Team UKNLP: Detecting ADRs, Classifying Medication Intake Messages, and Normalizing ADR Mentions on Twitter

Sifei Han, B.S[1], **Tung Tran**, B.S[1], **Anthony Rios**, B.S[1], and **Ramakanth Kavuluru**, Ph.D[1,2]

[1]Department of Computer Science, University of Kentucky
[2]Div. of Biomedical Informatics, Department of Internal Medicine, University of Kentucky

Abstract

This paper describes the systems we developed for all three tasks of the 2nd Social Media Mining for Health Applications Shared Task at AMIA 2017. The first task focuses on identifying the Twitter posts containing mentions of adverse drug reactions (ADR). The second task focuses on automatic classification of medication intake messages (among those containing drug names) on Twitter. The last task is on identifying the MEDDRA Preferred Term (PT) code for the ADR mentions expressed in casual social text. We propose convolutional neural network (CNN) and traditional linear model (TLM) approaches for the first and second tasks and use hierarchical long short-term memory (LSTM) recurrent neural networks for the third task. Among 11 teams our systems ranked 4th in ADR detection with F-score 40.2% and 2nd in classifying medication intake messages with F-score 68.9%. For the MEDDRA PT code identification, we obtained an accuracy of 87.2%, which is nearly 1% lower than the top score from the only other team that participated.

1. Introduction

Online social networks and forums provide a new way for people to share their experiences with others. Nowadays patients also share their symptoms and treatment progress online to help other patients dealing with similar conditions. Due to the individual differences and other factors that cannot be tested out during clinical trails, patients may experience adverse drug reactions (ADRs) even when taking FDA approved medications. ADRs lead to a financial burden of 30.1 billion dollars annually in the USA [1]. Automatic detection of ADRs is receiving significant attention from the biomedical community. While traditional approaches of reporting to the FDA (phone, online) are still important, given millions of patients are sharing their drug reactions on social media, automatic detection of ADRs reported on online posts may offer complementary valuable signal for pharmacovigilance. Among these posts, some ADRs are mentioned from personal experience while other ADR mentions are observed in other people. To identify whether or not a Twitter user has consumed the medication is also an important task. So is the normalization of different ways of expressing the same event using a standardized terminology. These are the three tasks in the 2nd Social Media Mining for Health Applications Shared Task at AMIA 2017. In task 1 there are 6822 training, 3555 development, and 9961 testing samples; for task 2 there are 7617 training, 2105 development, and 7513 testing samples. In task 3, there are 6650 training and 2500 testing samples. The first task is a binary classification task with F-score for the ADR (positive class) as the evaluation metric. Task 2 is a 3-way classification task where the evaluation metric is the micro-averaged F-score for "intake" and "possible intake" classes. Task 3 is a multiclass classification task where Twitter phrases are mapped to their corresponding MEDDRA PT codes and accuracy is the evaluation metric.

2. Methods

For tasks 1 and 2, we employ traditional methods (e.g. SVMs) and recent deep learning methods (e.g. CNNs) as well as their ensembles as detailed in Section 2.1. For task 3, we use a hierarchical character-LSTM model which is detailed in Section 2.2.

2.1. Tasks 1 and 2: TLMs, CNNs, CNNs with attention mechanism, and their ensembles

For tasks 1 and 2, we used both TLMs, specifically linear SVMs and logistic regression, and deep nets specifically CNNs. In the first task, we simply averaged the probability estimates from each TLM classifier with and without CNN ensemble. The features used in TLMs are itemized below.

- Uni/bi grams: Counts and real values of uni/bi-gram features in the tweet via *countvectorizer* and *tfidfvectorizer*, respectively, from Scikit Learn machine learning package [2].
- Sentiment lexicons (11 features): Four count features from the post based on the positive and negative counts of unigrams and bigrams using the NRC Canada emoticon lexicon[1], four additional sentiment score aggregation features corresponding to the count features, and overall sentiment score of uni-grams/bigrams/uni+bi grams.
- Word embeddings (400 features): The average 400 dimensional vector of all the word vectors of a post where the word vectors ($\in \mathbb{R}^{400}$) are obtained from a pre-trained Twitter word2vec model [3].
- ADR lexicon [4] (7423 features): One Boolean feature per concept indicating whether the concept ID is mentioned in the tweet; rest are count features identifying the number of drugs from a particular source (SIDER, CHV, COSTART and DIEGO_lab) and the number of times different drugs are mentioned in the tweet.
- Negation words (2 features): The first is a count of certain negation words (not, don't, can't, won't, doesn't, never) and second feature is the proportion of negation words to the total number of words in the post.
- PMI score (1 feature): The sum of words' pointwise mutual information (PMI) [5] scores as a real-valued feature based on the training examples and their class membership.
- For task 2 only – handcrafted lexical pairs of drug mentions preceded by pronouns (6 features): The count of first, second, and third personal pronouns with and without negation followed, with potentially other intermediate words, by a drug mention[2]. (e.g., "I did a line of cocaine")

For the CNN models, each tweet is passed to the model as a sequence of words vectors, $[\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_n]$, where n is the number of words in the tweet. We begin by concatenating each window spanning k words, $\mathbf{x}_{j-k+1} \| \ldots \| \mathbf{x}_j$, into a local context vector $\mathbf{c}_j \in \mathbb{R}^{kd_{emb}}$ where d_{emb} is the dimensionality of the word vectors. Intuitively, CNNs extract informative n-grams from text and n-grams are extracted with the use of *convolutional filters* (CFs). We define the CFs as $\mathbf{W}_k \in \mathbb{R}^{q \times kd_{emb}}$, where q is the number of *feature maps* generated using filter width k. Next, using a non-linear function f, we convolve over each context vector,

$$\hat{\mathbf{c}}_j = f(\mathbf{W}\mathbf{c}_j + \mathbf{b}),$$

where $\mathbf{b} \in \mathbb{R}^q$. Given the convolved context vectors $[\hat{\mathbf{c}}_1, \hat{\mathbf{c}}_2, \ldots, \hat{\mathbf{c}}_{n-k+1}]$, we map them into a fixed size vector using max-over-time pooling

$$\mathbf{m}_k = [\hat{c}^1_{max}, \hat{c}^2_{max}, \ldots, \hat{c}^q_{max}],$$

where $\hat{c}^j_{max}$ represents the max value across the j-th feature map such that $\hat{c}^j_{max} = max(\hat{c}^j_1, \hat{c}^j_2, \ldots, \hat{c}^j_{n-k+1})$. To improve our model, we use convolutional filters of different widths. With filters spanning a different number of words $(k_1, k_2, \ldots, k_p)$, we generate multiple sentence representations $\mathbf{m}_{k_1}, \mathbf{m}_{k_2}, \ldots, \mathbf{m}_{k_p}$, where p is the total number of filter widths we use.

When datasets contain many noisy features, it can be beneficial to use a simpler model. Simpler models are less prone to overfitting to the noise in the dataset. We augment the features generated from the CNN with a simple average of all the word vectors in a given tweet

$$\mathbf{m}_{bow} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{w}_i.$$

Now we have $p + 1$ feature representations of the final tweet $[\mathbf{m}_{k_1}, \mathbf{m}_{k_2}, \ldots, \mathbf{m}_{k_p}, \mathbf{m}_{bow}]$. Prior work using CNNs [6] simply concatenated each $\mathbf{m}_j$ to pass to the final fully-connected softmax layer. Rather than concatenating each $\mathbf{m}_{k_j}$, we use self-attention [7] (with multiple attention scores per feature representation) to allow the model to dynamically weight the feature representations. Specifically, we define the j-th attention of the i-th feature representation as

$$\alpha_{j,i} = \frac{exp(e_{j,i})}{\sum_{k=1}^{p+1} exp(e_{j,k})}, \text{ where } e_{j,i} = \boldsymbol{v}_j^T \cdot \tanh(\mathbf{W}_a \mathbf{m}_i),$$

such that $\mathbf{W}_a \in \mathbb{R}^{t \times q}$, $\mathbf{v}_j \in \mathbb{R}^t$, and $\alpha_{j,i} \in [0, 1]$. Intuitively, $\alpha_{j,i}$ represents the importance we give the feature representation $\mathbf{m}_i$. Next, we merge all feature representations into a matrix $\mathbf{M} \in \mathbb{R}^{(p+1) \times q}$. Likewise, given s total attentions, each attention weight is combined to form an attention matrix $\mathbf{A} \in \mathbb{R}^{s \times (p+1)}$. Here, the j-th row of A represents the importance of each $\mathbf{m}_i$ with respect to the attention weights $\mathbf{v}_j$. Finally, we represent each tweet as the weighted sum of the feature representations according to each of the attention weight vectors as

$$\mathbf{h} = vec(\mathbf{AM}),$$

where *vec* represents the matrix vectorization operation and $\mathbf{h} \in \mathbb{R}^{sq}$. Lastly, $\mathbf{h}$ is passed to a final fully-connected softmax layer.

[1] http://saifmohammad.com/WebPages/AccessResource.htm
[2] https://www.drugs.com/drug_information.html

2.2. Task 3: Hierarchical Character-LSTM

The deep model we propose for task 3 realizes a hierarchical composition in which an example phrase is segmented into N constituent words and each word is treated as a sequence of characters. In our formulation, the word at position $i \in [1, N]$ is of character length T^i. Herein, we formulate the model composition from the bottom up. At the character level, word representations are composed using a forward LSTM over each character and its corresponding character class. The former is a case-insensitive character embedding lookup and the latter is a separate label embedding lookup indicating the class of character: lowercase letter, uppercase letter, punctuation, or other. We denote $\mathbf{c}^{i,t}$ and $\mathbf{z}^{i,t}$ as the t-th character and class embedding respectively of the i-th word in the sentence for $t \in [1, T^i]$. For a word at position i, we feed its character embeddings to a forward-LSTM layer with 200 output units such that

$$\overrightarrow{\mathbf{g}}^{i,t} = CLSTM(\mathbf{c}^{i,t} \| \mathbf{z}^{i,t}) \qquad \text{for } t = 1, \ldots, T^i$$

where $\|$ is the vector concatenation operation, $CLSTM$ is an LSTM unit composition at the character level, and $\overrightarrow{\mathbf{g}}^{i,t} \in \mathbb{R}^{200}$ is the output at timestep t. The output state at the last step, or $\overrightarrow{\mathbf{g}}^{i,T^i}$, encodes the left-to-right context accumulating at the last character and is used to represent the word at position i. Next, we deploy a bi-directional LSTM layer at the word level which is useful for capturing the contextual information of the phrase with respect to each word. Concretely,

$$\begin{aligned}
\overrightarrow{\mathbf{h}}^i &= WLSTM^{\rightarrow}(\overrightarrow{\mathbf{g}}^{i,T^i}), \\
\overleftarrow{\mathbf{h}}^i &= WLSTM^{\leftarrow}(\overrightarrow{\mathbf{g}}^{i,T^i}), \\
\mathbf{h}^i &= \overrightarrow{\mathbf{h}}^i \| \overleftarrow{\mathbf{h}}^i \qquad \text{for } i = 1, \ldots, N
\end{aligned}$$

where $\overrightarrow{\mathbf{h}}^i, \overleftarrow{\mathbf{h}}^i \in \mathbb{R}^{200}$, $\mathbf{h}^i \in \mathbb{R}^{400}$, and $WLSTM^{\rightarrow}, WLSTM^{\leftarrow}$ are LSTM units composing words in the forward and backward direction respectively. We then perform a max-pooling operation over all $\mathbf{h}^i$ vectors to produce the feature vector $\hat{\mathbf{h}} = [h_1^{max}, \ldots, h_{2d}^{\max}]$ where $h_j^{\max} = \max(\mathbf{h}_j^1, \ldots, \mathbf{h}_j^N)$. We use a fully-connected layer output layer of unit size m, where m corresponds to the number of MEDDRA codes in the label space. The output is computed as

$$\mathbf{q} = W_q \cdot \hat{\mathbf{h}} + b_q$$

such that $\mathbf{q} \in \mathbb{R}^m$ and W_q and b_q are network parameters. In order to get a categorical distribution, we apply a softmax layer to the vector $\mathbf{q}$ such that

$$p_j = \frac{e^{\mathbf{q}_j}}{\sum_{j=1}^{m} e^{\mathbf{q}_j}}$$

where $p_j \in [0, 1]$ is the probability estimate of the label at index j.

Additional Training Data In addition to the training set provided for this task, we also experiment with using additional external datasets for training. Namely, we use the TwADR-L [8] and training data released as part of SMM2016 task 3. Since these datasets are labeled with CUIs as opposed to MEDDRA codes, we perform a mapping from CUIs to MEDDRA codes by referencing the NLM Metathesaurus; here we keep only examples for which such a mapping exists. This amounts to approximately 905 additional examples.

Model Configuration We train the model with 80% of the training data and use the remaining 20% for validation. We train for 40 epochs with a batch size of 8 and learning rate of 0.01. Here we use the RMSprop optimize with an exponential decay rate of 0.95. The character embeddings are of size 32 and the character class embeddings are of size 8. We apply dropout regularization at a probability of 50% at the feature vector layer. To make the final prediction, we performed model averaging using an ensemble of 10 such models each with a different random parameter initialization and random validation split.

3. Result and Discussion

3.1. Task 1 Results

Table 1 shows our official scores on task 1. The CNN with Attention (CNN-Att) model is observed to outperform the other models. On the training and development data set, we found logistic regression to perform better than SVM; TLM ensemble (model averaging) of two LR models with one SVM model and the base CNN have about the same performance while the averaged model of TLM-ensemble with CNN was the best performer. CNN-Att itself

was slightly better than TLM-ensemble and CNN when considered separately, but CNN-Att was worse off when TLM-ensemble and CNN were combined using model averaging. Our ensemble model has the top precision score among all teams and shows the potential of the ensemble approaches. However, our recall is significantly less than the best performer where it was over 48%. We will further investigate the discrepancies between train and test set performances of various models. Our initial assessment is that the attention model is able to more effectively weight what different CFs are capturing from each tweet. To verify this, we will examine the "popular" *n*-grams of filters (based on values of $\hat{\mathbf{c}}_j$ in Section 2.1) that are consistently being weighted above others by the attention mechanism. In turn, these *n*-grams can be used as additional features in the regular CNN or TML models to potentially improve their performances. We will need to experiment with these additional approaches to improve our recall without major compromises in precision.

Table 1: Task 1: Performance on the test set

	ADR Precision	ADR Recall	ADR F-score
TLM-ensemble	0.459	0.237	0.313
CNN+TLM-ensemble	**0.567**	0.259	0.356
CNN-Att	0.498	0.337	**0.402**

3.2. Task 2 Results

Table 2 shows our official scores on task 2. The CNN-Att model still has the best performance over the other two models we submitted. On the training and development data sets, we found deep nets significantly outperformed the TLMs (and their ensembles), and therefore, in task 2 we focused on incorporating deep nets in all ensembles.

Table 2: Task 2: Performance on the test set

	Prec. for classes 1 & 2	Recall for classes 1 & 2	F-score for classes 1 & 2
CNN+TLM-ensemble	0.688	0.607	0.645
CNNs	0.705	0.666	0.685
CNN-Att	0.701	**0.677**	**0.689**

Since each task allowed submission of only three models, the ones we submitted may not have been the best compared with additional models we built as part of our participation in the challenge. Therefore, we did some new experiments using unsubmitted models and found newer better performing ensembles on task 1 and 2 as shown in Table 3. The results from our CNN and CNN-Att models are from averaging ensembles of ten models each using both stratified and random 80%–20% splits. We did this because the test dataset distribution may vary from training data, and we believed stratified split may cause overfitting. Therefore, we felt that with the help of random splits it may help our model's ability to handle test data with a different distribution. In this particular case, the test data distribution is similar to that of the training data; so when we only consider the use of stratified splitting to tune the parameters, our results improved. In task 1, we found the ensemble model with CNN-Att and CNN with logistic regression had almost 1% improvement in F1 score. In task 2, we found CNN-Att trained with stratified splitting has an F1 score that is 0.4% higher than that of our submitted model and this score is equal to winning team's performance.

Table 3: Experiments on unsubmitted models

	Precision	Recall	F-score
CNN_Att+CNN+LR (task1)	0.483	0.358	0.411
CNN_Att (averaging_stratified_only) (task2)	0.701	0.686	0.693

3.3. Task 3 Results

We disclose our results for task 3 in Table 4. For our initial experiments, we performed testing on 30 different runs each evaluating on a different random held-out development set; the results are averaged and recorded in the second column of the table. We submitted only two models and these are marked accordingly in the table. Our first submission, the Hierarchical Char-LSTM model as described in Section 2.2, performed best at 87.4% on the validation set which is consistent with its performance on the actual test at 87.2%. Our second submission, which is the same as the first except it is trained using additional external datasets performed reasonably but did not generalize to the test set as initially hoped.

The *flat* character-based models performed poorly on both the validation and the test set but nevertheless saw a slight increase on the test set. The Hierarchical Char-CNN model results were underwhelming on the validation; however, on the test set it is observed to outperform even the Hierarchical Char-LSTM models which is unexpected. Furthermore, we experimented with appending pretrained word embeddings (trained on the Twitter corpus [3]) to the word representation layer of the model and this resulted in poor performance on the validation set at only 83.3% for either variant. However, the results on the testing set were much more competitive.

Table 4: Accuracy of candidate models for task 3.
* Models that were submitted.

	Dev.	Test
Flat Char-CNN	82.4	84.8
Flat Char-LSTM	81.7	84.7
Hierarchical Char-CNN	85.0	**87.7**
Hierarchical Char-LSTM *	**87.4**	87.2
Hierarchical Char-LSTM with External Training Data *	85.6	86.7
Hierarchical Char-CNN with Word Embeddings	83.3	87.4
Hierarchical Char-LSTM with Word Embeddings	83.3	86.0

Acknowledgments

Our work is primarily supported by the National Library of Medicine through grant R21LM012274 and the National Cancer Institute through grant R21CA218231.

References

[1] Sultana J, Cutroneo P, Trifirò G. Clinical and economic burden of adverse drug reactions. Journal of pharmacology & pharmacotherapeutics. 2013;4(Suppl1):S73.

[2] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 2011;12:2825–2830.

[3] Godin F, Vandersmissen B, De Neve W, Van de Walle R. Multimedia lab@ ACL W-NUT NER shared task: named entity recognition for twitter microposts using distributed word representations. ACL-IJCNLP. 2015;p. 146–153.

[4] Sarker A, Gonzalez G. Portable automatic text classification for adverse drug reaction detection via multi-corpus training. Journal of biomedical informatics. 2015;53:196–207.

[5] Bouma G. Normalized (pointwise) mutual information in collocation extraction. Proceedings of GSCL. 2009;p. 31–40.

[6] Rios A, Kavuluru R. Convolutional neural networks for biomedical text classification: application in indexing biomedical articles. In: Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics. ACM; 2015. p. 258–267.

[7] Lin Z, Feng M, Santos CNd, Yu M, Xiang B, Zhou B, et al. A structured self-attentive sentence embedding. In: Proceedings of the 5th International Conference on Learning Representations; 2017. .

[8] Limsopatham N, Collier N. Normalising Medical Concepts in Social Media Texts by Learning Semantic Representation. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1); 2016. p. 1014–1023.