Chapter

Categorizing Patient Disease into ICD-10 with Deep Learning for Semantic Text Classification

Junmei Zhong and Xiu Yi

Abstract

How to leverage insights into big electronic health records (EHRs) becomes increasingly important for accomplishing precision medicine to improve the quality of human healthcare. When analyzing big Chinese EHRs, there are a lot of applications that we need to categorize patients' diseases according to the medical coding standard. In this paper, we develop natural language processing (NLP), deep learning, and machine learning algorithms to automatically categorize each patient's individual diseases into the ICD-10 standard. Experimental results show that the convolutional neural network (CNN) algorithm outperforms the recurrent neural network (RNN)-based long short-term memory (LSTM) and gated recurrent unit (GRU) algorithms, and it generates much better results than the support vector machine (SVM), one of the most popular conventional machine learning algorithms, demonstrating the great impact of deep learning on medical big data analysis.

Keywords: electronic health record, natural language processing, deep learning, convolutional neural networks, long short-term memory, gated recurrent unit, SVM, ICD-10

1. Introduction

It has been found out that the analysis of the big EHR can help accomplish precision medicine for patients to improve the quality of human healthcare. In EHR analysis, there are a lot of applications that need to categorize each patient's disease into the corresponding category with respect to a medical coding standard. The automatic categorization is very desirable for the massive EHR data sets. In this paper, we develop deep learning algorithms for semantic text classification of EHR. Our categorization system consists of four core components to accomplish the disease categorization. Firstly, by using domain knowledge of medical informatics and techniques of information fusion, we construct structured & meaningful patients' clinic profiles from the scattered and heterogeneous medical records such as inpatient and outpatient records, lab tests, treatment plans, and doctors' prescriptions for medications in the EHRs. This makes it possible for us to leverage the insights from the big data with artificial intelligence (AI) algorithms. Secondly, we extract each patient's historical disease descriptions in the clinic profiles and take each of them as a document for categorization. Thirdly, we use NLP algorithms for document tokenization and vector representation when necessary. The last component is to train the predictive model based on supervised classification to

categorize each disease into one of the 26 categories according to the first-level disease categories of the 10th version of the International Classification of Diseases (ICD-10) and Related Health Problems to standardize medical records [1] through text data categorization using deep learning algorithms such as the CNN, LSTM, and GRU neural networks. Experiments of comprehensive studies show that the CNN algorithm outperforms the other deep learning algorithms, and it generates much better results than the traditional machine learning algorithms for the same data set according to the quantitative metric of F1-score.

Our contributions are demonstrated in the following two aspects:

- We construct the patients' individual clinic profiles from the scattered and heterogeneous clinic records and tables in real EHRs with our medical domain knowledge together with medical informatics for medical information processing. The constructed clinic profiles make it feasible for us to generate actionable intelligence from the unstructured EHR raw data sets using machine learning, NLP and artificial intelligence algorithms (AI).

- We design and train predictive models with NLP, embedded representations, and deep neural network algorithms to categorize patients' diseases into ICD-10 standard.

The remaining of this paper is organized as follows. The research methodology is discussed in detail in Section 2. The experimental results are presented in Section 3, and the conclusions of the paper are made in Section 4 together with some discussions and the direction of future work.

2. Research methodology

This research consists of 4 components for the categorization of EHRs: problem definition and data preparation and collection from EHR, text data extraction from the prepared and collected data, the tokenization of the Chinese documents using NLP, and supervised deep learning algorithms with embedded vector representations for tokens/words as inputs to the neural network architectures for the semantic categorization of each patient's disease symptom description into ICD-10 standard.

2.1 Problem definition and data collection

For the research of Chinese medical healthcare data analysis, we have obtained the Chinese EHRs from 10 Chinese hospitals in Shandong Province, China. These EHRs contain 179 scattered and heterogeneous clinic records and tables, for example, the patients' admission records, outpatient records, inpatient hospitalization records, all kinds of lab tests, prescriptions for medication information, surgery information, hospital and doctor information, patients' personal information and their family information. However, the data quality is not satisfactory because it is still at the early stage for the Chinese hospitals to create the EHRs for patients, and most of the doctors are more willing to write notes on their patients' record books rather than to type their notes in the computer systems. As a result, in the 179 clinic tables, only a portion of them contains useful information and there are too many non-filled columns in many tables. After the preprocessing process, only 85 tables are selected and even in these 85 tables, many records do not contain useful information and we need to do additional processing such as data governance [2] for some applications. We then construct individual clinic profiles for patients by using the

method of database information fusion on the scattered and heterogeneous 85 tables according to the medical domain knowledge. The constructed clinic profiles contain each patient's individual symptom descriptions, diagnosis records, lab tests, doctor's treatment plans, and prescriptions for each office visit or each day's hospitalization, forming the clinic profiles. The clinic profile is very useful for all applications about the EHR analysis with machine learning. But for this categorization task, we only extract the disease symptom description of individual patients from the constructed clinic profiles, and then we use NLP, machine learning, and AI algorithms to analyze them. The number of valid records with non-empty symptom description in the EHR is significantly reduced after all preprocessing steps.

2.2 The tokenization of Chinese documents

When we use machine learning for document categorization, documents first need to be tokenized into individual words or tokens. For traditional machine learning algorithms, documents are represented as feature vectors with the bag of words (BOW) feature engineering method. For deep learning algorithms, each document is the input of the deep learning architecture with individual tokens represented as word embeddings. So, we first tokenize each Chinese document into a collection of terms. The tokenization of Chinese documents is very different from that of English, which can be accomplished through the delimiters between terms. In this work, we use the Han LP, an open source tokenization tool for tokenizing Chinese documents. After some preprocessing for the tokens, we then represent each word with the pre-trained embedded vector representation with the language model of distributed representation learning algorithm word2Vec [3].

2.3 Embedded vector representation for words

Both traditional one-hot vector representation and BOW-based TF-IDF or binary feature representation for words have a lot of limitations for document classification. They are constrained with problems of high dimensionality and sparsity for the feature vectors and they are not able to capture any semantic information from the words due to the mutual orthogonality of the individual words with such representations. The motivation of using the distributed embedded vector representation is to reduce the dimensionality of the vector while at the same time providing semantic information of words. The word2Vec algorithm [3] is a kind of distributed representation learning algorithm for the language model to capture each word's semantic information through the embedded dense vectors so that semantically similar words can be inferred from each other. There are two different models in the word2Vec algorithm and they are the continuous bag-of-words (CBOW) model and the skip-gram model as shown in **Figures 1** and **2**, respectively.

For a sentence, the CBOW model is used to predict the current word from its left side and right side context words, which are within a window centered at the current word. On the other hand, the skip-gram model is used to predict the surrounding context words in a sentence for the given current word with the context words within a window whose center is at the current word. The Word2vec model can be trained in two different ways: using the hierarchical softmax algorithm or using the negative sampling method. For the hierarchical softmax algorithm, we first get the vocabulary of the corpus and then we create a binary Huffman tree for all words according to their frequencies of occurrences, and all words are the leaf nodes of the Huffman tree. The main benefit of the Huffman tree is that it offers the convenient access of the frequent information. In the Huffman tree, the high-frequency words have short paths from the root of the tree to the individual leaf nodes, and the low-frequency

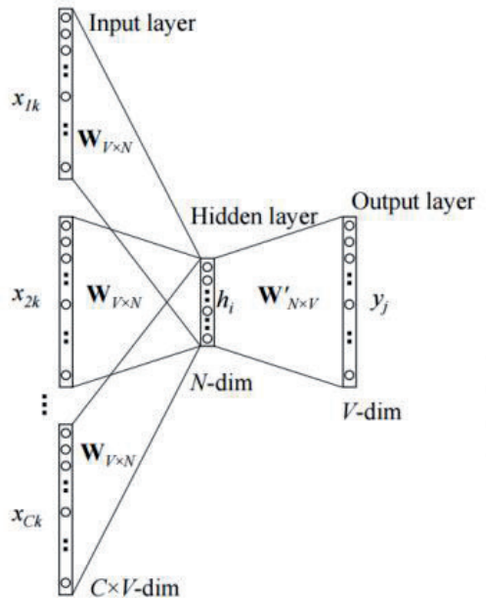


Figure 1.
The CBOW model in word2Vec.

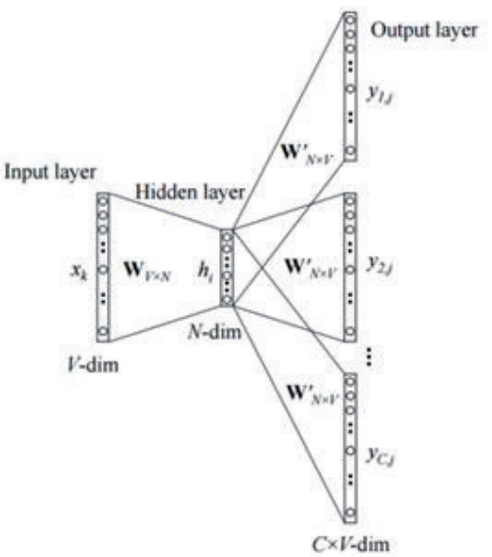


Figure 2.
The skip-gram model in word2Vec.

words have long paths, thus accomplishing optimal encoding performance from the viewpoint of bit-rate. In the original softmax algorithm, as shown in formula (1), when updating one word's vector during the training process, all words' vectors should be involved in the calculation and the computational complexity is accordingly as high as O(N). On the other hand, in the hierarchical softmax, when updating the embedded vector for a leaf node in the tree, only the leaf node and non-leaf-nodes on the path from the root of the tree to the leaf node need to be involved in the calculation. As a result, by using the Huffman tree based hierarchical softmax algorithm, we can significantly reduce the computational complexity from O(N) to O(logN).

$$softmax(x, w) = \frac{e^{x^T.w}}{\sum_j e^{x_j^T.w}} \tag{1}$$

As for the negative sampling, it does not need all words' vectors to be involved in updating a word's vector in the back-propagation process and it only needs to sample

a few negative pairs like (central word, [negative context words]) for the central word of each positive pair (central word, positive context word) in the corpus. Furthermore, to improve the vector quality of low-frequency words, the high-frequency words are down-sampled and the low-frequency words are up-sampled by using a method called frequency lifting. This helps because otherwise the high-frequency words are more likely to be sampled than the low-frequency words for updating their vectors. With the embedded representation from Word2Vec, each word in the corpus can be represented by a unique low-dimensional dense vector. **Figures 3** and **4** demonstrate the clustering feature of this kind of word embeddings that if some words are semantically close to each other, their representations in the vector space are close to each other.

2.4 Deep learning for categorizing diseases into ICD-10

In this work, we build the predictive models based on deep learning techniques with the supervised learning methodology. To this end, we train four different deep learning models for performance comparison: the CNN with max pooling and k-max pooling, respectively, LSTM, and GRU.

2.4.1 The CNN model

The CNN is one of the popular deep neural network algorithms for both NLP and computer vision applications. The CNN architecture contains the layers of automatic feature extraction, feature selection, and the pattern classification. It has attracted great attention in text classification [4] and computer vision [5]. As demonstrated in **Figure 5**, different channels of the data can be used as inputs of the CNN architecture for feature extraction and feature selection via convolutional operations together with the pooling process and nonlinear activation. For computer vision applications, the R, G, B colors of an image are usually used as CNN's inputs. For text classification applications, a matrix of word embeddings stacked by the words' embedded vectors according to the order of the words in the sequence, is usually used as the input of the CNN architecture. The embeddings can be either from word2Vec, one-hot representation and/or other vector representations of words, forming different channels of the representation of text data. Within the CNN architecture, each channel of the texts is represented as a matrix, in which, the rows of the matrix represent the sequence of words according to their order, and each row is a word's vector representation with the number of columns being the

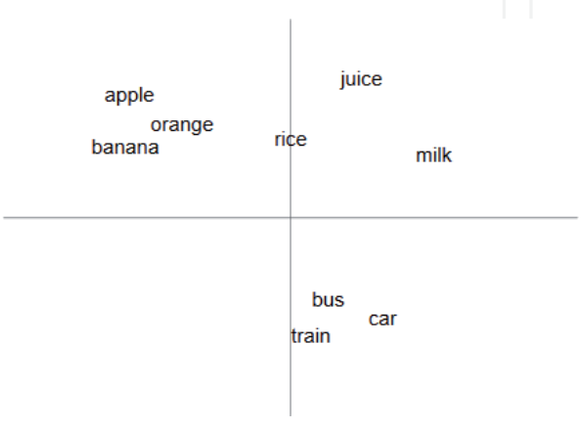


Figure 3.
The clustering characteristics of embedded vector representations of semantically similar words in the dense vector space.

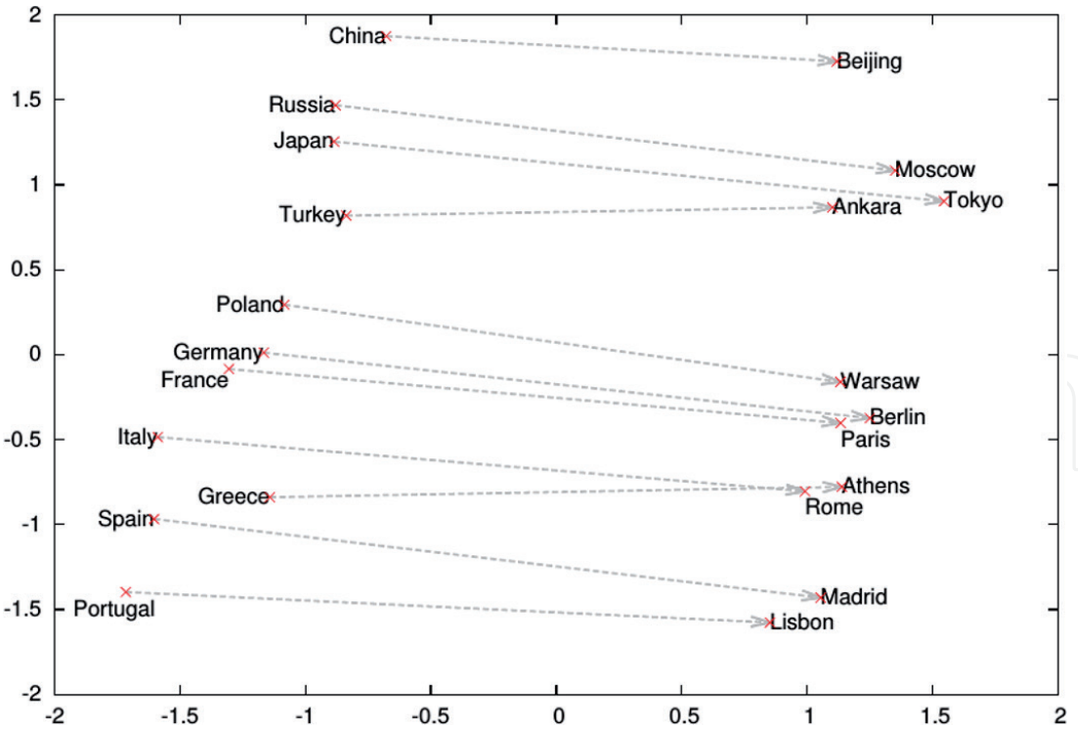


Figure 4.
The similar sub-structures of the embedded vectors in the dense vector space between semantically meaningful country-capital word pairs with the courtesy of Mikolov etc. [3].

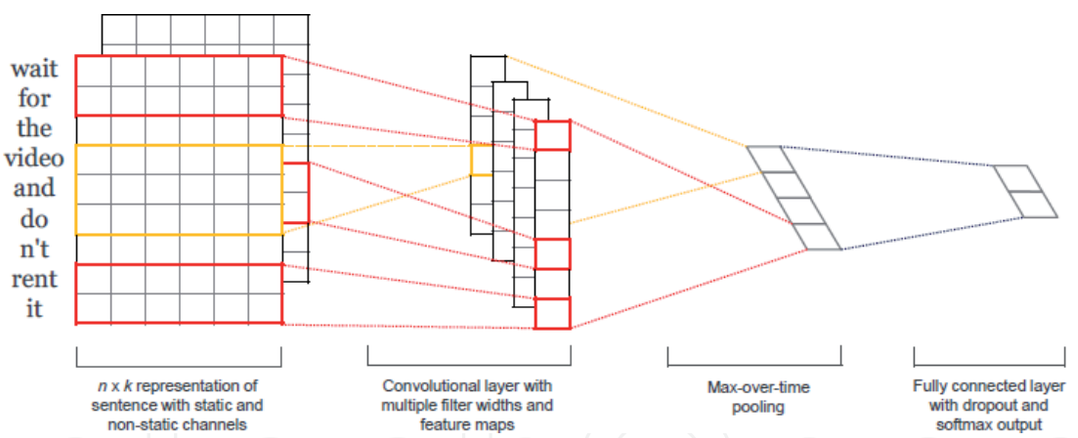


Figure 5.
Text classification using the CNN architecture with courtesy of Kim [4].

dimension size of the embedded vector space. For feature extraction, the matrix is convolved with some filters of different sizes, respectively. All filters have the same number of columns as the words' embeddings.

For text classification, the main idea of CNN using different sizes of filters is pretty much like that for computer vision. It is for the purpose of extracting the semantic features (patterns) of different N-Grams characterized by the filters. The different filter sizes correspond to different N-Grams. The words' vector representations can be either from the pre-trained word embeddings, for example, the word2Vec embeddings, or randomly initialized. For the randomly initialized words vectors, they are iteratively updated in the back-propagation process during the training stage until the training process is done, and this is another way of representation learning. For the clear illustration, we assume the filter w to have m rows, the maximum length of each sentence x is n, the dimensionality of word embeddings is d, then the sentence matrix and filter w can me mathematically formulated

as $x \in R\wedge(n \times d)$ and the filter $w \in R\wedge(m \times d)$. CNN usually uses multiple such filters with different sizes m, for example, it uses 128 filters for each size m. During the convolution process, each of the filters gradually moves down by one word each time along the sequence of words. At each position, the filter covers a few (m) rows of the words' vector matrix, and the element-wise multiplication of the filter with the covered matrix is taken, and the multiplication results are summed up. This sum is then fed into the nonlinear rectified linear unit (Relu) activation function with an added biased term $b \in R$. The squeezed value is generated as a feature value, mathematically represented in the following formula:

$$f\_i = activation(w \cdot x[i{:}i + m - 1] + b) \tag{2}$$

where *activation* denotes the activation function such as Relu(), the dot operation between matrix w and x denotes the element-wise multiplication operation, and the subscript index i is the position index of the filter. After the convolution process is done for one filter for a sentence, a list of feature values is obtained like $feature\_map = [f\_1, f\_2, f\_3, \ldots, f\_(l - m + 1)]$. This is called a feature map corresponding to the filter. Finally, the pooling operation continues to take the most important feature value or a few most important feature values from the feature map as the output of the filter's convolution result with the sentence. When all filters are applied for convolution with the sentence's matrix, we obtain a feature vector for the input sentence. This feature extraction process and selection process makes the length of the final feature vector only dependent on the number of filters used for convolutions. The final step in the CNN architecture is a full connection layer including the dropout strategy and regularization, from the final feature vector to the output layer. The last layer of the CNN architecture is a full connection layer from the feature vector to the output. We finally obtain the classification result of a sample by applying the softmax function to the output for either binary or multi-class classification. For multi-class classification, the number of neurons of the output is equal to the number of classes to be predicted.

In addition to the commonly used max-pooling for CNN, another pooling method is the k-max pooling CNN [6], in which for each feature map obtained from the convolution of a filter w with the sequence matrix x, instead of selecting a single maximum value, it selects K consecutive maximum values. The benefit of using the k-max pooling strategy is to get some distribution information about the features in addition to the features themself in the feature map.

2.4.2 The RNN model

RNNs are very efficient for modeling the time series data for prediction and forecasting tasks. As shown in **Figure 6**, the chain-link RNN architecture consists of a sequence of neural network modules sharing the same parameters. Each module is used to analyze the information at a time in the sequence. At any time, the RNN system takes as inputs the information $x(t)$ at the specific time t, and the hidden state $h(t-1)$ of the previous module $c(t-1)$.

The typical NLP applications of using RNN are text classification, sentence generation, and language translation. With the sequence modeling, a sentence is taken to be an ordered time series with the starting time 0 at the first word and at time t, the corresponding word in the sentence has dependencies on its left-side words and its right-side words, which are considered as the contexts demonstrating the underlying syntactic and semantic information of the word in the sentence. For text classification with RNN, a sentence is usually encoded into a single fixed-length feature vector for classification, while for sentence generation and language

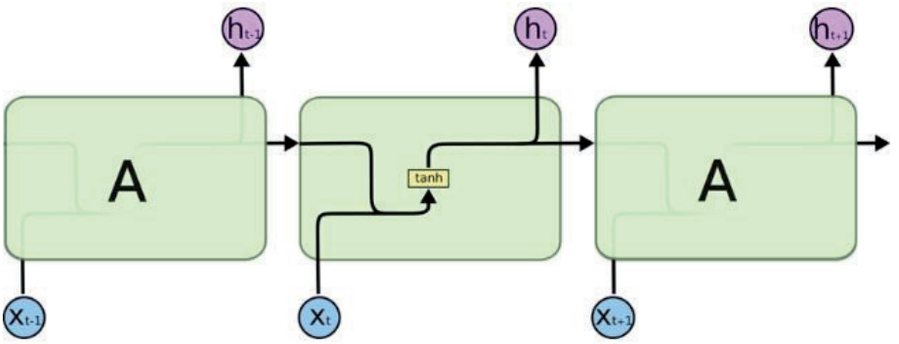


Figure 6.
The RNN structure with courtesy of Colah [7].

translation, the typical tasks are about predicting the next word given the context words seen so far together with what are generated or translated. The RNNs are great for prediction tasks that only need short contexts, but they no longer work for scenarios in which the long-term dependencies need to be remembered due to the intrinsic vanishing gradient problem of RNN during the back-propagation process. As a result, the long-term dependencies cannot be used for learning purposes and it is also very slow for the RNNs to be convergent. A significant tweaked version of RNNs, the long short-term memory (LSTM) networks [8], or the gated recurrent units (GRU) [9] is proposed to tackle the challenge of vanishing gradient.

2.4.3 The LSTM architecture

The LSTM networks are a special kind of RNNs, capable of learning the long-term dependencies in the input data. LSTMs are explicitly designed to avoid the vanishing gradient problem in learning the long-term dependencies in the data and they have achieved great success in sentiment analysis, text classification, and language translation. As shown in **Figure 7**, the LSTM architecture is very similar to that of the RNN. It is a chain of repeating modules, each of which is a modified version of that in RNNs.

By comparing a module at time *t* in **Figures 6** and **7**, respectively, we can clearly observe that the LSTM architecture has some additional components in each module and these components are called gates with different functionalities to work together so that the long-term dependencies of words in the input sequence can be used for learning and the gradient vanishing problem can be solved. For NLP, both syntactic and semantic information can be encoded by LSTM networks for prediction and classification purposes. The first gate is the forget gate, and it is

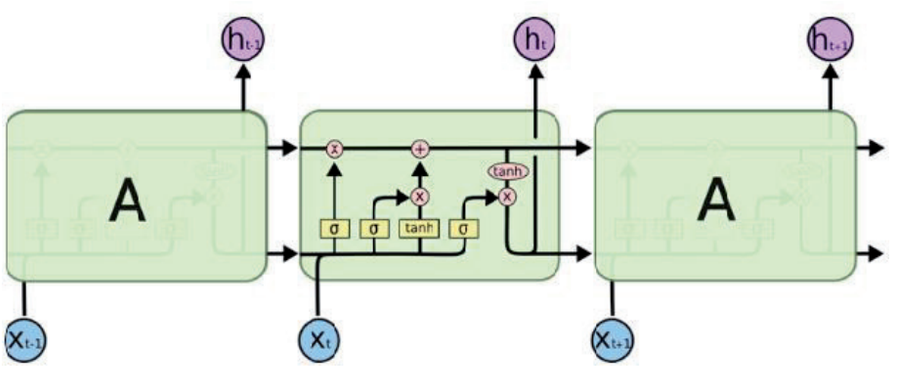
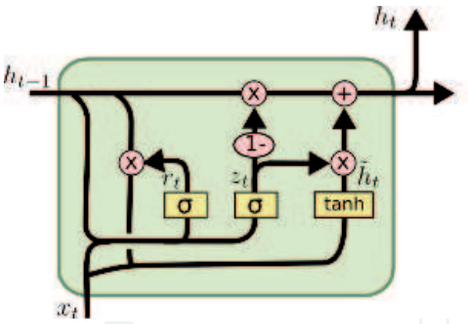


Figure 7.
The LSTM architecture with courtesy of Colah [7].



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 8.
The GRU structure with courtesy of Colah [7].

used to prevent some input information from entering into the module through a sigmoid layer to process $h(t-1)$ and $x(t)$. The output of the forget layer is a vector of values between 0 and 1, indicating how much the corresponding element in the old cell state $C(t-1)$ can be reserved to be the current cell state. If a value is 1, the corresponding information in the old state cell $C(t-1)$ is completely kept. On the other hand, if the value is 0, the corresponding element in the old state cell $C(t-1)$ is forbidden, and if the value is between 0 and 1, then only a portion of the corresponding element in the old state cell $C(t-1)$ is kept. The second gate is the input gate including a sigmoid layer and a tanh layer. The sigmoid layer is used to determine how much information in the inputs can be added to the current cell state $C(t)$, and the tanh layer creates a new vector of values to determine the polarity and proportion for the output of the sigmoid layer to be added to the current cell state $C(t)$. The elementwise product of the output of the sigmoid layer and the tanh layer is added to the current cell state $C(t)$. The third gate is the output gate which includes the filtered cell state $C(t)$ and a sigmoid layer. The output of the sigmoid layer with inputs of $x(t)$ and $h(t-1)$ is used to determine how much we are going to output from the inputs. The filtered cell state through a tanh activation function determines the polarity and proportion for each element in the current cell state $C(t)$ for updating the result of the sigmoid layer. Then the result of the filtered cell state is multiplied with the result of the sigmoid layer in an elementwise way to get the output of the current module as well as the hidden state $h(t)$ for the next module at time $t+1$. Also the current cell state $C(t)$ is transferred to the next module too.

2.4.4 The GRU architecture

The architecture of gated recurrent units (GRU) [9] is a simplified version of the LSTM architecture with only two gates, a reset gate r and an update gate z. As shown in **Figure 8**, the reset gate determines how to combine the input $x(t)$ with the previous memory $h(t-1)$, and the update gate defines how much of the previous memory can be kept. If the reset gate is set to be all 1's, and update gate is set to be all 0's, the GRU is reduced to be the original RNN model. Compared with LSTM, GRU has fewer parameters and thus may train the deep learning model in a faster way.

3. Experimental results and analysis

The ICD-10 coding system is essentially a tree-like hierarchical structure with 3 layers to encode patients' diseases. For a larger layer number in the tree, it categorizes diseases into finer disease categories. In the first layer, it only classifies disease symptoms into 26 coarse disease categories, but, in the second layer, it can classify disease symptoms into more than 500 disease categories, and in the third

layer, it can accomplish the categorization for about 21,000 diseases. But for the time being, for supervised machine learning, we can only train a model to classify patient diseases into one of the 26 categories corresponding to the first layer of ICD-10 coding system. This is for the fact that as the number of classes increases for supervised machine learning, the required annotated training data increases significantly, however we do not have so many patient records now. Furthermore, our current EHR data is very unbalanced and most of the patients' diseases belong to popular diseases. For the 26 disease categories in the first-layer of ICD-10, our EHR system only has sufficient disease examples for 14 popular disease categories, but we do not have sufficient disease examples for the rest 12 unpopular disease categories. In this paper, we can only annotate disease examples for the 14 popular disease categories to train a 14-class classifier.

In this paper, we use the quantitative metrics to measure the performance of the models. They are the precision, recall, and F1-score, and they are calculated in the following way:

$$Precision = tp/(tp + fp) \tag{3}$$

$$Recall = tp/(tp + fn) \tag{4}$$

$$F1\_score = 2 \cdot (precision \cdot recall)/(precision + recall) \tag{5}$$

where tp denotes true positives, fp denotes false positives, and fn denotes false negatives.

For the fair performance comparison of deep learning algorithms with traditional machine learning algorithms, we compare the deep learning algorithms with what we have done with SVM [10]. For SVM, we use two kinds of vector representations for document representation: the BOW vectors and the embedded vectors. The BOW vectors can be further separated into TF-IDF weighted vectors and binary vectors. The embedded vectors can be separated into two forms: the averaged word embeddings of the pre-trained word embeddings from word2Vec to represent a document, and the doc2Vec vectors from the PV-DM model [11], respectively. As a result, we totally use 4 document representations as the inputs for SVM. For BOW method using both the TF-IDF weighted and binary vector representations, we use the keywords as the features instead of the individual tokenized words, and the dimensionality of the feature vectors is 53,039 after performing some preprocessing to filter out the infrequent keywords. When optimizing the word2Vec to get the pre-trained word embeddings, we have tried 4 models and they are the Skip-gram with the hierarchical softmax, Skip-gram with the negative sampling, CBOW with the hierarchical softmax, and CBOW with the negative sampling. We select the CBOW with the negative sampling to get the pre-trained word embeddings, and the feature vector of the SVM classifier is obtained as discussed above. The values for the optimized hyperparameters are: the number of epochs is set 10, minibatch is 32, dimension size of embeddings is 100, low frequency threshold for sampling is 1e−5, window size is 3, and 5 samples are used for negative sampling,

To train the multi-class SVM classifier, we use the grid search with the regularity $\boldsymbol{L1}$ and $\boldsymbol{L2}$, respectively, to find the optimal hyperparameters. For both $\boldsymbol{L1}$ and $\boldsymbol{L2}$, we use the hunger function to measure the empirical learning risk. The classification results of the 14-class SVM classifier are displayed in **Tables 1** and **2**, with respect to the regularity of $\boldsymbol{L1}$, and $\boldsymbol{L2}$, respectively. From the classification results of SVM, we can see that there is almost no difference between the regularity $\boldsymbol{L1}$, and $\boldsymbol{L2}$.

Input vector(s)	F1-score	Precision	Recall
Binary BOW	0.76	0.76	0.76
TF-IDF BOW	0.76	0.76	0.76
Avg. Word2Vec	0.70	0.71	0.71
Doc2Vec	0.64	0.65	0.65

Table 1.
The prediction results of 14-class SVM classifier with L1 regularity.

Input vector(s)	F1-score	Precision	Recall
Binary BOW	0.76	0.76	0.76
TF-IDF BOW	0.77	0.77	0.77
Avg. Word2Vec	0.70	0.71	0.71
Doc2Vec	0.64	0.65	0.65

Table 2.
The prediction results of 14-class SVM classifier with L2 regularity.

To train the multi-class CNN classifier, we use the pretrained word embeddings of word2Vec as the inputs of words' embeddings for the CNN architecture. For each filter size, we use 100 convolutional filters to extract features. The crawled Chinese medical documents are used as the corpus for training the word2Vec algorithm to get the pre-trained word embeddings. The optimized hyperparameter values of the CNN are: epoch size is 40, batch size is 20, dropout probability is 0.5, the $L2$-norm is used for regularization, and the dimension size of word embeddings is 200.

To train the multi-class LSTM and GRU classifiers, to make sure the deep learning models are trained in an optimal way, we have tried different optimization methods, such as the Adam, AdaDelta, and RMSprop. We find that the RMSprop optimizer works best for our data set. We set the hyperparameters to be: batch size 32, hidden size 64, epoch size 50, the dimension of word embeddings 300, dropout 0.5, and L2 regularization lambda 0.7. The LSTM and GRU algorithms are implemented in Python and Tensorflow. Since the training process for both LSTM and GRU is very time consuming for the use of 6000 training and validation examples for each disease category, for computational efficiency, we shorten each disease description to be 700 words. This assumes that customers' disease symptoms are mainly contained in the first 700 words of each note. Otherwise the memory requirement is very huge. This assumption is of course not 100% correct, but it greatly helps speed up the training process.

In this paper, the NLP, deep learning algorithms, and the algorithm of SVM are implemented in Python, Tensorflow, and the open source software of machine learning library Sklearn. The results of CNN, LSTM and GRU algorithms are displayed in **Table 3**, from which we can see that the CNN algorithm with max-pooling works best.

Input vector(s)	F1-score	Precision	Recall
Word2vec + CNN + k-max pool	0.74	0.73	0.73
Word2vec + CNN + Max pool	0.80	0.80	0.81
Word2vec + LSTM	0.74	0.75	0.74
Word2vec + GRU	0.74	0.75	0.74

Table 3.
The prediction results of 4 deep learning algorithms.

4. Conclusions and future work

In this paper, we develop NLP and deep learning algorithms to categorize patients' diseases according to the ICD-10 coding standard. Through comparative studies, we find out that the CNN model achieves better performance than the RNN-based LSTM and GRU models. The CNN model also outperforms the popular traditional machine learning model SVM for the same data set. In the future, we are going to investigate the transfer learning and deep learning algorithms with the attention mechanism for semantic text classification. At the same time, it is very necessary for hospitals and doctors to provide high-quality medical healthcare data and the high-quality EHR data is equally important as the medical services provided to patients.

Acknowledgements

The authors would like to thank the support from the Health Planning Commission of Shandong Province, China.

Author details

Junmei Zhong[1*] and Xiu Yi[2]

1 Marchex Inc, Seattle, WA, USA

2 Delta Technology Inc, Shandong, China

*Address all correspondence to: zhong.junmei@gmail.com

References

[1] ICD-10 Homepage. Available from: http://apps.who.int/classifications/icd10/browse/2016/en

[2] Junmei Z, Xiu Y. Artificial intelligence based data governance for Chinese electronic health record analysis. International Journal of Data Mining & Knowledge Management Process (IJDKP). 2018;**8**(3):29-41

[3] Tomas M, Kai C, Greg C, Jeffrey D. Efficient estimation of word representations in vector space. Advances in Neural Information Processing Systems. 2013:3111-3119

[4] Kim Y. Convolutional neural networks for sentence classification. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. 2014. pp. 1746-1751

[5] Krizhevsky A, Sutskever I, Hinton G. ImageNet classification with deep convolutional neural networks, NIPS'1. In: Proceedings of the 25th International Conference on Neural Information Processing Systems. Vol. 1. 2012. pp. 1097-1105

[6] Nal Kalchbrenner, Edward Grefenstette, Phil Blunsom. A convolutional neural network for modeling sentence. In: Proceedings of the 52nd Annual Meeting of Association for Computational Linguistics. Vol. 1. 2014. pp. 655-665

[7] Colah's Blog. Available from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[8] Sepp H, Schmidhuber J. Long short-term memory. Neural Computation. 1997;**9**(8):1735-1780

[9] Chung J, Gulcehre C, Cho KH, Bengio Y. Empirical evaluation of gatedrecurrent neural networks on sequence modeling. arXiv:1412.3555v1

[10] Zhong J, Gao C, Yi X. Categorization of patient disease into ICD-10 with NLP and SVM for Chinese electronic health record analysis. In: Proceedings of 2018 International Conference on Artificial Intelligence and Pattern Recognition; Aug. 18-20, 2018; Beijing, China

[11] Le Q, Tomas M. Distributed representation of sentence and documents, proceedings of the international conference on. Machine Learning. 2014:1188-1196