MSc Natural Language Processing - 2020/2021

Software Project

Text Normalization Challenge - English Language

Submitted By:

Pavithra POORNACHANDRAN

Abdelhalim Hafedh Dahou

Tahani FENNIR

Supervisors:

Miguel Couceiro

Ajinkya Kulkarni

15.01.2021

Abstract

It is Kaggle competition challenge to automate the process of developing text normalization grammars via machine learning focusing on English language. It is funded by Google's Text Normalization Research Group, which conducts research and develops tools for the identification, normalization and denormalization of non-standard terms, such as abbreviations, numbers or currency expressions, measuring phrases, addresses or dates, representing unique entities that are semantically limited.

However, one of the biggest challenges when developing a TTS or ASR system for a new language is to develop and test the grammar for all these rules. This project presents a challenge to the community given a large corpus of written text aligned to its normalized spoken form. We have applied three different algorithms with different approaches to predict normalized text, such as XGBoost, Sequence-to-Sequence, Long Term Short Memory (LSTM). Our approach takes very long time to train and to evaluate the model. We present different model experiments and results with various parameter settings. We have achieved excellent accuracy score in XBboost which got 99.52%.

Contents

List of Table

List of Figure

Chapter 1
Introduction

In the field of Text-to-Speech synthesis and the Automatic Recognition System, Deep Neural Networks have provided some of the most impressive results in recent years. With the application of deep learning, natural language processing has also seen major changes. There are, however, several unanswered problems that the neural models still have plenty of room for development to overcome. The creation of a flawless text normalization solution for machine translation and automatic speech recognition systems is one of the basic yet interesting challenges.

In speech technology, text normalization has a rich history [1], going back to the earliest work on complete TTS synthesis (Allen et al., 1987). In terms of weighted finite-state transducers, Sproat (1996) presented a unifying model for most text normalization issues (WFSTs). (Sproat et al., 2001) was the first study to treat the problem of text normalization as simply a problem of language modeling. More recent machine learning work specifically addressed to TTS text normalization involves (Sproat, 2010; Roark and Sproat, 2014; Sproat and Hall, 2014).[2]

For making 'silly' errors such as transforming measurements from 100 KG to 100 Kilobytes or transforming dates from 1/10/2017 to first January twenty-seventeen, a model would be highly penalized. We experimented with various models with distinct results to address this problem. We begin by recognizing the challenges of normalizing text and trying to understand the reason behind problems. We decided to make a different model for tackling this issue after a brief overview of the previous work done on this topic. We proposed a deep learning-based approach to text normalization using Recurrent Neural Network (RNN) based Encoder-Decoder architecture and sequence-to-sequence model with attention mechanism. Also, we trained our model with the XGBoost algorithm.

Text normalization is a ubiquitous pre-processing stage for a range of speech and language processing applications. The example is text normalization, specifically in the sense of a system that converts from a written representation of a text into a representation of how that text is to be read aloud. we use the term semiotic class to denote things like numbers, times, dates, monetary amounts, etc. Text normalization refers to the process of verbalizing semiotic class instances (e.g., converting something like 3 kg into its verbalization three kilograms).

1.1 Dataset

1.1.1 Description of the data

In this project we used the dataset submitted by the authors [3] for the Kaggle competition https://www.kaggle.com/c/text-normalization-challenge-english-language.

Our training dataset contains a total of 9 million observations and our testing dataset contains 10 million observations.

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

Table 1: Number of observations

The dataset is extracted from Wikipedia that could be decoded as UTF8. The text is then split into sentences and divided into sentences and through the Google TTS system's Kestrel text normalization system to produce the normalized version of that text. Kestrel's verbalizations are produced by first tokenizing the input and classifying the tokens, and then verbalizing each token according to its semiotic class.

1.1.2 Exploratory Data Analysis

Training data:

Training dataset contains 9,918,441 observation and totally five features such as sentence_id, token_id, class, Before and after. Each token is labelled with its respective class of semiotics.

- Sentence_id represent the id of the sentence
- Token_id represent the id of the token
- Before feature represent actual sentence, which is input to the model
- After feature represent predicted output which is normalized one.

	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

Table 2: Training Dataset

In total there are 16 classes. The PLAIN class is by far the most frequent, followed by PUNCT and DATE, TIME.

FRACTION, and ADDRESS classes having the lowest number of occurrences.

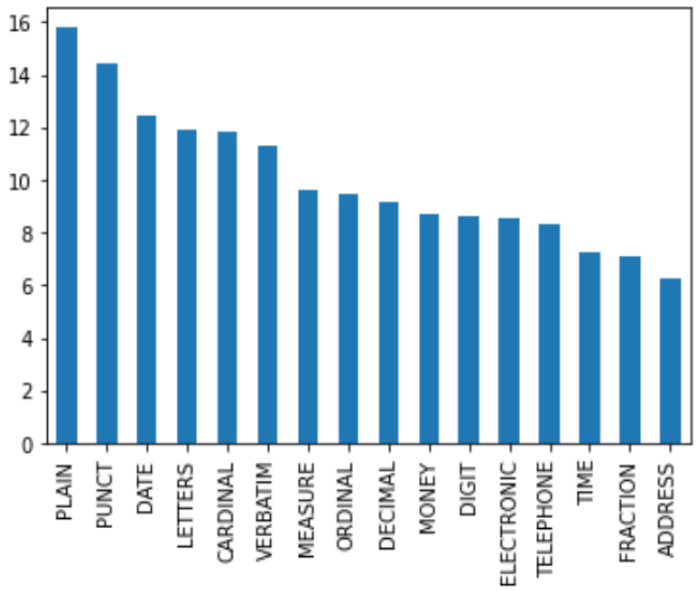


Figure 1: Semiotic class distribution of the training dataset

Testing data:

Testing dataset contains three features such as sentence_id, Token_id, Before. The aim of the project is to apply different models to the training dataset and validation on the testing dataset to predict the "after" column which is a normalized word.

	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

Table 3: Testing Dataset

Chapter 2

Experiments and Results

Our approach involves modelling the problem as classification. We have used 3 distinct models for our project with different activation functions and optimization techniques. The classifier determines the tokens that need to be normalized and a Sequence-to-Sequence model that normalizes the nonstandard tokens. We trained the classifier and sequence to sequence model individually and at last we compared the results. The following content will explain the models and implementation of the models. The three distinct models are as follows,

- XGBoost
- Sequence-to-Sequence model
- LSTM

2.1 XGBoost

XGBoost is a decision-tree-based ensemble Machine Learning algorithm and uses a gradient boosting (GBM) framework at its core. It is an optimized distributed gradient boosting library. XGBoost is well known for having better solutions than other algorithms for machine learning. The following diagram represents the evaluation of tree-based algorithms over the year [4].

Internally, XGBoost has parameters for cross-validation, regularization, objective user-defined functions, missing values, tree parameters, compatible API with scikit-learn. Boosting is a sequential technique which works on the principle of an ensemble. It combines a set of weak learners and delivers improved prediction accuracy. At any instant t, the model outcomes are weighed based on the outcomes of previous instant t-1. The outcomes predicted correctly are given a lower weight and the ones miss-classified are weighted higher.

2.1.1 Implementation

The key component of detecting the semiotic class of the token is about this task. Once the class of a token has been defined, we can normalize it accordingly properly. The usage of a token in a sentence determines its semiotic class. 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.

We choose a window size k and we represent every character in the token with it's ASCII value. We pad the empty window with zeros. We use the preceding k characters of the tokens and the later k characters of the tokens around the token in focus which helps the classifier understand in which context the token has been used. We use gradient boosting algorithms without any parameter tuning. It takes a lot of time to train with 9 million tokens, for the lack of computer and time we choose 3 million tokens from the dataset. And we used 10% of the train data as our validation set. Here we have seen an interesting thing, the window size is decreased and the classifier's accuracy also increases. It is reasonable, because most of the tokens are less than 10 length.

Also, the starting characters and the surrounding context of the long tokens are enough to determine their semiotic class. Also, the starting character and the surrounding context is enough to determine the semiotic class. Once we have trained this classifier, we predict the classes for the test data and label each token with its semiotic class.

2.1.2 Results

The maximum test accuracy we touched was 99.52%.

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

```
Accuracy: 99.52%
```

Figure 2: Accuracy of XGBoost

2.2 Sequence to Sequence model

2.2.1 Definition

For the problem of text normalization, the system input will be an unnormalized text represented as an input sequence of words or characters x = [X1, . . . , XT] with length **T** , we consider generating another output sequence y = [Y1 , . . . , YL] with length **L** that has the same meaning as x . The task is defined as a **sequence-to-sequence** learning problem which aims to learn the mapping from one sequence to another where the length of the input and output may differ. The most common sequence-to-sequence (seq2seq) models are encoder-decoder models [3] (Cho et al. 2014; Sutskever, Vinyals, and Le 2014), which commonly use a recurrent neural network (RNN) to encode the source (input) sentence into a single vector (context vector).

The context vector is an abstract representation of the entire input sentence. This vector is then decoded by a second RNN which learns to output the target (output) sentence by generating it one token at a time.
As the below figure illustrates, the model consists of parts: encoder phase and decoder phase.

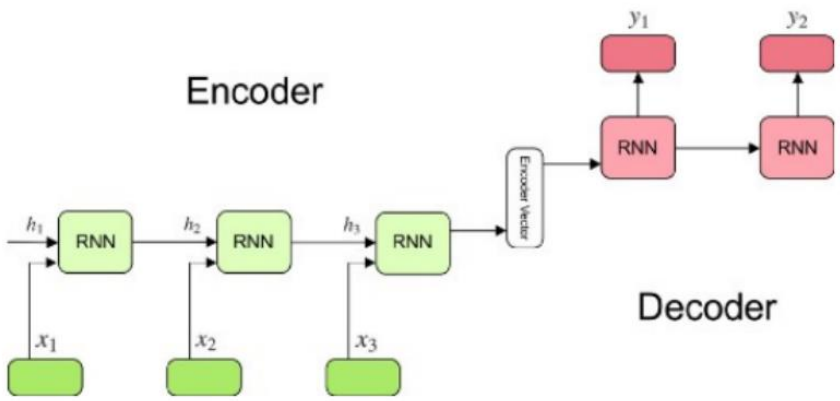


Figure 3: Encoder -Decoder Sequence-to-Sequence

Encoder

A stack of several recurrent units (LSTM or GRU cells for better performance) where each accepts a single element of the input sequence, collects information for that element and propagates it forward. Resume that to reads the input sequence x and transforms it to a corresponding context-specific sequence of hidden states h = [h_1 , . . . , h_T] using the following formula,

$$h_t = f(W^{(hh)} h_{t-1} + W^{(hx)} x_t)$$

By applying the appropriate weights to the previous hidden state h(t-1) and the input vector x_t. The encoder vector is the final hidden state produced from the encoder which encapsulates the information for all input elements in order to help the decoder make accurate predictions.

Decoder

A stack of several recurrent units where each predicts an output y_t at a time step t. the Decoder class does a single step of decoding, i.e., it outputs single token per time-step.

The layer will receive a hidden and cell state from the previous time-step, (St-1, Ct-1), and feed it through the (LSTM or GRU) with the current embedded token yt, to produce a new hidden state (St, Ct). We then pass the hidden state from the top layer of the RNN, through a linear layer, to make a prediction of what the next token in the target (output) sequence should be yt+1.

Recently, several extensions of the Seq2Seq models have been proposed with mechanisms such as attention [4] (Bahdanau, Cho, and Bengio 2014), copying [5] (See, Liu, and Manning 2017) and coverage [6] (Tu et al. 2016) Which contribute on the performance of the Seq2Seq models in different natural language tasks.

2.2.2 Implementation

As we knew, the models that work on natural language tasks can treat the sequence of text as words or characters.

For the text normalization and especially in the words-level models, in most cases 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 can apply solutions like

- Copying source words
- Rely on models fully trained on character-based information

Character models overcome the bottleneck of restricted vocabularies and do not require any pre-processing or tokenization. So, we decided to treat the input sequence as character-level.

Our experimental setup for assessing the performance of the text normalization model described above has an 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. We want to test if the character level models are appropriate for normalizing OOV tokens, which Seq2Seq model is the most effective and finally whether the Seq2Seq resolve the task with good accuracy.

- Loading data

  Represent each character in the data as a one-hot vector, or giant vector of zeros except for a single one (at the index of the characters) and set a unique index per character to use as the inputs and targets of the networks. 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 the model

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

Table 4: The best performing hyper-parameter

2.2.3 Evaluation

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 word, 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. We also store the decoder's attention outputs for display later.

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

Table 5: F1 score for each class in evaluation phase

2.3 Long Short-Term Memory (LSTM)

With high-level human language processing, LSTMs share strong similarities. Within the form of semantic and syntactic contexts, humans process and understand sentences. Based on their comprehension of previous terms, comprehend each word; they don't throw anything away and start thinking again from scratch. There is persistence in thinking.

A special variant of RNN that is capable of learning long-term dependencies is the Long Short Term Memory Network (LSTMs). All recurrent neural networks have the shape of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very basic structure, such as a single tanh layer. LSTMs also have this chain structure, but the repeating module has a different structure. Instead of having a single neural network layer, the repeating module in an LSTM contains four interacting.

One problem with applying LSTMs to text normalization is that in text normalization, the collection of significant cases is typically very sparse. Most tokens map to themselves, and although having that right is definitely critical, it's a relatively trivial case. The important cases that need special treatment are the numbers, times, dates, measurement expressions, currency quantities, etc., which matter most in text normalization systems.

2.3.1 Model approaches and architecture

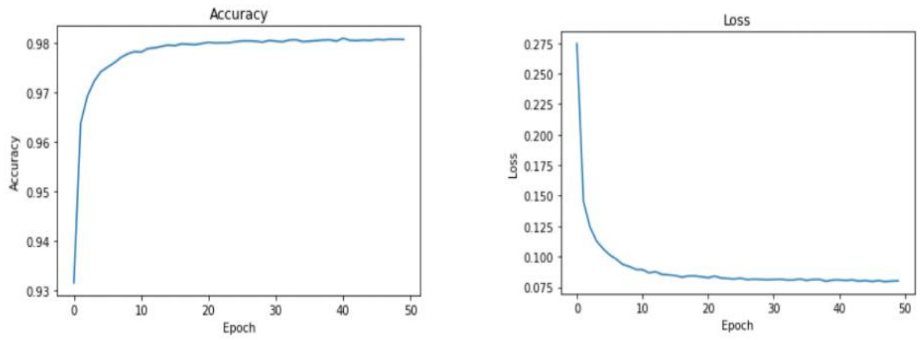
We model the whole text normalization task as one where the model maps a sequence of input characters to a sequence of output words [9]. For the input, the string must be in terms of characters. The normalized text output, which is dependent on its surrounding sentence context, is most appropriately represented as words.

This LSTM model has two components an encoder and a decoder. This encoder-decoder process first reads the input using an encoder to generate a high-dimensional vector of "thought" reflecting the meaning of the sentence; Then, to produce the translated (normalized) text, a decoder processes the "thought" vector. The encoder is a three-layer RNN encoder that translates a high-level feature representation into the input character sequence. The second part, 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. The main advantage of this approach is the ability to train a single end-to-end model directly on source and target sequences.

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

2.3.2 Results



((a)) Accuracy ((b)) Loss

Figure 4: LSTM result

Model achieves a validation loss of 0.0802 and validation accuracy of 0.9807 (98%). But on particular dates, measures, and several acronyms, (called LETTERS in the semiotic notation) the model does not work well.

Chapter 3

Conclusion

The aim of this project is to “Automate the process of developing text normalization grammars via machine learning”. A text normalization system needs to be able to accurately predict non-trivial normalizations that it hasn’t been trained on before (that’s the entire point). A good text normalization system must be generalized i.e. needs to be able to accurately predict the normalization of the input that it hasn’t been trained on before (unseen data). However, LSTM model perform well and tends to generate complete sense and canonical format for the unseen input. The XGBoost or Extreme Gradient Boosting method which has quite often proved to be a winning method. It operates as a boosted random forest method and has shown comparatively good results. In the other hand, we have introduced a neural model for text normalization that utilizes a character-level sequence-to-sequence model to handle problematic OOV cases. The Sequence-to-Sequence model give us a good performance in some classes and low performance for the others and this is due to the different amount of training data in each class. In addition, lack of the computer we didn’t spend much time to tune the hyper-parameters of the model, so we just apply the most common used values.

Bibliography

[1] Peter Ebden and Richard Sproat. The kestrel tts text normalization system. Natural Language Engineering, 21(3):333, 2015.

[2] Maryam Zare and Shaurya Rohatgi. Deepnorm-a deep learning approach to text normalization. arXiv preprint arXiv:1712.06994, 2017.

[3] Richard Sproat and Navdeep Jaitly. Rnn approaches to text normalization: A challenge. arXiv preprint arXiv:1611.00068, 2016.

[4] Tianqi Chen and Carlos 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.

[5] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. Advances in neural information processing systems, 27:3104–3112, 2014.

[6] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. Deep learning, volume 1. MIT press Cambridge, 2016.

[7] Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: Summarization with pointer-generator networks. arXiv preprint arXiv:1704.04368, 2017.

[8] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. Modeling coverage for neural machine translation. arXiv preprint arXiv:1601.04811, 2016.

[9] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259, 2014.