver='liblinear'))

model.fit(x, y)
model_index = model.get_support()

count = collections.Counter(model.get_support())
count
```

Out[66]:
```
Counter({False: 2742, True: 1258})
```

Modèles classiques

- Régression Linéaire Multiple

Entrée [73]:
```
std_scale = preprocessing.StandardScaler().fit(X_concat_train)
X_train_std = std_scale.transform(X_concat_train)
X_test_std = std_scale.transform(X_concat_test)
```

Entrée [79]:
```
# Entraînement
linearReg = LinearRegression().fit(X_train_std,y_concat_train)

# Test et évaluation
y_pred_linearReg = linearReg.predict(X_test_std)
print(evaluation(y_concat_test,y_pred_linearReg))
```

```
0.8150072853099513
```

Modèles classiques

- SVR

SVR

Entrée [77]:

```
for k in ['linear','poly','rbf','sigmoid']:
    svr = svm.SVR(kernel=k)
    svr.fit(X_train_std, y_concat_train)
    y_pred_svr = svr.predict(X_test_std)
    print(evaluation(y_concat_test,y_pred_svr))
```

```
0.8682720826389111
0.8610071312343667
0.8609954247080754
0.8610273368360938
```

Modèles classiques

- Arbre de décision

Entrée [80]:
```
# Entraînement
model_tree = tree.DecisionTreeRegressor()
model_tree.fit(X_concat_train, y_concat_train)

# Test et évaluation
y_pred_tree = model_tree.predict(X_concat_test)
print(evaluation(y_concat_test,y_pred_tree))
```

```
0.8446131273380897
```

Modèles classiques

- Forêt aléatoire

Forêt aléatoire

Entrée [81]:
```
# Entraînement
model_rf = RandomForestRegressor()
model_rf.fit(X_concat_train, y_concat_train)

# Test et évaluation
y_pred_rf= model_rf.predict(X_concat_test)
print(evaluation(y_concat_test,y_pred_rf))
```

```
0.8776789435673219
```

Autres modèles

- Lasso
- Ridge
- XGBoost
- LightGBM
- CatBoost

Lasso

```
# Entraînement
model_concat_lasso = Lasso(alpha = 100, max_iter = 30, tol = 0.1)
model_concat_lasso.fit(X_concat_train, y_concat_train)

# Test et évaluation
y_pred = model_concat_lasso.predict(X_concat_test)
print(evaluation(y_concat_test,y_pred))
```

```
0.8766135170259098
```

RIDGE

```
# Entraînement
model_concat_ridge = Ridge(alpha = 50, max_iter = 100, tol = 0.1)
model_concat_ridge.fit(X_concat_train, y_concat_train)

# Test et évaluation
y_pred = model_concat_ridge.predict(X_concat_test)
print(evaluation(y_concat_test,y_pred))
```

```
0.8353283037588435
```

XGBoost

```
# Entraînement
xgb_model = XGBRegressor()
xgb_model.fit(X_concat_train, y_concat_train)

# Test et évaluation
y_pred_xgb = xgb_model.predict(X_concat_test)
print(evaluation(y_concat_test,y_pred_xgb))
```

```
[19:04:31] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
0.8768094101211513
```

LightGBM

```
# Entraînement
model_lgbm = LGBMRegressor()
model_lgbm.fit(X_concat_train, y_concat_train)

# Test et évaluation
y_pred = model_lgbm.predict(X_concat_test)
print(evaluation(y_concat_test,y_pred))
```

```
0.8785824302244919
```

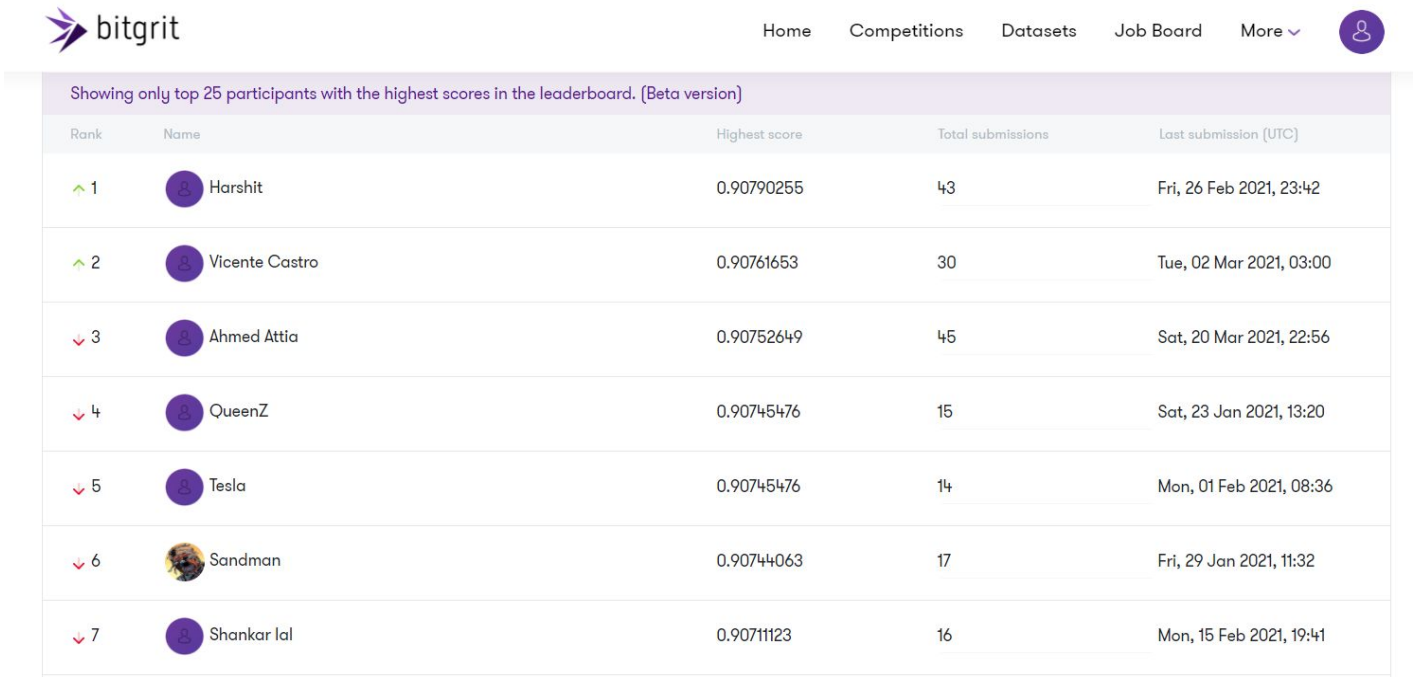
CatBoost

```
# Entraînement
model_catboost = CatBoostRegressor()
model_catboost.fit(X_concat_train, y_concat_train)
```

```
# Test et évaluation
y_pred_catboost = model_catboost.predict(X_concat_test)
print(evaluation(y_concat_test,y_pred_catboost))
```

```
0.8806404350492417
```

Résultats du concours



bitgrit

Home Competitions Datasets Job Board More

Showing only top 25 participants with the highest scores in the leaderboard. (Beta version)

Rank	Name	Highest score	Total submissions	Last submission (UTC)
^ 1	Harshit	0.90790255	43	Fri, 26 Feb 2021, 23:42
^ 2	Vicente Castro	0.90761653	30	Tue, 02 Mar 2021, 03:00
v 3	Ahmed Attia	0.90752649	45	Sat, 20 Mar 2021, 22:56
v 4	QueenZ	0.90745476	15	Sat, 23 Jan 2021, 13:20
v 5	Tesla	0.90745476	14	Mon, 01 Feb 2021, 08:36
v 6	Sandman	0.90744063	17	Fri, 29 Jan 2021, 11:32
v 7	Shankar lal	0.90711123	16	Mon, 15 Feb 2021, 19:41

	solution_xgboost.csv 22.3 KB	Tue, 09 Mar 2021, 17:04	solution with xgboost	0.89141519
	solution_xgboost2.csv 21.59 KB	Tue, 09 Mar 2021, 17:41	xgboost with all the columns and without negative values	0.89764708
	solution_xgboost3.csv 21.55 KB	Wed, 10 Mar 2021, 10:22	xgboost with min = 20 (and not neg values or 0)	0.89767413
	solution_randomforest.csv 22.17 KB	Wed, 10 Mar 2021, 11:20	randomforest, all columns, min=20	0.89658541
	solution_lasso.csv 22.26 KB	Wed, 10 Mar 2021, 13:09	simple lasso without optimisation of the parameters, all columns, min=20	0.89791767
	solution_lasso4.csv 22.3 KB	Tue, 16 Mar 2021, 09:01	lasso, data scaled, opti, min=20	0.8989735
	solution_lgbm.csv 22.32 KB	Tue, 16 Mar 2021, 10:33	lgbm, sans opti, sans traitement, min=20	0.8961885

	solution_lasso4.csv 22.3 KB	Tue, 16 Mar 2021, 09:01	lasso, data scaled, opti, min=20	0.8989735
	solution_lgbm.csv 22.32 KB	Tue, 16 Mar 2021, 10:33	lgbm, sans opti, sans traitement, min=20	0.8961885
	solution_lgbm2.csv 22.29 KB	Tue, 16 Mar 2021, 10:41	with data scaling	0.89715968
	solution_mix.csv 22.37 KB	Tue, 16 Mar 2021, 13:12	mix xgboost3 and lasso4	0.89990392
	solution_mix2.csv 22.37 KB	Tue, 16 Mar 2021, 13:43		0.89952991
	solution_mix3.csv 22.34 KB	Wed, 17 Mar 2021, 10:43		0.89978791
	solution_mix4.csv 22.32 KB	Wed, 17 Mar 2021, 10:50		0.89955147

Conclusion

- Essayer d’autres types de pré propressing afin de trouver la manière optimale de traitement de ce type de données
- Explorer d’autres méthodes de réduction de dimension afin de garder les meilleures variables possibles
- Tester d’autres modèles de la même logique que CatBoost
- Explorer davantage le gridSearch