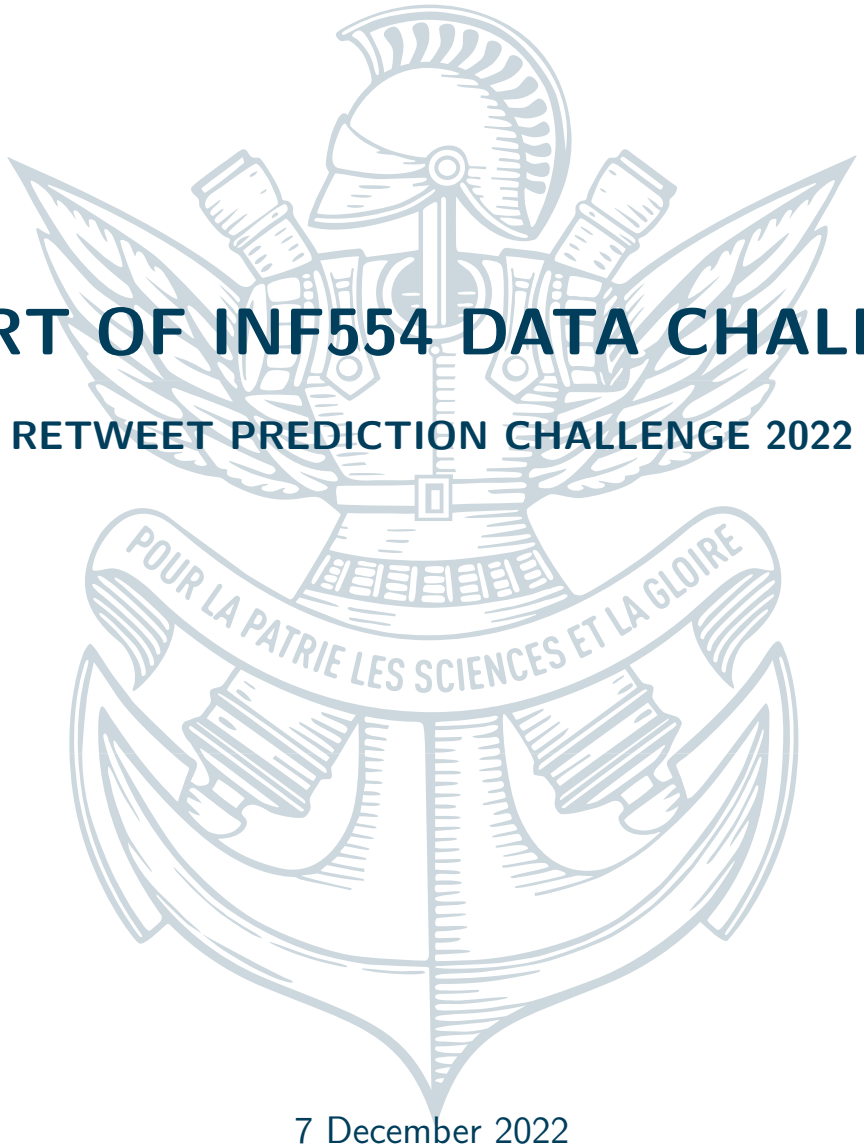

REPORT OF INF554 DATA CHALLENGE

RETWEET PREDICTION CHALLENGE 2022

7 December 2022

Kaggle Team : **Retweet**
By : Laychiva CHHOUT : laychiva.chhout@polytechnique.edu
Vannvatthana NORNG : vannvatthana.norng@polytechnique.edu

1 FEATURE SELECTION/EXTRACTION

Given the dataset "train.csv", in our prediction, we select some of the features that are already given in the original data, while also add some new features in order to increase the performance of the prediction.

1.1 FEATURES SELECTED FROM THE ORIGINAL DATA

There are 12 features that are given in the training dataset, and we select 6 among them by dropping retweets_count (which is the feature we need to predict) and TweetID which is an irrelevant feature for prediction. Those 6 parameters are : favorites_count, followers_count, statuses_count, friends_count, timestamps, verified.

1.2 NEW FEATURES ADDED

1.2.1 • DEAL WITH FEATURE **text**

Given in the dataset "train.csv", the feature "text" play an important role in the prediction. We add another feature which is **len_text** which obtain by extracting the lenght from each text. We add this features because in the data, the text with longer length has more number of retweets (Cf. Figure 1).

What we have tried but decided not to use : For this feature, we also tried to use sentiment analysis to extract the emotion from the text, since texts from the given dataset are not all in the same language (mostly in French), firstly we do the translation of every language to English in order to use "sentiment analysis from TextBlob" [1], we also do the pre-processing on that text by dropping stopword, most frequency word and the less frequency word. But this process doesn't improve the performance of the model, so we decided not to use it.

1.2.2 • DEAL WITH FEATURE **timestamps**

By intuition, we think that exact time and day that the tweet has been posted will affect the performance of the model, people surfs twitter more at night, on weekend, etc. We explore the given timestamp by adding 3 new features from this timestamps, **weekend**, **cos** and **sin**.

— **weekday** : The given timestamp is given in millisecond, so what we do is to convert it to "datetime" type, and extract the day from that datetime, which we assigned **monday = 1, ..., sunday = 7**.

— **cos, sin** : This new feature are created in goal of assigned similar or the same value of **cos,sin** to each tweet which is posted in 24h.

$$\cos(TweetID) = \cos(\text{total seconds start from 00 :00} \times 2\pi/\text{total second in a single day})$$

$$\sin(TweetID) = \sin(\text{total seconds start from 00 :00} \times 2\pi/\text{total second in a single day})$$

1. Cf. https ://textblob.readthedocs.io/en/dev/quickstart.html

Finally, we have 10 features for training our model which are : favorites_count, followers_count, statuses_count, friends_count, timestamps, verified, weekend, cos, sin and len_text.

After adding new features, by calculating correlation matrix (as shown in Figure 2), we observe that the correlation between features are small, it means that all features are independent.

We can also know the importance of each features after training the model.

2 MODEL CHOICES, TUNING AND COMPARISONS

2.1 Model Choices

We have used Linear Regression, Extreme Gradient Descent, Random Forest Regression, Decision Tree Regression and Gradient Boosting Regression to do the prediction. We considered XGB Regression because we have a large observation in dataset, the number of features is significantly smaller than the number of observations, and the data has mixture of numerical features and categorical features (for example, boolean value for "verified" feature). For RandomForest, we chose it due to the same reason of large dataset, and the interpretation of the output does not really matter. In addition, it can avoid the overfitting problem. Generally, XGB and RandomForest will give the best accuracy because they are usable with non-linear data and they have great adaptability according to the given data. Especially, XGB can also ignore outlier and missing data.

We also have tried using SVM and Neural Network Keras but both models took too much times to be executed, so we decided to role out these two models.

2.1.1 • Comparison of model before tuning

Performance of models (cf Figure 4) :

	lin_reg	random_forest	decision_tree	XGB	GradientBoosting
mae	9.603	5.490	7.178	5.593	5.95

Conclusion before tuning : We observe that random forest and XGB give us the smallest error among those 5. So, we decided to choose **XGBRegressor()** and **RandomForestRegressor()** for tuning hyperparameter.

2.2 Tuning Hyperparameters

2.2.1 • More details about **XGBRegressor()** and **RandomForestRegressor()**

1. **XGBRegressor()** : In this model, the most important parameters are n_estimators, max_depth, learning_rate, gamma and min_child_weight [2]. Tuning these parameters can improve accuracy and avoid overfitting. But, Machine Learning is a field of trade-off, so the accuracy will decrease in order to avoid overfitting. For example, if max_depth is too big, the model is more complex so there will be overfitting.

2. cf. https ://xgboost.readthedocs.io/en/stable/tutorials/param_tuning.html

2. **RandomForestRegressor()** : In this model, the most important parameters are : n_estimators, max_depth, min_samples_split, min_samples_leaf, and bootstrap, same as we mentioned above, tuning these parameters can improve accuracy and avoid overfitting. But, Machine Learning is a field of trade-off, so decrease the accuracy will decrease in order to avoid overfitting.

2.2.2 • Tuning process

In order to do hyperparameters tuning, both models can be tuning by using RandomizedSearchCV and GridSearchCV. In this data challenge, we use RandomizedSearchCV, it takes less time to execute because it randomly chooses a certain number of combinations of parameters unlike GridSearchCV which fits all number of combinations (too costly). In this project, we use only use "cv=3" which means we use 3-Fold splitting strategy.

1. **XGBRegressor()** : Thanks to the best performance of this model, tuning hyperparameters of XGBRegressor took less time to be executed. Here we define parameters distribution as in the (Figure 5) :

   In order to get the best paramters,we used technique of parameter narrowing. First, we tried with n_estimators : [100,500,1500] and decrease it sequentailly, until [100,500,700] and we repeated the same technique for the other hyperparameters, (we did one hyperparamter at a time). We get the following result (Figure 6)

   Remark :
   - Gamma is a regulation parameters, so increasing this parameter could help avoiding overfitting, but the accuracy will decrease.
   - Decreasing max_depth and learning_rate can reduce overfitting because it reduces the complexity of the model.
2. **RandomForestRegressor()** : is heavy in term of execution time, so tuning this model is too costly(it took more than 3h to fit 150 model with X_train). Here we used parameters distribution as in the (Figure 7), here we have also used the technique of parameters narrowing to find the best parameter grid. But, due to complexity of model itself, it took lots of time to be tuned and we get the following result (Figure 8)

   Alternative way that we also used : Since the execution time of model is costly, we also tried to use 20% partition of the data to do tuning (Cf. Figure 9). By doing so, we can gain lots of time.

2.3 Comparisons

After tuning, we train the model with best_param_ from each tuning and we get the following results (cf. Figure 10 and Figure 11) :

	random_forest	XGB
MAE	5.551	5.457

Conclusion : We used XGBRegression for our final prediction, because :

1. It is more accurate.
2. Take less time to train, easy to tune the hyperparameters.

A APPENDIX

```
# the dependence between number of retweets and lenght of text
X_train['retweets_count'].groupby(pd.qcut(X_train['text'].apply(lambda x: len(x)),5)).mean()
```

```
text
(5.999, 32.0]      8.736020
(32.0, 51.0]      11.972802
(51.0, 73.0]      12.422868
(73.0, 109.0]     18.745351
(109.0, 279.0]    29.717083
Name: retweets_count, dtype: float64
```

FIGURE 1 – Retweet and Length of text

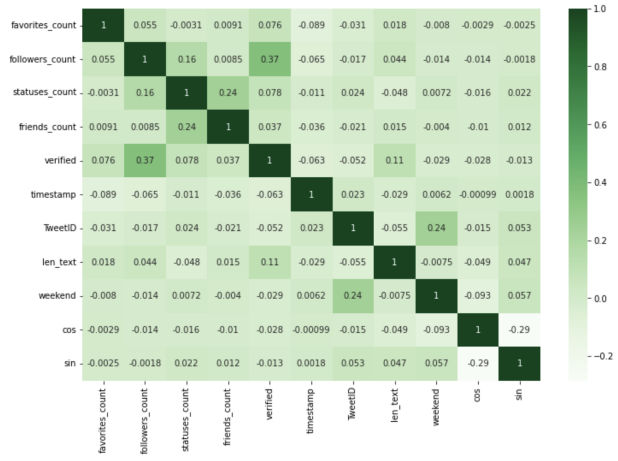

FIGURE 2 – Correlation matrix

```
import xgboost as xgb
```

```
xgb.plot_importance(reg1);
plt.tight_layout()
```

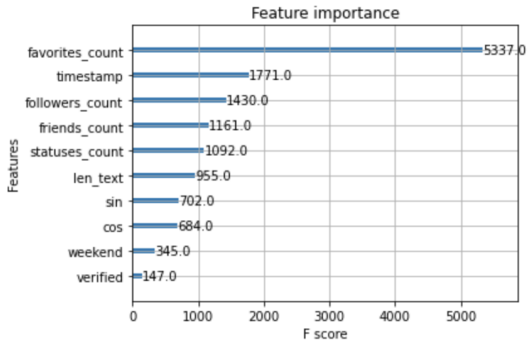

FIGURE 3 – Features importance

```
In [48]: #Error of each model, here we use mean absolute error
models= [('lin_reg', lin_reg), ('random forest', forest), ('decision tree', tree),('XGB', xgb),('GradientBoosting', g
# models= [('random forest', forest)]
for i, model in models:
    predictions = model.predict(X_test.drop(['TweetID'],axis=1))
    MAE = mean_absolute_error(y_test, predictions)
    print('Prediction error of ', i , 'is ', MAE)
```

```
Prediction error of  lin_reg is  9.603221934042974
Prediction error of  random forest is  5.490115028580577
Prediction error of  decision tree is  7.718747351470464
Prediction error of  XGB is  5.593042710281965
Prediction error of  GradientBoosting is  5.952153249541317
```

FIGURE 4 – Comparison of those 5 models

```
param_distributions={'gamma': [0, 0.5, 1],
                     'learning_rate': [0.05, 0.1, 0.15, 0.2],
                     'max_depth': [5, 7, 10],
                     'min_child_weight': [1, 2, 3, 4],
                     'n_estimators': [100, 500, 700]},
random_state=42, return_train_score=True,
scoring='neg_mean_absolute_error', verbose=5)
```

FIGURE 5 – Parameters grid

```
random_cv.best_params_
```

```
{'n_estimators': 500,
 'min_child_weight': 2,
 'max_depth': 7,
 'learning_rate': 0.05,
 'gamma': 1}
```

FIGURE 6 – Best parameters

```
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(), n_iter=50,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [5, 8, 12, 16, 20, None],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 3, 4],
                                        'n_estimators': [100, 200, 300, 400,
                                                         500]},
                   random_state=42, verbose=2)
```

FIGURE 7 – Parameters grid

```
#best parameters of random forest
rf_random.best_params_
```

```
{'n_estimators': 300,
 'min_samples_split': 2,
 'min_samples_leaf': 4,
 'max_depth': 8,
 'bootstrap': True}
```

FIGURE 8 – Best parameters

```
# Sample training data for tuning models (use full training data for final run)
tuning_sample = 50000
idx = np.random.choice(len(X_train), size=tuning_sample)
X_tuning = X_train.iloc[idx]
y_tuning = y_train.iloc[idx]
```

FIGURE 9 – Snip code of partition strategy

```
# Prediction use the best paramters
reg1 = random_cv.best_estimator_
# reg1 = XGBRegressor(gamma=1, learning_rate=0.05,
#                     max_cat_threshold=64, max_depth=7, max_leaves=0, min_child_weight=2,
#                     n_estimators=500)
reg1.fit(X_train.drop(['TweetID'],axis=1), y_train)
y_pred = reg1.predict(X_test.drop(['TweetID'],axis=1))
print("Prediction error:", mean_absolute_error(y_true=y_test, y_pred=y_pred))
```

```
Prediction error: 5.4572512355641
```

FIGURE 10 – Result of XGBRegression

```
# rf = rf_random.best_params_
rf = rf_random.best_estimator_
# We fit our model using the training data
rf.fit(X_train.drop(['TweetID'],axis=1), y_train)
# And then we predict the values for our testing set
yrf_pred = rf.predict(X_test.drop(['TweetID'],axis=1))
# We want to make sure that all predictions are non-negative integers
yrf_pred = [int(value) if value >= 0 else 0 for value in yrf_pred]

print("Prediction error:", mean_absolute_error(y_true=y_test, y_pred=yrf_pred))
```

```
Prediction error: 5.551077774952679
```

FIGURE 11 – Result of RandomForestRegression