DHRS Medical Insistence System

Jian Xu JXQ9011 | Rongqi Ding RDX3151

Purpose:

The health and medical care industry is an industry of great importance and intricacy because it is related to everyone. What we're looking forward to establish is a tool which takes patients' symptoms (typed in words) as inputs and gives basic diagnose as outputs. A tool like this can be of great use. Patients can use it to get an idea of the potential explanation to their health problems and how severe it is. Doctors and hospitals can use it to assign patients to the most suitable department and reduce the amount of human labor involved.

Dataset:

We used a medical transcription dataset from Kaggle:
https://www.kaggle.com/tboyle10/medicaltranscriptions
This data set contains around 5000 samples, covering 40 medical specialties. From the various attributes, we made use of "description" and "medical specialty". The former one is a sentence describing the symptoms the patient is having. The latter one is the professional area or department the patient's doctor belong to. We split the whole data set in training set and test set with the ration of 7:3. For comparison purpose, the two sets were fixed throughout the process.

Pipeline:

Our work can be broken down into three major subtasks: "Word Vectorization", "Feature Extraction" and "Classification". The raw input are first preprocessed into vectors using the "bag of words" idea, and then corresponding features (or key words) are extracted separately for every medical specialty (for reference simplicity, referred as 'class' in the following report). In the classification process, output value are assigned by the class whose features are what the test sample is closest to. A weighted distance calculation is used in this process.

i) Word Vectorization

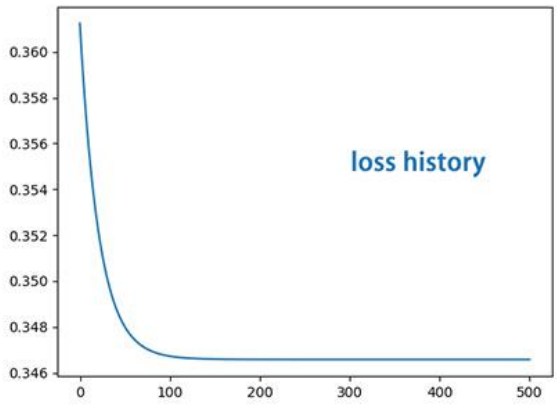
Our preprocessing procedure contains the following steps. (a) Punctuation: "-" is replaced with " " to break down the words before all the punctuations are removed. (b) Regularization: All the letters are regularized to lowercase. (c) Stop Words: Stop words are removed from the sentence. (d) Tokenization: Up to this point, the input is still in "string" form. By tokenizing it, the string is separated into individual words and make up a "list" of word tokens. (e) Lemmatization: To further generalize the applicability of this system, lemmatization is introduced. Words from the same family but with different affixes are grouped into one. For example, "allergies" is swapped for "allergy". (f) Vectorization: After all the samples are tokenized, a vector marking the number of appearance of all the non-repeat input tokens can be created for every description (Bag of words). After all these steps, the preprocessing procedure is complete. Fig. 1 is shows the progress of preprocessing after each step.

```
Raw Input:   A 23-year-old white female presents with complaint of allergies.
After Puctuation:   A 23 year old white female presents with complaint of allergies
After Regularization:   a 23 year old white female presents with complaint of allergies
After Stop Words Remeval and Vectorization:  ['23', 'year', 'old', 'white', 'female', 'presents', 'complaint', 'allergies']
After lemmatization and Deduplication:  ['allergy', 'white', 'female', 'present', 'year', 'old', 'complaint', '23']
```

Fig. 1 Step by step progress of preprocessing

ii) Feature Extraction

Our idea for feature extraction came from the two-class classification problem. Instead of the classification results, what we made use of are the most weighted input dimensions, in this case, words. We implemented a slightly twisted "One-verses-all" liner classification, trained with gradient descent. We tried out both softmax loss function and perceptron loss function. They performed pretty much the same and we ended up going with perceptron. Input samples wise, the positive samples are obviously the sample from the class of interest. The negative samples are randomly chosen from the rest of the classes for more diversity. For the purpose of a well-balanced data set, the same exact number of positive and negative samples is guaranteed. As shown in Fig.2, the training process converged nicely (we used a learning rate of 3e-2 and 500 iterations). After the training, the top 15 weighted input dimensions are extracted and saved as the feature of the corresponding class.



Loss history in one training.

fever 4.420995478203382
etiology 4.018200021784411
allergic 3.9367192364636336
however 3.2322583113659715
asthma 2.3069448746372405
diagnosis 2.153627900244556
discomfort 2.120955339161469
day 2.111148293996619
reaction 2.05982694058335
thrombocytosis 2.0264880282782363
complaint 1.915065951157577
three 1.8737132920320896
year 1.8527232792649064
severe 1.6641273340465894
arthritis 1.6603679680938925

Extracted features and their weights.

Fig. 2

iii)Classification

As we already have the training and testing data and the labels, then we can build our system. Base on the problem we are trying to solve and the form of our input data, also according to our former experiments, we decided to use KNN as our classification algorithm. So in that case there is no training required. As our training data are some txt files each contain 15 keywords and corresponding weight, first thing is to parse the training data. We create a list to save all the symptoms and their keywords, each symptom is a dictionary and the keywords in it is the index, each keywords' weight saved in their corresponding index. Then we writing our own KNN algorithm to compute the distance between our new input data and the training data. We use weighted L1 norm to compute the distance, which means if there is one keyword in the new input that match the keywords in our training data, then our distance minus the weight plus one, if the keyword is not in the training keywords then the distance minus the weight plus zero which equal to zero. We use a loop to compute all the keywords' matching and after the loop is done, we output a list contain the distance of all the distance between our new input and our training data, then we use 'argmin' to select the smallest label as the output classifications, meanwhile we also output the second smallest and the third smallest labels as candidates.

Because each training keywords in different symptoms has different weight scale, so in order to

achieve a more reasonable result, we normalize all the weights in a symptom to 1, so the result will have little influence if there is a keyword that has a much larger weight that disturb the other keywords. Even more, we averaged each distance so we won't have a super large distance if the input sentence has a lot of keywords.

Analysis:

Next, after we tested our code and make sure it can achieve what we want, we use our model to test on our test dataset.

```
resolving 9.80963709777853
eye 4.333747658110337
allergic 4.24946477324361
complaint 3.559393464346946
diagnosis 3.51241445155196
inhalant 3.211182866014142
physical 2.8969212500445476
pain 2.6335566876652883
uncertain 2.433953822569765
5 2.212876519066899
wheezing 2.1593933492921944
nasal 2.130978485460863
came 1.8308820636332817
keflex 1.8081951240915002
discomfort 1.7882976835560689
```

Fig. 3 keywords & weights of Allergy Immunology

According to the test result, we can achieve very high accuracy in some symptoms, like Allergy Immunology(Figure 3) and Autopsy can achieve 100% accuracy and SleepMedicine(Figure 4) and Rheumatology have 60%.

```
sleep 11.323241491156255
chronic 4.043242908362234
normal 3.5475283020318593
thoracoabdominal 3.3226481826440613
female 2.6070017854040426
obtained 2.579722447080113
performed 2.407395814314364
independent 1.8381505521428285
43yearold 1.6186317535950174
history 1.4270226672324953
polyarteritis 1.3801063890031156
demonstrated 1.2704312844970584
event 1.1673940507420504
ass 1.156807208560693
somewhat 1.12413623190278
```

Fig. 4 keywords & weights of SleepMedicine

But there do exist some symptoms that the accuracy is super low, like the accuracy of Surgery(Figure 5) and Neurosurgery are nearly zero.

```
supple 15.373424522010612
subtrochanteric 1.7495249331517246
precaval 1.403857933335003
cystopyelogram 1.3515115527324146
orbitozygomatic 1.2916718830266745
medication 1.2902102266749997
pedunculated 1.2088499100587027
posterolateral 1.1760389493243644
osteomyelitis 1.1686846937151492
gag 1.1532533156894005
performed 1.148618193370724
ablation 1.1452050830125735
jp 1.1416796567100276
interbody 1.102482530324378
radiology 1.1015797381596943
```

Fig. 5 keywords & weights of Surgery

We believe that this is due to the inconsistency of the training and testing dataset. The keywords we

extracted from the training data is somehow a lot different from the keywords we extracted from the testing data, so when we run the test dataset on the model we trained based on training dataset, the model cannot correctly detect the similarity, then the accuracy becomes a lot lower. So the quality of the keywords we extracted from the data is really important to our system.

Final step, we tried our model on some simple sentences that we created to test the robustness of our system. See figure 6.

```
Please enter your symptom(use' 'between words):My eyes is allergic to pollen and I feel pain
Your symptom is: ['allergic', 'pollen', 'feel', 'pain', 'eye']
Your first recommendation is: Allergy_Immunology -0.04618720347057147
Your second recommendation is: OfficeNotes -0.020676724280392293
Your third recommendation is: Hospice-PalliativeCare -0.011870084747189527
```

Fig. 6 A good example and the result

The result is reasonable for some sentences, but there also have some sentences that output the less related labels. See figure 7.

```
Please enter your symptom(use' 'between words):My legs are broken
Your symptom is: ['leg', 'broken']
Your first recommendation is: Dermatology -0.029409025587711447
```

Fig. 7 A bad example and the result

We think that is due to the dataset we use is limited so the keyword we extracted from it is also limited, which mean the keywords we have may not perfectly represent the symptom we want, there exist some noisy keywords like '2', 'year' etc. So when the keywords we used to test are not that representative, the result may not that reasonable as well.

Future work:

As our dataset isn't that large and the consistency is also not so good, we would like to find a better dataset and maybe using LSTM and RNN to extract keywords is a better choice. If the system is greatly done, this system can apply to a lot of daily-life scenarios.

Teamwork:

Jian Xu: worked on feature extraction and website design

Rongqi Ding: worked on classification algorithm design