CSE 635: Social Media Mining for Health Monitoring

Mohd Ehtesham Shareef
Computer Science and Engineering
University at Buffalo
mohdehte@buffalo.edu

Priya Rao
Computer Science and Engineering
University at Buffalo
prao4@buffalo.edu

Abstract

In this paper, we present an end to end architecture to identify tweets which contain an adverse drug reaction (ADR), extract this ADR if present and map it to a MedDRA preferred term (PT) by means of various NLP and ML models. The training and validation corpus for these tasks have been included as part of the SMM4H 2020 Shared Task (CodaLab competition – Task 2 and 3 respectively) and all metrics elaborated in this paper have been against the validation dataset. We demonstrate various models and their implementation in achieving optimal results by means of their evaluation metrics such as strict and relaxed f1-scores, accuracy, precision and recall as well as by data visualizations.

1 Introduction

Drugs by themselves can produce side effects of which not all are documented or predicted by pharmaceuticals or researchers alone. A comprehensive knowledge of adverse drug reactions (ADRs) can reduce their harmful impacts on patients as well as benefit the health system in diagnosing accordingly. And while it is not impossible to get such data from passive spontaneous reporting system databases, such as the Federal Drug Administration's Adverse Event Reporting System (FAERS), such systems may be restrictive due to delayed, biased or underreporting of events affecting the patients. And so, to address this drawback, electronic health records (EHRs) are maintained but even this method did not prove beneficial due to lack of ADR information. Social media on the other hand, though not cited as an official source, contained a plethora of data, which was an ideal complement to the EHR database. This data is predominantly the most widespread means of channeling an individual's expressions. With popular social media platforms like Twitter which receives about 500 generic million tweets per day there would be ample information to fill up the gaps in understanding the EHR dataset as well. Additionally, statistically significant correlations have been identified between certain ADRs described in Twitter data and those reported in FAERS, suggesting that Twitter is a viable drug safety data source. The only drawback in doing so would be that mining such data involves handling linguistic variations and semantic complexities in terms of how people express medication-related concepts and outcomes.

This paper targets at classification of the tweets of adverse drug reactions (ADRs) and mapping the extracted colloquial mentions of ADRs in the tweets to standard concept IDs in the Medical Dictionary for Regulatory Activities (MedDRA) vocabulary from these expressions which correspond to unique MedDRA preferred terms (PT). It is based off the shared tasks (task 2 and 3 respectively) defined in the SMM4H 2020 competition.

2 Literature Study

The SMM4H Shared Task is organized every year with either new or improved datasets and relatively similar task definitions. We referred to a few papers which explained the approaches taken in the previous years to gain a better understanding of the challenges and to experiment with any new hybrid systems that could prove to be more beneficial.

2.1 Baseline paper

The baseline paper defines an approach to utilize this data by dividing the process into 4 subtasks and how the approach for the said tasks has shifted over the years. The paper describes how the participating teams used baseline models built using SVM and CRF and then compared the performances of other approaches with their baseline models. There has been a recent shift from recurrent neural networks and pre-trained word embeddings like GloVe and word2vec to approaches like word embeddings pre-trained with the Bidirectional Encoder Representations from Transformers (BERT) which has led to better results.

2.2 Reference paper on joint RNN and CRF

Yet another paper explores a joint RNN and CRF model which produced better results than their individual counterparts combined. The paper also elaborates several of the individual models being used for this purpose:

1. CRF model with the following baseline features: each word itself with a part-of-speech (POS) tag, the suffixes and prefixes to 6 characters in length, and a window of two words in both directions (backward and forward) from the current word.
2. A feature-rich CRF-based approach which utilizes the following features: baseline contextual features, dictionaries, and cluster-based and distributed word representation.
3. Deep bidirectional RNN approach with a softmax layer, in particular, LSTM and GRU, where the combination of the network's outputs is fed into a fully connected layer with softmax activation.

By comparing the above models with their joint model, they arrived at the conclusion that a combination of RNN and CRF led to quality improvement for ADR extraction from free-text reviews. The second conclusion that was highlighted in this paper was that concatenating input word embeddings with an extra embedding vector based on a character-level CNN significantly improved the results. And finally, they also observed that GRU-based recurrent architectures consistently outperformed the LSTM-based architectures in the exact matching experiment.

2.3 Reference paper on BiLSTM+RNN

Another paper advocated the use of BLSTM-RNN model which essentially translated to using a bidirectional model because internally it combined 2 RNNs: a forward RNN which processed the sequence from left to right, and a reverse RNN which processed the sequence from right to left. The outputs of both RNNs were averaged for each token to compute the model's final label prediction. Their model takes as input an index that maps the current token to a column in a word embedding matrix. The column is a real-valued numeric representation known as a word embedding. They then tweaked this model to work with three methods. Method 1 (BLSTM-M1) where the word-embedding values were randomly initialized (standard normal scaled by the square root of our vocabulary size) and treated as learnable parameters. Method 2 (BLSTM-M2) where the word-embedding values were initialized with a publicly available pre-trained dataset and treated as learnable model parameters. Method 3 (BLSTM-M3) where the word-embedding values were initialized as fixed constants.

The BLSTM-M3 model significantly outperformed all other models in terms of F-Score and even though the CRF model had the highest precision, the BLSTM-M3 model had a significantly better balance of precision and recall, as reflected in its F-measure, and thus represented a state-of-the-art performance. Both BLSTM models initialized with pretrained embeddings (BLSTM-M2 and BLSTM-M3) performed significantly better than the baselines.

3 Dataset

We obtained the twitter corpus as part of the SMM4H 2020 Shared Task itself. The training corpus of the first task of classifying the tweets as ADR and Non-ADR contained about 20,544 tweets while the validation corpus consisted of 5134 tweets. Aside from the tweet, we were also provided with the tweet id, the user id and class as the target label (0 for Non-ADR and 1 for ADR) as can be seen below:

	tweet_id	user_id	class	tweet
20534	458799806639800320	361476905	1	@joshthechosen1 not as focused no try focalin ...
20535	469207151496675328	74866693	0	@lesimonator @ccfa he wants me to stay on humi...
20536	521138647610191872	49374146	0	like why the fuck is my asthma so bad that i'm...
20537	526032609214599170	1410794000	1	@cdcgov 10/25/09 cdc response i contracted h1n...

The corpus was however not a balanced one. Roughly only 9% of the tweets contained an ADR for both the training as well as the validation dataset. The second task and the third task shared the twitter corpus provided as task 3 of the SMM4H 2020 Shared Task. The training and validation dataset contained about 2246 and 560 entries respectively with each row containing the tweet id, the begin and end position of an ADR extraction (if present), the ADR extraction, the drug (if any) mentioned in the tweet, the tweet itself and the corresponding MedDRA code and term (preferred term).

	tweet_id	begin	end	type	extraction	drug	tweet	meddra_code	meddra_term
2241	347921687729299456	NaN	NaN	NaN	NaN	NaN	i will admit that most people would agree zypr...	NaN	NaN
2242	348383854089871360	72.0	91.0	ADR	gain so much weight	zyprexa	that zyprexa really makes your vocal chords ma...	10047896.0	weight gain
2243	348595096352075776	NaN	NaN	NaN	NaN	zyprexa	@incubator04 i'm hoping that i get on the zypr...	NaN	NaN

Both the training and the validation dataset contained about 65% of the tweets as containing an ADR and were hence mapped to a MedDRA term and code. Also, since one tweet could contain multiple ADR mentions, this would mean that there were multiple mappings possible where each mention would map to a different MedDRA term and code. For this reason, a single tweet was present multiple times in the dataset to allow for this scenario. Although there are more than 19,000 MedDRA preferred terms present in the official database, we found that the corpus given to us had only about 477 unique terms.

4 Preprocessing

Tweets are generally expressed in an informal manner with the presence of abbreviations, slangs, contractions and much more. For this reason, there arises a need to pre-process or clean the tweets to ensure better training in any model. Although not all three tasks demanded the same level of preprocessing, there were steps that were employed to all three:

1. We removed parts of the tweet which did not contribute to the context of the tweet itself. This included hyperlinks, emoticons, emojis, certain punctuations (not including apostrophe and hyphens) and even the word “rt” when it occurs at the beginning of a tweet.
2. We expanded contractions (such as wanna to want to) and abbreviations (such as OMG to oh my gosh) and contracted slangs (such as amaziiiiing was contracted to amazing) using a prebuilt dictionary.
3. We tokenized the tweet into tokens (words, hyphenated words, numbers) which would enable us to easily train any model thereafter.

For task 1, we attempted stemming as well as lemmatization but this did not contribute to better results. Also, in case of context-based models stemming proved to be causing a drop in the precision and recall rather than improving it. As part of task 2, for the baseline model, if the tokens would fall within the range of the begin and end position, we had an ‘ADR’ tag assigned else ‘O’ tags were assigned for each “outside” token in the tweet. For the final model however, we performed an I-O-B tagging (Inside-Outside-Beginning). This was done by parsing the tweet for the begin and end position provided in the corpus with the first token within this range being assigned the ‘B-ADR’ tag and the subsequent ones

being assigned a ‘I-ADR’ tag. All tokens outside this range were assigned the ‘O’ tag. The third task employed only the pre-processing steps mentioned above.

5 Architecture

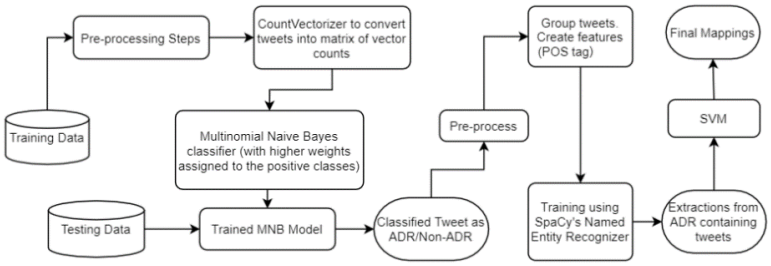
As part of the implementation of this project we started off with several naïve systems to establish a baseline model. After a baseline had been established we explored advanced and/or hybrid systems to achieve better precision and recall.

5.1 Baseline Model

Task 1
As part of the baseline model, a multinomial naïve bayes classifier was used to train the tweets to classify them into an ADR tweet or a Non-ADR tweet. We used the sklearn CountVectorizer to turn the tweets to a matrix of token counts. Due to the highly imbalanced dataset, it was important that the positive classes have more weight in the training process than the negative classes. By doing this, we are boosting the loss function towards the positive class. This methodology gave us the best performance out of the other models we tried, which included a multinomial naïve bayes without using weights, a multinomial naïve bayes with the TF-IDF vectorizer and a Recurrent Neural Network as we can see later in the results section.

Task 2
In addition to the pre-processing steps mentioned above, we also employed a few more additional steps for this task. Aside from tokenization, every individual token had to be tagged according to the begin and end spans of the tweet given. So, for the tokens that fell in the range of this span, we tagged them as ADR and for those that did not, we tagged them as ‘O’ or Outside. Also, every token had a POS tag associated with it which was retrieved by using the NLTK POS tagger. Since a tweet can have multiple ADR’s and the dataset had a new line for every ADR present in the same tweet, the data had to be grouped in a way such that there was only one line for every unique tweet, and all its corresponding information about the ADRs and spans were grouped into that one line. This was important because in our experiments, the results drastically varied when this preprocessing step was skipped.We used a Conditional Random Fields (CRF) model as the baseline model. To facilitate creating new features to supplement the existing data, features like the last 3 letters of the token, whether the token is in upper case, the token is in lower case, if the token is a number, the POS tag of the token, were used. These features were passed onto the CRF model to train the NER tagger. Another approach involved using the spaCy library for training the NER tagger. The data was converted to a form accepted by spaCy. Since there cannot be overlaps in the different entities for it to be trained by spaCy, for tweets where there were multiple ADRs and the ADRs were overlapping, the biggest span had to be considered. For eg. “tremors in hands and legs” has four ADRs present in it according to the training dataset, i.e tremors, tremors in hands, tremors in legs, and tremors in hands and legs. For the data to be trained by spaCy, only the largest span, i.e tremors and hands and legs” had to be considered.

Task 3
Our baseline model was using an SVM to classify these extractions into their respective MedDRA terminologies. The pipeline for this model consisted of 3 main parts - CountVectorizer, Tf-idf transformer and SGDClassifier which by default, fits a linear support vector machine (SVM) and this was trained for 50 epochs. This methodology provided us with the best results. Aside from this model, the features were also trained using Logistic Regression, Word2Vec with Logistic Regression, Rocchio’s algorithm and Bagging. The complete baseline end-to-end system can be outlined as follows:



5.2 Final Proposed Model

Task 1

Our final model was that of using a RoBERTa model with the roberta-large pre-trained model. But before we could finalize on this model, we also tried out BERT (with pre-trained models of bert-base-uncased, bert-large-uncased and bio-bert). We will briefly attempt to explain all before diving into the RoBERTa model. Since we were using BERT to train a text classifier, we took the pre-trained BERT model, added an untrained layer of neurons on the end, and trained the new model for our classification task. The reason why we employed BERT instead of a deep learning model was because it facilitated quicker development as well as would provide us with better results. We used the BertForSequenceClassification pre-trained transformer model (package offered by Hugging Face). But to apply the pre-trained BERT, we *must* use the tokenizer provided by the model. This is not only because the model has a specific, fixed vocabulary but also because the BERT tokenizer has a way of handling out-of-vocabulary words. The bert-base-uncased is trained on lower-case English text and has about 110M parameters. The bert-large-uncased is also trained on the same corpus but has 340M parameters. Bio-bert on the other hand is based on bert-large-cased and is trained on the BioMedical corpus (PubMed). Now we set the maximum length of a sentence as 64 and perform tokenization. We use the tokenizer.encode_plus function to split the sentence into individual token, add [CLS] and [SEP] tokens, map these to their IDs, pad all sentences and create attention masks which explicitly differentiate real tokens from [PAD] tokens. Finally using the BertForSequenceClassification (providing the number of target labels as 2), we load our model and using the AdamW optimizer we tune our hyperparameters. Although all the above pre-trained models did outperform our baseline model, we found that the RoBERTa model (which iterates on BERT's pretraining procedure, including training the model longer, with bigger batches over more data; removing the next sentence prediction objective; training on longer sequences; and dynamically changing the masking pattern applied to the training data) outperformed all the above models.

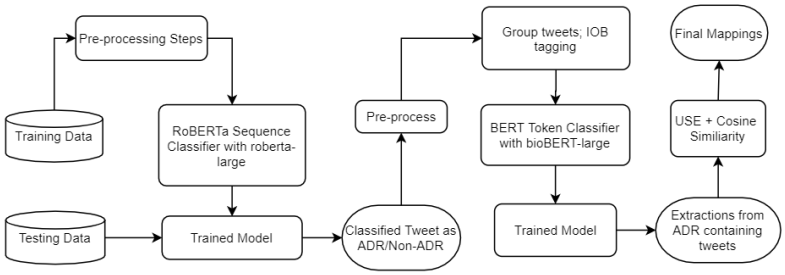
Task 2

Before arriving at the final model, we tried implementing several models including that of a residual LSTM with ELMo embeddings (embeddings from a language model trained on the 1 Billion Word Benchmark). But in the end, the BERT model outperformed the previously tried models. We briefly touch upon what was done as part of this model in this section and elaborate on the results later. This BERT implementation comes with a pretrained tokenizer and a defined vocabulary. We load the one related to the pre-trained model bio-bert since our dataset places more importance on medical terminology and jargon. Then we tokenized all sentences using the BERT tokenizer which is a Wordpiece tokenizer and splits tokens into sub-word tokens. Then we cut and padded the token and label sequences to our desired max length. Since the bert model supports attention masks, we created the mask to ignore the padded elements in the sequences and converted the dataset into torch tensors. Finally, we defined the dataloaders and shuffled the data at training time with the RandomSampler and passed them sequentially with the SequentialSampler at test time. The transformer package provides a BertForTokenClassification class for token-level predictions which is a fine-tuning model that wraps bert model and adds token-level classifier on top of the bert model. We load the pre-trained bio-bert model and provide the number of labels as 3. We also employed the AdamW optimizer, added weight decay for regularization for the main weight matrices and a scheduler to linearly reduce the learning rates through the epochs.

Task 3

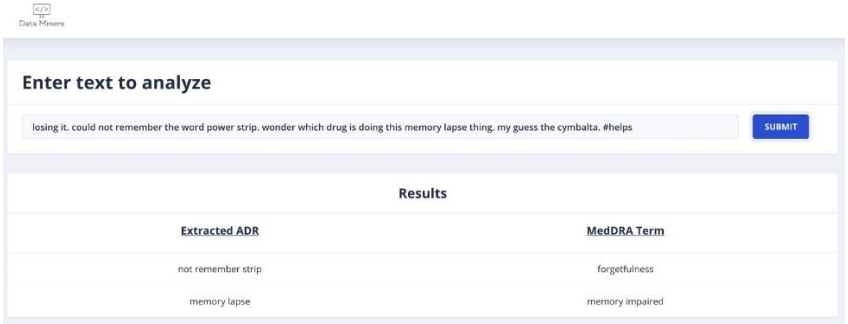
Since there were 477 individual classes and a lot of them only have a couple of examples each in the dataset, it did not make a lot of sense to us to use a classifier to map the extracted terms against a MedDRA ID. We thought that using some sort of embedding of the extracted terms and using a similarity function to find out the closest match from the MEDDRA terms would be a better approach here. Now to vectorize the extracted terms, instead of using the word embeddings like word2vec, or FastText, we used sentence embeddings because simply averaging word vectors to get a vector representation of the sentence does not take into consideration the interaction between the words in a sentence, nor the word order. To achieve this, we tried out Google's Universal Sentence encoder which is trained on a number of natural language prediction tasks that require modelling the meaning of word sequences rather than just individual words and BERT sentence encodings using BERT and used cosine similarity to find out the closest possible match.

Combining all three tasks, we designed an end-to-end system and designed a web app which supported the combined functionality. Our final end-to-end system is represented below:



To demonstrate our results, we verified if the tweets that we passed through our pipeline rendered the expected results or retrieved a partial match or was entirely incorrect. The following image depicts a correct match of the ADR identification and extraction ("emotional mess") and maps it to the correct MedDRA PT ("emotional distress"):

The following image depicts a partial match where the "strip" word was an extra extracted term and "not remember" should've mapped to memory loss rather than forgetfulness. The actual embeddings "forget-fullness" and "memory loss" are similar and hence the model is not able to distinguish between sentences close in the vector space.



The following image shows the presence of no ADRs in the tweet accurately predicted by the system:

6 Evaluation Metrics

For the evaluation metrics, task 1's classifier was primarily measured against F1-score for the successful detection of ADR/Non-ADR. For task 2 and 3, strict F1-score, relaxed F1-score, precision and recall were used in evaluating the performance of the baseline as well as the final classifier. Below we briefly explain all terminologies:

Accuracy – Accuracy is a ratio of correctly predicted observations to the total observations but it is not necessarily the sole factor of importance while evaluating any model. Accuracy is calculated as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

F1-Score – The F1 score considers both the precision and the recall of the test to compute the score: it is the harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst at 0.

$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Strict F1-Score – Strict F1-Score is the harmonic mean of the strict precision and the strict recall.
Relaxed F1-Score – Relaxed F1-Score is the harmonic mean of the relaxed precision and the relaxed recall.
Precision – Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

$$Precision = \frac{TP}{TP + FP}$$

Recall – Recall is the fraction of the total amount of relevant instances that were actually retrieved.

$$Recall = \frac{TP}{TP + FN}$$

7 Results

Below we present with our results for all the models tried for each of the tasks. The blue row indicates the ideal baseline model while the green row indicates the best model for the task.

Task 1: We evaluated the models implemented for this task primarily using their F1 Scores. We have also listed out the models tried prior to the baseline model. As evident, the RoBERTa model outperformed all others by achieving an F1 Score of 0.65.

Model	F-1 Score
MNB with Tf-Idf vectorizer	0.28
MNB without sample weights	0.33

RNN	0.35
MNB with CountVectorizer	**0.48**
BERT	0.59
BERT (bert-base-uncased)	0.60
BERT (bert-large-uncased)	0.63
BERT (biobert-large)	0.63
RoBERTa (roberta-large)	**0.65**

Task 2: Here we analyze both the relaxed F1 Score and the strict F1 score to judge our model. As we can see the BERT trained on biobert-large outperformed the other models with a relaxed F1 Score of 0.7 and a strict F1 Score of 0.6.

Model	**Relaxed F-1 Score**	**Strict F-1 Score**
CRF	-	0.33
spaCy (without grouping)	-	0.317
spaCy (residual CNNs)	**0.417**	**0.36**
Residual LSTM with ELMo embeddings	0.52	0.45
BERT (biobert-large)	**0.70**	**0.60**

Task 3: Here we analyze both the F1 Score and the accuracy to judge our model. As we can see the USE+Cosine Similarity model trained outperformed the other models with an F1 Score of 0.48 and an accuracy of 0.49.

Model	**F-1 Score**	**Accuracy**
Rocchio's	0.33	0.375
Logistic Regression (LR)	0.35	0.367
LR + Word2Vec	0.37	0.361
SVM	**0.38**	**0.375**
BERT + CS	0.45	0.48
USE + CS	**0.48**	**0.49**

8 Work Distribution

Task	Distribution of work - Ehtesham	Distribution of work - Priya
Task 1	Implemented the baseline MNB classifier. Preprocessed the data for this task.	Implemented the final models using BERT and its various pretrained models.
Task 2	Implemented BERT, BiLSTM with Elmo embeddings.	Implemented spaCy baseline model and preprocessed the data for Task-2.
Task 3	Implemented the task using Universal Sentence Encoder and BERT sentence embeddings.	Implemented the baseline model using SVM, and other previously tried out models.
Task 4	Built an end to end RESTful API to fetch the data which will be bound to the website.	Designed the UI of the website using Angular.

Conclusion

To conclude, although we did achieve results that approximated the state-of-the-art solutions of last year's SMM4H Shared Task competition, those results were obtained post the publication of the testing corpus. So as of now, since our results are based off the validation dataset, the above-mentioned statistics are not an final indicative measure of how well the ensemble system would perform. We also intend to research on more approaches to further fine-tune our models and achieve a higher performance in terms of F1-Scores as well as to work with a larger diversification of tweets.

References

[1]https://huggingface.co/transformers/pretrained_models.html

[2]https://github.com/google-research/

[3]https://www.depends-on-the-definition.com/named-entity-recognition-with-bert/

[4]https://mccormickml.com/2019/07/22/BERT-fine-tuning/

[5]https://www.aclweb.org/anthology/W19-32.pdf

[6]https://www.depends-on-the-definition.com/named-entity-recognition-with-residual-lstm-and-elmo/

[7]https://github.com/nlptown/nlp-notebooks/blob/master/Data%20exploration%20with%20sentence%20similarity.ipynb

[8]https://github.com/naver/biobert-pretrained

[9]https://github.com/nlptown/nlp-notebooks

[10]https://github.com/adsieg/text_similarity

[11]https://www.aclweb.org/anthology/W19-3203.pdf

[12]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5605929/