iTalk

A 3-Component System for Text-to-Speech Synthesis

Chris Lin, Qian (Sarah) Mu, Yi Shao
Dept. of Statistics, Management Sci. & Eng., Civil Eng.
Stanford University
Stanford, CA, USA
{clin17, sarahmu, yishao} @ stanford.edu

***Abstract*—*Text-to-speech (TTS) synthesis is the text normalization from written input form to spoken output form, with applications in intelligent personal assistants (IPAs), voice-activated emails, and message readers for the visually impaired. In this project, we built a 3-component TTS text normalization system with token-to-token Naïve Bayes (NB) classifiers, a token-to-class Support Vector Machine (SVM) classifier, and hand-built grammar rules based on regular expression. The performance of this system reached a development accuracy of 99.36% and a testing accuracy of 98.88%. These results are comparable to those obtained by deep learning approaches.*

Keywords*—*Machine Learning; Natural Language Processing; Text Normalization; Support Vector Machine; Naïve Bayes

I. INTRODUCTION

In natural language processing (NLP), the transformation of one text form to another is known as text normalization. In particular, text-to-speech (TTS) synthesis is the normalization of a text from its written form to its spoken form. For example, the original form of "379.9ft" would be verbalized as "three hundred and seventy-nine point nine feet tall". The broad application of TTS synthesis on NLP includes intelligent personal assistants (IPAs), voice-activated emails, and voice response systems. There are also TTS applications that read messages, books, and navigation instructions for the visually impaired. In this project, we aim to provide an efficient TTS system that can be used across different applications.

In TTS synthesis, all written tokens (defined as clause-separating punctuations and whitespace-separated words) are grouped into semiotic classes such as measure, date, cardinal number, plain text, and others. Theoretically, each semiotic class has a specific grammar rule that normalizes a written input form to the corresponding spoken output form [1].

Specifically, we incorporated machine learning algorithms and hand-built grammar rules to build a 3-component TTS system. The machine learning algorithms included a token-to-token Naïve Bayes classifier and a token-to-class Support Vector Machine (SVM) classifier. The input for the token-to-token Naïve Bayes classifier is a written token (e.g. 16ft), and the output is a predicted spoken form (e.g. sixteen feet). In the token-to-class SVM classifier, the input is the term frequency-inverse document frequency (TF-IDF) of a written token (with each character defined as a term and each token as a document) [2], and the output is a predicated semiotic class (e.g. measure). Based on the predicted semiotic class, the corresponding grammar rule is applied to the written token to output a predicted spoken form.

The code base and the accuracy calculations for the token-to-token Naïve Bayes classifier were shared with a final project in CME193*.

II. RELATED WORK

One approach of TTS text normalization focuses on developing grammar rules that map from written input form to spoken output form. The prominent strategy is to treat the normalization as weighted finite-state transducers (WFSTs). The transducers are constructed and applied using lexical toolkits that allow declarative encodings of grammar rules [3], [4]. This approach works well for standard language forms and was adopted by the multilingual Bell Labs TTS system [3]. On the other hand, this approach requires linguistic expertise to develop the transducers. The 3-component system in this project is similar to this strategy because it also includes hand-built grammar rules. However, since the focus of this project is machine learning, simple grammar rules using regular expression were used instead of WFSTs.

Machine learning algorithms such as Classification and Regression Trees (CART), perceptrons, decision lists, and maximum entropy rankers, combined with lexicon alignment and n-gram, have been applied to TTS text normalization [5-7]. These methods negate the need to develop grammar transducers. Nevertheless, the performance of these methods, as the respective authors suggested, did not meet the standard for a working TTS application. Our methodology in this project mostly aligns with this framework of combining linguistic model and machine learning and produced similar results.

Recently, deep learning methods—particularly recurrent neural networks (RNNs) and long short-term memory (LSTM) models—have been combined with vector space representation and word embedding to yield good results [8-10]. These models tend to incorrectly normalize tokens of certain semiotic classes such as number and measure [10]. Despite of its computational cost, the deep learning approach is the most promising because it can be generalized to non-standard language forms such as social media messages [11-12].

III. DATASET & FEATURES

Our dataset was provided by Google's Text Normalization Research Group through a data science competition hosted on Kaggle [13]. The dataset consists of 8.9 million tokens of English text from Wikipedia, divided into sentences and run

* Computational & Mathematical Engineering 193 (Introduction to Scientific Python) at Stanford University. Project by CL and QM.

through the Google TTS system's Kestrel text normalization system to generate the spoken output form [14]. Even though the target outputs were produced by a TTS system and would not perfectly resemble a hand-labeled dataset, the nature of this dataset does not conflict with our goal of developing a system of TTS text normalization. The Kestrel system also labeled each token as one of 16 semiotic classes. The dataset was split into 6.6 million tokens for training, 1 million tokens for evaluation during training (the development set), and 1.3 million tokens for testing. Examples of data are shown in Table 1.

For token-to-class classification, written tokens need to be represented in numeric values. We used the bag-of-words model with a bag of 162 English characters found in our dataset (discarding 2,933 non-English characters such as Arabic and Chinese characters). With each character defined as a term and each token as a document, term frequency (TF) and L2-normalized term frequency-inverse document frequency (TF-IDF) were constructed for each written token [2]. These features are appropriate since the semiotic classes tend to have different distributions of characters.

Table 1. Examples of Class

written token	semiotic class	spoken token
Blanchard	PLAIN	Blanchard
2007	DATE	two thousand seven
200	DIGIT	two o o
BC	LETTERS	b c
1GB	MEASURE	one gigabyte
32	CARDINAL	thirty-two
#	VERBATIM	number
.	PUNCT	.

IV. METHODS

A. Token-to-Token Naïve Bayes Model

In the first model, we trained a token-to-token Naïve Bayes (NB) for each written token in the training set. To prevent common stop words from having dominant prior probabilities, we restricted the outcome space to only the normalized spoken tokens of a given written token. For a written token, the indices of its outcome space 1, …, k_x, and a spoken token y, our objective is to maximize the following likelihood with respect to its parameters:

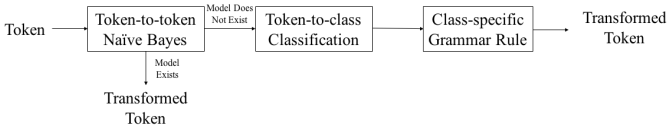
$$\mathcal{L}(\phi_y, \phi_{x|y}) = \prod_{i=1}^{m} \prod_{j=1}^{k_x} p(x^{(i)} = 1 | y = j) p(y = j) \quad (1)$$

where $\phi_y = (\phi_{y=1}, \dots, \phi_{y=k_x})$ with $\phi_j = p(y = j)$, and $\phi_{x|y} = (\phi_{x|y=1}, \dots, \phi_{x|y=k_x})$, with $\phi_{x|y=j} = p(x = 1|y = j)$. After training, we obtained a set of token-to-token NB models.

B. 3-Component System

Token-to-token Naïve Bayes has the advantages of computationally efficient and high accuracy if the token has been seen in the training set (i.e. model exists for this token). But Naïve Bayes is unable to reliably predict new tokens that have never been seen before (i.e. model does not exist), especially for classes that have number such as 'CARDINAL' and 'DATE'. In order improve the accuracy of predicting new tokens, we built a token-to-class classifier and then applied class-specific grammar rule. Finally, a 3-component system (Figure 1) is adopted to take advantages of both Naïve Bayes and classifier-based prediction.

Figure 1. Architecture of the 3-Component System



C. Multinomial Naïve Bayes as Token-to-Class Classifier

In the token-to-class classification of the 3-component system, we trained a multinomial Naïve Bayes in the standard maximum likelihood formulation with Laplace smoothing. Specifically, the distribution is parameterized by vectors $\theta_y = (\theta_{y1}, \dots, \theta_{yn})$, for each semiotic class y, where $n = 162$ is the number of characters in the bag-of-words model, and $\theta_{yj} = p(x_j|y)$. In particular, the smoothed maximum likelihood estimator is the following:

$$\hat{\theta}_{yj} = \frac{N_{yj} + 1}{N_y + n} \quad (2)$$

where N_{yj} is the number of times character x_j appears in class y, and N_y is the total count of all characters for class y. TFs were used as input for this method.

D. SVM Classifier

Support Vector Machine (SVM) method is used to predict the class of each token. An 'one-vs-the-rest' scheme is adopted, which means each class is separated against all other classes and totally 16 models are built for 16 classes. L2-SVM is adopted because it is less susceptible to outliers and more computationally stable:

$$\begin{aligned} &minimize \quad \frac{1}{2}\|w\|^2 + \frac{C}{2}\sum_{i=1}^{2} \xi_i^2 \\ &subject\ to \quad y_i(w^T x_i + b) \geq 1 - \xi_i \\ &\qquad \xi_i \geq 0 \end{aligned} \quad (3)$$

where w, b, ξ_i is the optimization variables; C is the penalty. In our dataset, some classes (e.g. 'DIGIT') have significantly less data points than other classes (e.g. 'PLAIN'). Using same penalty parameters C for all classes (named 'unbalanced L2-SVM' here after) will results in low prediction accuracy for classes that have small data size. As a result, we tried using different penalty number for different classes in order to balance the class weight (named 'balanced L2-SVM' hereafter). The principle of balanced L2-SVM is multiply penalty C by a class weight for each class:

$$C_i = Class\_weight \cdot C = \frac{\#\ data\ points}{\#\ data\ points\ in\ class\ i} \cdot C \quad (4)$$

V. RESULTS & DISCUSSION

In our first experiment, we studied the performance of the token-to-token NB classifier. Here, if a written token did not have a corresponding token-to-token NB model, the written token was normalized as it was. This resulted in 99.81% accuracy in the training set and 98.97% accuracy in the development set, suggesting that the token-to-token NB classifier by itself can have high accuracy compared to the benchmark of 93.34% (normalizing all written tokens as they are) (Table 4).

The token-to-class classification with multinomial NB resulted in an overall accuracy of 96.98% in the training set and 96.96% in the development set. The SVM classifier performed better than the multinomial NB classifier in token-to-class classification (Table 2). Therefore, we focused our improvement effort on the SVM classifier.

Different penalty parameters (C=0.01~4) are tried for both balanced L2-SVM and unbalanced L2-SVM models. Slight fluctuation occurs because the underlying C implementation uses a random number generator to select features when fitting the model. Generally, training set accuracy matches the development set accuracy, and larger penalty number results in higher accuracy (Figure 2-3).

Figure 2. Balanced SVM Parameter Tuning

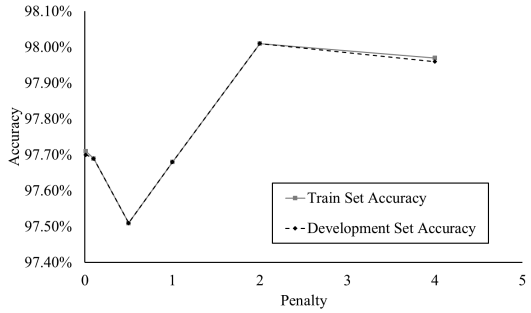
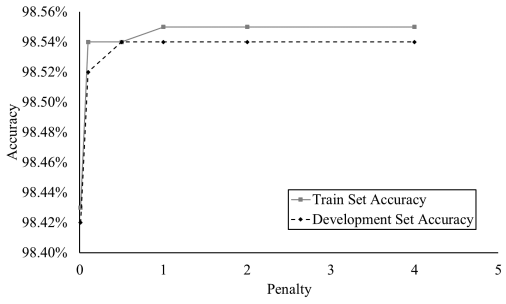


Figure 3. Unbalanced SVM Parameter Tuning



Based on parameter tuning results, unbalanced L2-SVM with penalty parameter of 0.5 and balanced L2-SVM with penalty parameter of 2 is adopted and compared. As shown in Table 2, the unbalanced model out-performed the balanced model in accuracy, precision as well as recall by 0.001% to 0.005%. The accuracy, precision and recall for each class using unbalanced SVM model are summarized in Table 3.

Table 2. Comparison of Average Metrics for Unbalanced and Balanced SVM Models

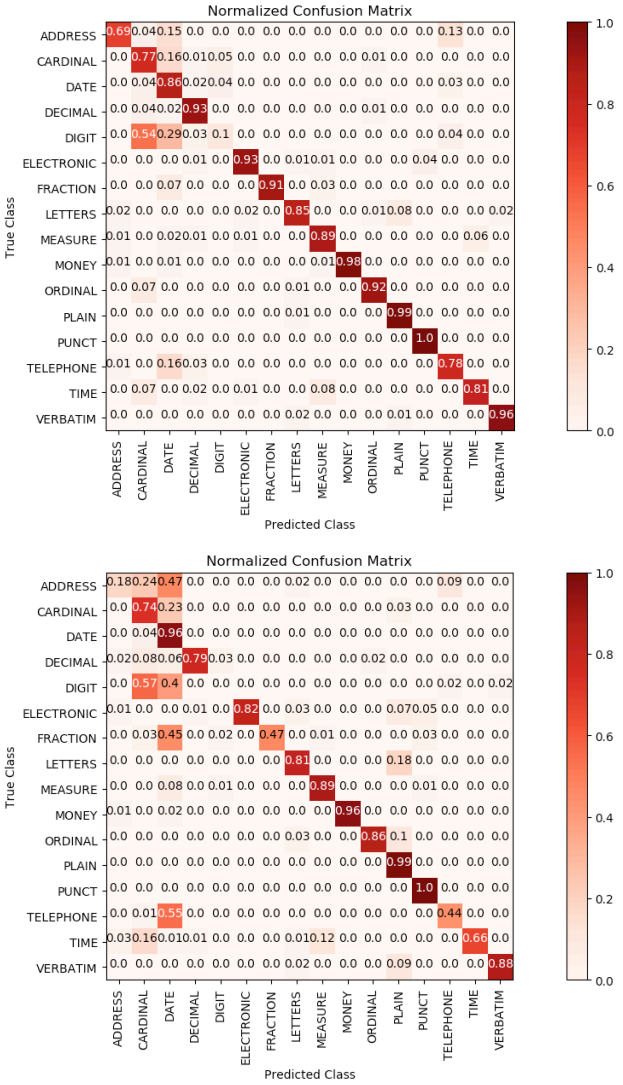
Parameter	Accuracy	Precision	Recall
Unbalanced	98.54%	98.50%	98.54%
Balanced	98.01%	98.45%	98.01%
Percent Out-performance	0.005%	0.001%	0.005%

Table 3. Class-wise Metrics for Unbalanced Model

Class	Accuracy	Precision	Recall
ADDRESS	0.18	0.16	0.18
CARDINAL	0.74	0.87	0.74
DATE	0.96	0.87	0.96
DECIMAL	0.79	0.95	0.79
DIGIT	0	0	0
ELECTRONIC	0.82	0.84	0.82
FRACTION	0.47	0.63	0.47
LETTERS	0.81	0.79	0.81
MEASURE	0.89	0.98	0.89
MONEY	0.96	0.99	0.96
ORDINAL	0.86	0.97	0.86
PLAIN	0.99	0.99	0.99
PUNCT	1	1	1
TELEPHONE	0.44	0.8	0.44
TIME	0.66	0.96	0.66
VERBATIM	0.88	0.86	0.88

Normalized confusion matrix of token-to-class classification for the unbalanced and balanced model is shown in Figure 4. For both balanced and unbalanced L2-SVM model, the matrices have large diagonal values, which indicates high classification accuracy. The classes with higher confusion are mostly associated with numbers, such as ‘CARDINAL’, ‘DIGIT’ and ‘DATE’. These classes share similar characters (i.e. similar TF-IDF input) and our model only considered the token itself, thus it would be hard to correctly classify these classes. This number-related-classes classification problem is also discussed in [10]. Comparing unbalanced L2-SVM to balanced L2-SVM model, the prediction accuracy for classes that have small data size, such as ‘DIGIT’ and ‘FRACTION’, has improved due to applying higher class weight to them.

Figure 4. Normalized Confusion Matrix for Balanced (top) and Unbalanced (bottom) SVM Class-wise Prediction



The token-to-token NB and 3-component system accuracy on training and development set is compared in Table 4. Although token-to-token NB and 3-component system produce similar results in training set, 3-componenet system generates higher accuracy in development set because it has higher accuracy in predicting unseen tokens. As a result, 3-component system based on SVM classifier is chosen as our final model and the test accuracy reaches 98.88%.

Table 4. Result Summary

Model	Training	Dev.	Test
Set size	6,600,000	1,000,000	1,300,000
Token-to-token NB[1]	99.81%	98.97%	98.88% (Token-to-token NB + SVM classifier)
Token-to-token NB + SVM classifier[2]	99.81%	99.36%	
Token-to-token NB + NB classifier[3]	99.81%	99.32%	

Note:
1. Benchmark accuracy: 93.34% (spoken=written)
2. Used TF-IDF as input and 3-component system
3. Used TF as input and 3-component system

Finally, to direct the future work an error analysis is performed by manually bringing the accuracy of each component to 100%. The improvement of final development set prediction accuracy is shown in Table 5. Thus, improving grammar rules will be the primary task for future work.

Table 5. Error Analysis of the 3-Component System

Component	Accuracy Improvement
Token-to-token NB	0.20%
SVM classifier	0.18%
Grammar Rules	0.26%

VI. Conclusion & Future Work

As shown in the confusion matrix, we are unable to classify some tokens based on the information from token itself (e.g. 'FRACTION' vs 'DATE'). Thus, the sentence information would be helpful to improve class classification accuracy. Recurrent Neural Network, which has Long Short-Term Memory (LSTM), will be promising in this case. Also, heavy human work is needed to build the class-specific grammar rules, and the rules are unable to evolve with the development of language. It would be helpful to build an algorithm that learns the grammar rules for each class, and Recurrent Neural Network will still be very promising [10] as discussed in related work.

CONTRIBUTION

CL: implemented token-to-token NB and the 3-component system; performed error analysis. QM: conducted data exploration; constructed confusion matrices and evaluation metrics; explored k-nearest neighbor classifier. YS: data exploration (step 2 in milestone); token-class-wise NB; developed and tuned SVM classifier and error analysis.

REFERENCES

[1] P. Taylor, *Text-to-Speech Synthesis*. Cambridge: Cambridge University Press, 2009.

[2] G. Salton and C. Buckley, "Term-Weighting Approaches in Automatic Text Retrieval," *Inf. Process. Manag.*, vol. 24, no. 5, pp. 513–523, 1988.

[3] R. Sproat, "Multilingual text analysis for text-to-speech synthesis," *Nat. Lang. Eng.*, vol. 2, no. 4, pp. 369–380, 1996.

[4] B. Roark, R. Sproat, C. Allauzen, M. Riley, J. Sorensen, and T. Tai, "The OpenGrm open-source finite-state grammar software libraries," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, 2012, pp. 61–66.

[5] F. Mana, P. Massimino, and A. Pacchiotti, "Using Machine Learning Techniques for Grapheme to Phoneme Transcription," in *Interspeech*, 2001.

[6] R. Sproat, "LIGHTLY SUPERVISED LEARNING OF TEXT NORMALIZATION : RUSSIAN NUMBER NAMES," in *IEEE SLT*, 2010, pp. 436–441.

[7] R. Sproat and K. Hall, "Applications of Maximum Entropy Rankers to Problems in Spoken Language Processing," in *Interspeech*, 2014, no. September, pp. 761–764.

[8] H. Lu, S. King, and O. Watts, "Combining a Vector Space Representation of Linguistic Context with a Deep Neural Network for Text-To-Speech Synthesis," in *8th ISCA Speech Synthesis Workshop*, 2013, pp. 261–265.

[9] P. Wang, Y. Qian, F. K. Soong, L. He, and H. Zhao, "WORD EMBEDDING FOR RECURRENT NEURAL NETWORK BASED TTS SYNTHESIS," in *IEEE ICASSP*, 2015, pp. 4879–4883.

[10] R. Sproat and N. Jaitly, "RNN Approaches to Text Normalization : A Challenge," *CoRR*, vol. 1611.00068, 2016.

[11] G. Chrupała, "Normalizing tweets with edit scripts and recurrent neural embeddings," in *ACL*, 2014.

[12] W. Min and B. Mott, "NCSU SAS WOOKHEE: A deep contextual long-short term memory model for text normalization," in *WNUT*, 2015.

[13] Data downloaded from Kaggle. The Text Normalization Challenge - English Language. Sponsored by Google's Text Normalization Research. Group. https://www.kaggle.com/c/text-normalization-challenge-english-language.

[14] P. Ebden and R. Sproat, "The Kestrel TTS text normalization system," *Nat. Lang. Eng.*, 2014.

CME193 Final Project Report
Team Members: Chris Lin (clin17), Qian (Sarah) Mu (sarahmu)

Text normalization is the transformation of one text form to another. In text-to-speech (TTS) synthesis, the written form of a text is transformed to its spoken form. For example, the written sentence "The CME193 lecture is every Tuesday/Thursday at 10:30am" would be transformed to "the c m e one ninety-three lecture is every thursday tuesday at ten thirty a m." A dataset curated by Sproat and Jaitly contains 1.1 billion pairs of written and spoken tokens (one token is a whitespace-separated string) [1].

In our project, we obtained 1.4 million data points* from the above dataset from Kaggle [2]. As shown in Fig. 1, the majority of the tokens are considered as plain and punctuations.

Instead of implementing a weighted classification tree, we found a more computationally efficient method. For each unique written token we fitted a Naïve Bayes classifier. When predicting the spoken form of a token we used the classifier to find the most likely normalization. When a written token does not appear during the model training, we simply predict the spoken token with the written token itself.

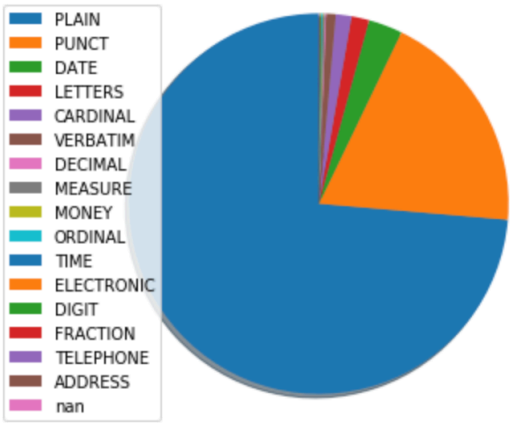


Figure 1. Distribution of token classes in the whole dataset.

The Naïve Bayes classifiers had a training accuracy of 99.0% and a testing accuracy of 99.0%. These are higher than the benchmark of 93.3% (predicting the spoken token with the written token directly).

As shown in Fig 2., most of the incorrect predictions came from tokens of date, letters, and plain. However, if we look at the count normalized by their prevalence in the data set, as shown in Fig 3., we see that most of the incorrect predictions came from tokens of type "telephone", "electronic", and "address." Furthermore, around 94% of the incorrect predictions are the same as the input written tokens, whereas only 1% of the actual spoken tokens are the same as their input. Given these, to improve our model we should develop a strategy to classify unseen written token, especially of type "telephone", "electronic", and "address."

* Since we cannot upload such a large file, we submit a subset of the 1.4-million data set.

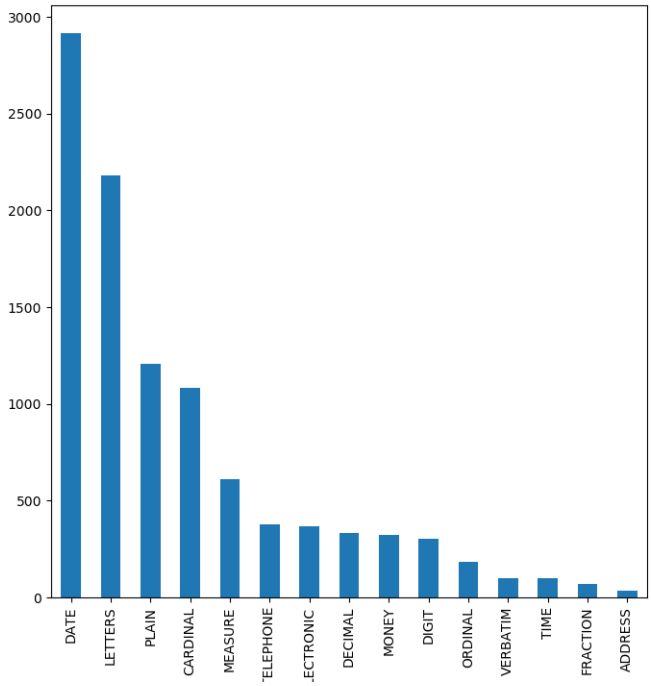


Figure 2. Count of incorrect predictions for each token class.

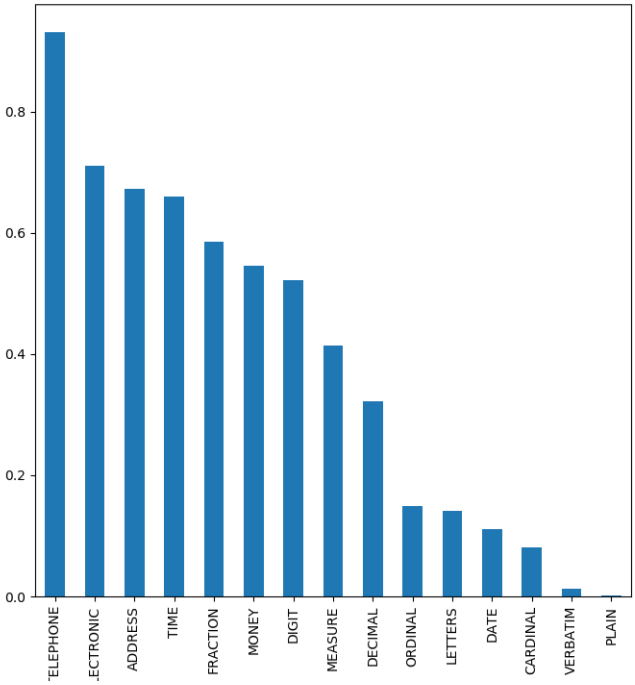


Figure 3. Count of incorrect predictions for each token class, normalized by the total count of each token class in the data set.

References

[1] R. Sproat and N. Jaitly, “RNN Approaches to Text Normalization : A Challenge,” *CoRR*, vol. 1611.00068, 2016.

[2] Kaggle: Text Normalization Challenge – English Language, https://www.kaggle.com/c/text-normalization-challenge-english-language