Text Normalization Challenge - English Language

Pavithra POORNACHANDRAN
Dahou Abdelhalim Hafedh
Tahani FENNIR

UNIVERSITÉ DE LORRAINE

IDMC Institut des sciences du Digital Management & Cognition
INSTITUT UNIVERSITAIRE DU NUMÉRIQUE

October 27, 2020

Table of Content

Introduction & data

XGBoosting

Neural Network

Milestone

Project & Data source

- Subject of Kaggle compition in 2017 .
- **The aim of this project** is to “automate the process of developing text normalization grammars via machine learning”.

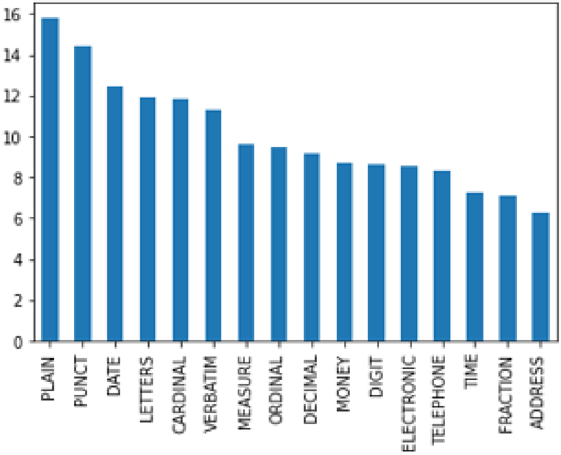
Data description

- The data consist of 5 files: en_train, en_test, en_test_2, en_sample_submission, and en_sample_submission_2
- **en_train.csv** dataset contains 5 features: **Sentence_id, Token_id, Class :16 classes, Before and after** and 9,918,441 observation.
- **en_test.csv** contain 3 features: **Sentence_id, Token_id, and Before** and 1088563 observation.
- **en_sample_submission.csv** contains 2 features: **id and after**
- **en_sample_submission_2.csv** contains 2 features: **id and after**

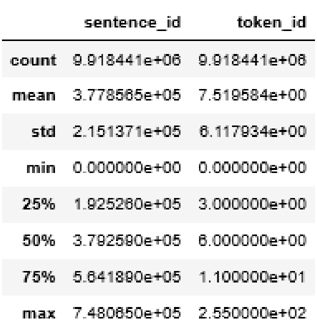
Training Dataset

- The maximum size of training data is : 9,918,441
- the maximum size of id_sentences is :748,066
- The maximum size of id_tokens is :256
- The data contain 16 classes: 'PLAIN', 'PUNCT', 'DATE', 'LETTERS', 'CARDINAL', 'VERBATIM','DECIMAL', 'MEASURE', 'MONEY', 'ORDINAL', 'TIME', 'ELECTRONIC', 'DIGIT', 'FRACTION', 'TELEPHONE', 'ADDRESS'.
  - The **"PLAIN"** class is by far the most frequent, followed by **"PUNCT" and "DATE".** With **"TIME"**.
  - **"FRACTION", and "ADDRESS"** having the lowest number of occurrence.

Training Dataset

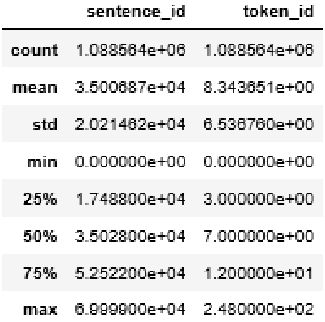


Training & Testing Data



	sentence_id	token_id
count	9.918441e+06	9.918441e+06
mean	3.778565e+05	7.519584e+00
std	2.151371e+05	6.117934e+00
min	0.000000e+00	0.000000e+00
25%	1.925260e+05	3.000000e+00
50%	3.792590e+05	6.000000e+00
75%	5.641890e+05	1.100000e+01
max	7.480650e+05	2.550000e+02

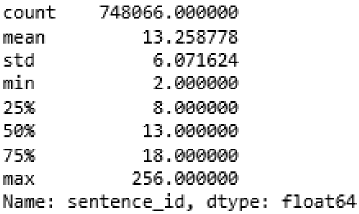
(a) Training Data



	sentence_id	token_id
count	1.088564e+06	1.088564e+06
mean	3.500687e+04	8.343651e+00
std	2.021462e+04	6.536760e+00
min	0.000000e+00	0.000000e+00
25%	1.748800e+04	3.000000e+00
50%	3.502800e+04	7.000000e+00
75%	5.252200e+04	1.200000e+01
max	6.999900e+04	2.480000e+02

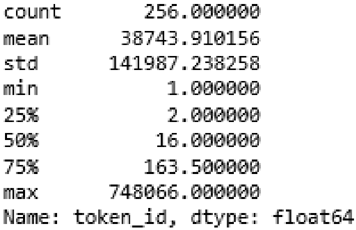
(b) Testing Data

Training Data



```
count    748066.000000
mean         13.258778
std           6.071624
min           2.000000
25%           8.000000
50%          13.000000
75%          18.000000
max         256.000000
Name: sentence_id, dtype: float64
```

(c) Training Data



```
count        256.000000
mean       38743.910156
std       141987.238258
min            1.000000
25%            2.000000
50%           16.000000
75%          163.500000
max       748066.000000
Name: token_id, dtype: float64
```

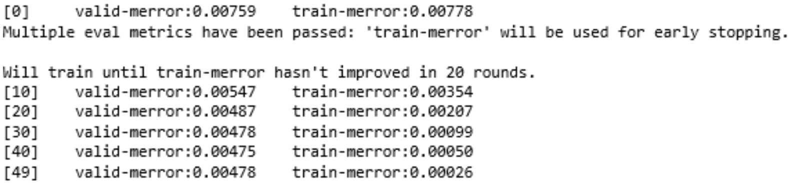
(d) Training Data

XGBoosting

XGBoosting

- XGBoost stands for eXtreme Gradient Boosting.
- XGBoost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data.
- **XGBoost** is an implementation of gradient boosted decision trees designed for speed and performance.
- We built XGboost model to use the context to label data.
- This model is trained with all training dataset.

XGBoost



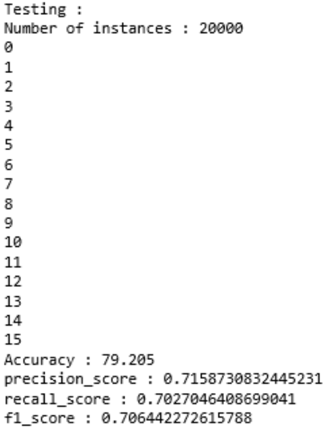
```
[0]     valid-merror:0.00759    train-merror:0.00778
Multiple eval metrics have been passed: 'train-merror' will be used for early stopping.

Will train until train-merror hasn't improved in 20 rounds.
[10]    valid-merror:0.00547    train-merror:0.00354
[20]    valid-merror:0.00487    train-merror:0.00207
[30]    valid-merror:0.00478    train-merror:0.00099
[40]    valid-merror:0.00475    train-merror:0.00050
[49]    valid-merror:0.00478    train-merror:0.00026
```

- We choose 320,000 samples instead of 960,000 to save time and lack powerful computers.
- The accuracy of our model is 99.52%

Neural Network

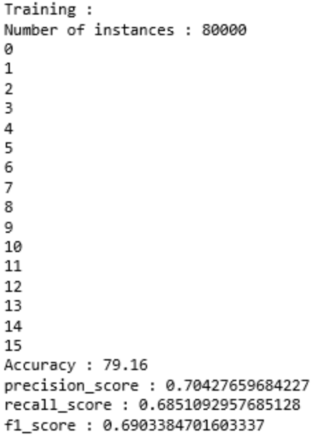
- We used recurrent neural networks to predict the outputs of the data.
- We used Keras to build our model. we used Sequential API,
- Sequential API: It is used to build models as a simple stack of layers. First, we instantiate our Sequential model object and then, you add layers to it one by one using the add() method.

Results



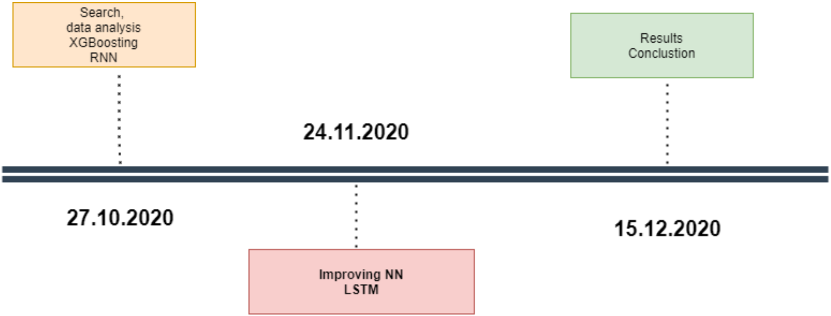
```
Testing :
Number of instances : 20000
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Accuracy : 79.205
precision_score : 0.7158730832445231
recall_score : 0.7027046408699041
f1_score : 0.706442272615788
```

(e)



```
Training :
Number of instances : 80000
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Accuracy : 79.16
precision_score : 0.70427659684227
recall_score : 0.6851092957685128
f1_score : 0.6903384701603337
```

(f)



Search,
data analysis
XGBoosting
RNN

27.10.2020

24.11.2020

Improving NN
LSTM

Results
Conclustion

15.12.2020

Thank you!