Text Normalization Challenge - English Language

Pavithra POORNACHANDRAN,
Abdelhalim hafedh Dahou,
Tahani FENNIR

Supervisors:
Miguel Couceiro
Ajinkya Kulkarni

26.01.2021

Table of contents

1. Introduction

    Description of the data

2. XGBoost

3. LSTM

4. Sequence-to-Sequence

5. Conclusion

Introduction

Introduction

Our project is **Kaggle competition challenge** to automate the process of developing text normalization grammars via machine learning.

Text normalization is the process of transforming a text into a canonical (standard) form[4].
Examples:

- Transform word numerals into numbers (eg.: 'twenty three'→'23')
- Acronym normalization (e.g.: 'US'→'United States'/'U.S.A')
- Removal or substitution of special characters(e.g.: remove hashtags).
- Removal of duplicate whitespaces and punctuation.

This normalized text is used in:

- **Text-to-Speech systems (TTS)** - used a textual data as a standard representation that can be converted into the audio form.
- **Automatic Speech Recognition (ASR)** - raw textual data is processed into language models using text normalization techniques.

Description of the data

- Data-set submitted by Richard Sproat and Navdeep Jaitly for the Kaggle competition.

https://www.kaggle.com/c/text-normalization-challenge-english-language

- Training dataset contains 9 million observations and testing dataset contains 1 million observations.

Data	No. of tokens
Train	9,918,442
Test	1,088,565

Training Data set

Training dataset contains five features such as:

- **Sentence_id** represent the id of the sentence.
- **Token_id** represent the id of the token.
- **Before** feature represent actual sentence.
- **After** feature represent predicted output.
- **Class** feature represent the category of each input.

	sentence_id	token_id	class	before	after
0	0	0	PLAIN	Brillantaisia	Brillantaisia
1	0	1	PLAIN	is	is
2	0	2	PLAIN	a	a
3	0	3	PLAIN	genus	genus
4	0	4	PLAIN	of	of

Figure 1: Training Data set

Training Data Set

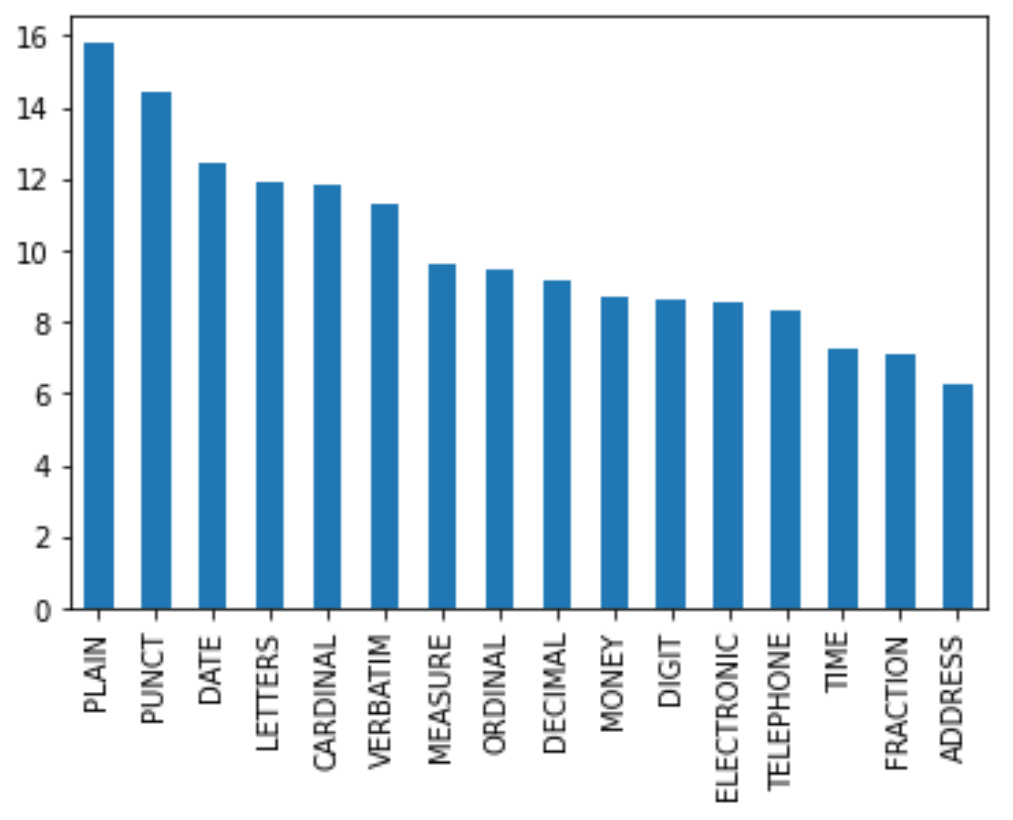


Figure 2: Semiotic class distribution of the training dataset

Test Data Set

Testing data set contains three features:

- sentence_id
- Token_id
- Before

	sentence_id	token_id	before
0	0	0	Another
1	0	1	religious
2	0	2	family
3	0	3	is
4	0	4	of

Figure 3: Testing Data set

Experiment

Here we used three different models with different approaches. They are,

- XGBoost
- LSTM
- RNN Sequence-to-Sequence model with Attention Mechanism

XGBoost

XGBoost

- XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework[1].
- It has better control against over-fitting by using more regularized model formalization.
- Internally, XGBoost has parameters for cross-validation, regularization,objective user-defined functions, missing values, compatible API with scikit-learn.

Implementation

- Totally we used 320000 tokens
- Represent every character in the token with it's ASCII value.
- Once the class of a token has been defined, we normalized it accordingly.
- The usage of a token in a sentence determines its semiotic class. And the surrounding tokens play a substantial role in deciding the class of the token.
- In specific, distinction between classes such as **DATE** and **CARDINAL**, for example, CARDINAL 2016 is normalized as two thousand and sixteen, while DATE 2016 is twenty sixteen, the surrounding context is very important

Implementation

- Pad the empty window with zeros
- Pass the parameter *`Multi:softmax`* (XGBoost performed multiclass classification by using softmax) and *`merror`* (used to find Multiclass classification error rate.
- Once we trained this classifier, we predict the classes for the test data and label each token with its semiotic class.

Results

```
[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
```

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(pred, y_valid)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 99.52%
```

LSTM

LSTM

Long short-term memory (LSTM) is an artificial Recurrent Neural Network (RNN) architecture used in the field of deep learning[2].

Model approaches and architecture

- The model maps a sequence of input characters to a sequence of output words.
- This LSTM model has two components an **encoder** and a **decoder**.
  - First reads the input using an encoder to generate a high-dimensional vector of ”thought”.
  - To produce the translated (normalized) text, a decoder processes the ”thought” vector.

LSTM

- The encoder is a three-layer RNN encoder that translates a high-level feature representation into the input character sequence.
- The decoder, is a two-layer RNN decoder that attends to the high-level features and spells out one token at a time for the normalized text.

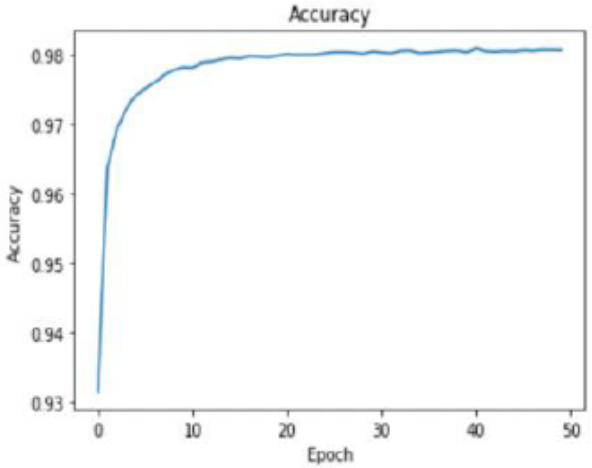
LSTM

Model architecture:

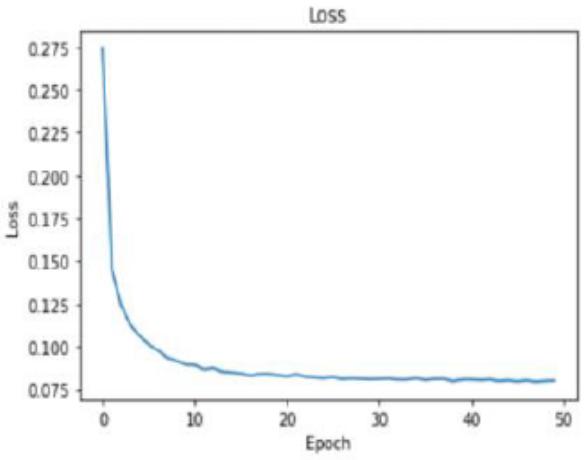
- 256 hidden units in each layer of the encoder and decoder.
- Three bidirectional LSTM layers in the encoder
- Two LSTM layers in the decoder

Results

Results:



((a)) Accuracy



((b)) Loss

Sequence-to-Sequence

Sequence-to-Sequence

- A sequence to sequence model aims to map a fixed-length input with a fixed-length output.
- For the text normalization in the words-level models, only the most frequent words are kept, creating a fixed-sized vocabulary, with OOV words mapped to a common UNK token.
- Consequently, the performance is affected by the limited vocabulary.
- To tackle this problem, we applied some approaches like
  - Copying source words
  - Rely the models fully trained on character-based information

- The most common sequence-to-sequence (seq2seq) models are encoder-decoder models which takes the input sequence and predict the normalized sequence based on the label[3].

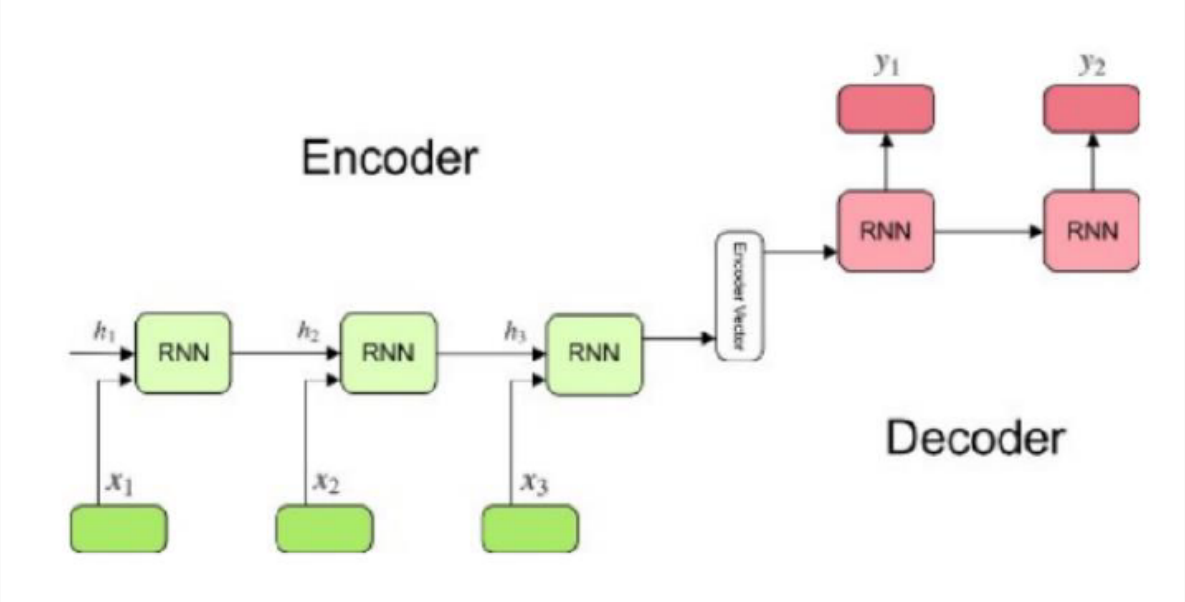


Figure 4: Encoder -Decoder Sequence-to-Sequence

Sequence-to-Sequence

Encoder

- Encoder reads the input sequence and summarizes the information which is called context vector.
- This context vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.
- This vector is then decoded by RNN which learns to output the target (output) sentence by generating one token at a time.

Decoder

- The decoder is an RNN whose initial states are initialized to the final states of the Encoder RNN.
- The context vector of the encoder's final cell is initial input to the decoder network.
- Using these initial states, the decoder starts generating the output sequence.

IMPLEMENTATION

We used attention mechanism which allows the decoder network to “focus” on a different part of the encoder’s outputs for every step of the decoder’s own outputs.

- Pytorch
- Relu - Hidden layers
- Softmax - Output layers

Loading data

- To read the data file we split the file into pairs and filter it by length and content.

```
Read 258348 sentence pairs
Counting chars...
Counted chars:
after 62
before 26
['2002', 'two thousand two']
```

Training data

- We run the input data through the encoder, and keep track of every output and the latest hidden state.
- Then the decoder is given the token as its first input, and the last hidden state of the encoder as its first hidden state.
- We add a teacher forcing concept for using the real target outputs as each next input, instead of using the decoder's guess as the next input.

- The Table contains the best performing hyperparameter settings of our proposed text normalization model.

Hyper-parameter	Value
Embedding dimension	256
Learning rate	0.001
Dropout	0.2
Number of Epochs	20

Figure 5: The best performing hyper-parameter

RESULTS

- Evaluation is mostly the same as training, but there are no targets so we simply feed the decoder's predictions back to itself for each step.
- Every time it predicts a token, we add it to the output string, and if it predicts the EOS token we stop there and finally compare it to the truth value to get the accuracy rate of the model on this unseen data.

Results

Class	F1 score
DATE	0.90
DECIMAL	0.89
LETTERS	0.86
CARDINAL	0.62
ADDRESS	0.53
FRACTION	0.67
MEASURE	0.84
VERBATIM	0.84

Figure 6: F1 score for each class in evaluation phase

Conclusion

Conclusion

- In this project we proposed three different models for the text normalization task. A good text normalization system must be generalized i.e., needs to be able to accurately predict the normalization input that has not been trained on before (unseen data).
- The XGBoost or Extreme Gradient Boosting method which has been proved to be a winning method with 99.52 %. It operates as a boosted random forest method and has shown comparatively good results.
- However, LSTM model perform well and tends to generate complete sense and canonical format for the unseen input.

Conclusion

- In the other hand, we have introduced a Neural Network model for text normalization that utilizes a character-level Sequence-to-Sequence model to handle problematic OOV cases.
- The Sequence-to-Sequence model gives us a good performance in some classes and low performance for the others due to the different amount of training data in each class.
- Finally, we conclude classifier has an important role in this model. But we more focus on the sequence-to-sequence model as we wanted to understand and implement it. Due to the lack of time and limited resources we did not spend much time to tune the hyper-parameters of the model and we list this as our future work.

REFERENCES

References i


T. Chen and C. Guestrin.
Xgboost: A scalable tree boosting system.
In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.


I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio.
***Deep learning*, volume 1.**
MIT press Cambridge, 2016.


I. Sutskever, O. Vinyals, and Q. V. Le.
Sequence to sequence learning with neural networks.
Advances in neural information processing systems, 27:3104–3112, 2014.

References II


M. Zare and S. Rohatgi.
Deepnorm-a deep learning approach to text normalization.
arXiv preprint arXiv:1712.06994, 2017.

Questions?