Classification of Ailments Given Description of Symptoms

Urmzd Mukhammadnaim
Ben MacDonald

December 11, 2021

Contents

1 Introduction **1**

2 Related Works **3**

3 Methodology **5**

3.1 Data Collection 5

3.2 Data Processing 6

3.2.1 One Hot Encoding and Witten Bell Generation 6

3.2.2 FastText Model 6

3.3 Training & Achitecture 7

3.3.1 Layers 7

3.4 First Implementation 9

3.4.1 Results 9

3.4.2 Discussion 10

3.5 Second Implementation 11

3.5.1 Results 11

3.5.2 Discussion 12

4 Conclusion **14**

References **15**

Abstract

In this paper, we address the challenges experienced in the preliminary research phase of ailment diagnosis performed by many individuals prior to visiting a healthcare professional. Due to the large quantity of varying results appearing once someone searches their current symptoms, we developed a Convolutional Neural Network (CNN) that reduces this clutter by returning only the most probable medical condition given the user's description. To address the problem, we introduce two CNN implementations with different preprocessors. The first implementation explores the use of a One Hot Encoder, while the second implementation utilizes a FastText model trained via unsupervised learning. Through the retrieval of open source data from various medical platforms such as UpToDate and Mayo Clinic, a recall value of 90% is achieved.

Chapter 1

Introduction

Individuals are able to obtain data in a wide array of topics nearly instantaneously through the internet. Among these searches are descriptions of symptoms people are experiencing in an attempt to find the primary cause. According to a study conducted by Eligibility, approximately 89% of American patients search what they are experiencing on a search engine prior to visiting their doctor (Guarino, 2019).

The result of the query often results in a list of websites such as WebMD and MayoClinic. These pages will typically contain an ailment, a description, a list of potential symptoms, as well as possible treatments. In these encounters, the user may have a negative experience as a direct result of the overwhelming quantity of information potentially shown. Additionally, based on the description provided, the user may also find conflicting resources. Contradictory information is further exaggerated by broader descriptions, such as "pounding head", which can be attributed to a large number of illnesses.

This problem led us to consider how to structure a process that streamlines this preliminary research. As opposed to using a search engine which indexes its data primarily on a platform's metadata, we viewed the ideal scenario as condensing the actions to entering the text into a model that would provide the singular most probable diagnosis. To implement this functionality, we determined three necessary objectives:

- Acquire a dataset that includes the descriptions of an ailment.
- Construct a model that produces a probability distribution of all the available classifications.

- Utilize the developed model to classify symptom descriptions as an ailment.

To develop our dataset, we scraped text from websites which included descriptions of ailments and their associated symptoms. Moreover, only websites containing "semi-informal" terminology was used. This was done in order to improve the similarity between the training data and the input the model would likely recieve from a user. There were several possible approaches we could have taken to develop a model for this multi-classification problem. Among these, we found that Convolutional Neural Networks (CNNs) provided a flexible model which performed well with classifying text, and has previously been used to classify symptoms at a sentence level to a great degree of success. We were then able to observe the network's capacity to classify training and testing data, and proceeded to test its ability to categorize our own descriptions of any of the three ailments.

Chapter 2

Related Works

There are various ways in which we could have the user interact with the model. Kurup and Shetty document their creation of an conversational chatbot that utilizes Neural Networks and Decision Tree Classifiers to classify the ailment that a particular user is experiencing given their responses to questions about their symptoms. Due to a sparsity of publicly available datasets, they created a JSON file with custom patterns classified by their expected responses, and used this to train a Neural Network composed of two dense layers, as well as dropout layers to prevent overfitting. The Decision Tree Classifier was then trained with a dataset of binary values marking whether or not a symptom is associated with a particular ailment. When interacting with the Neural Network via messaging, the user could indicate that they wished to take a "prognosis quiz", where they would answer yes or no to experiencing a symptom. The maximum performance of the Neural Network reached 95% accuracy, and the Decision Tree Classifier functioned well. This implementation, however, does not provide a seamless series of interactions for the user as they converse with a chatbot and, in order to classify their symptoms, switch to a yes or no questionnaire.

Utilizing CNNs in Natural Language Processing (NLP) is growing in popularity, with several papers documenting their efficacy in text classification. Hughes et al describe their use of a Word2vec model and CNN to classify medical text at the sentence level, and compare their accuracy to other NLP techniques. Their vectorization model was trained through the use of a dataset containing 15,000 clinical research. A series of clinical articles were then each pre-categorized as one of 26 medical categories with 4000 sentences being randomly selected for each classification. These sentences were then vectorized by the Word2vec model and used to train the CNN. This setup then went on to perform with 68% accuracy, out performing other text classification techniques such as a bag-of-words and logistic regression model which possessed an accuracy of 51%. This is an effective model that displays the success of CNNs in this context and on a larger scale with 26 individual ailments. The use of word embeddings provides the neural network greater possible depths of understanding for the relationships between separate words and their associations. These benefits, however, are dependent on the large quantity of available data to train the Word2vec model.

There is an evident distinction between classifying professional medical text and casual descriptions with informal terminology. Gambhir et al studied the accuracy of a Convolutional Neural Network-Long Short Term Memory (CNN-LSTM) model in regards to monitoring social media for posts mentioning a drug name, and classifying them as presenting personal medication intake, possible medication intake and non-intake. In this setup, the CNN would take vectorized training data from Word2vec and apply a series of convolution layers and max pooling layers. The output being a series of feature maps, this would then be input into the LSTM model, which in turn feeds data to a series of fully connected layers and classifies the data as one of the three possible categories. The CNN-LSTM performed with the greatest precision in comparison to other tested models, but a lower recall than the LSTM model. The distinction between professional medical text and casual descriptions is an important one to make. There is a large difference between the two both in terms of sentence structure and diction. Professional text can contain a variety of rare terms for specific conditions which can impact a models capacity to properly classify a description should a large corpus be unavailable.

Chapter 3

Methodology

3.1 Data Collection

One of the challenges in the endeavour was finding accessible and well documented open-source records. As medical data is difficult to come across due to HIPAA, PIPEDA, and other laws that bar hospitals, and other medical organizations from distributing patient's health documents, we had to settle for medical encyclopedias and clinics that published articles intended for educating individuals. To ensure baseline consistency in the data used across the desired classifications, the descriptions were pulled from the same sites. However, between articles on the same website, the detail in which the article was written varied greatly. Some articles had more advertisements, others had more miscellaneous content embedded, and most of them differed in document structure. In other words, the data could not be accessed directly via its url and DOM selector. The data was collected using the topmost DOM element encapsulating all the desired information, resulting in documents with irrelevant content within them. As a result, a greater burden was moved onto the preprocessor to ensure clean and usable data was entering the CNN. It should also be noted that we used the automation library "pyppeteer" in lieu of a simple "GET" retrieval of the html document to prevent issues accessing the data due to Server Side Rendering (SSR). Consequently, this resulted in greater overhead during this phase.

3.2 Data Processing

After the extraction and consequent concatenation of the documents for each classification, we explored two preprocessor implementations. We first describe the use of One Hot Encoder and Witten-Bell Probability Distribution to retain morphological-level information, and generate unique sets of words that can later be fed into the CNN. Subsequently, we analyze the use of the FastText model as a means of retaining semantic-level information, and ensuring words with similar words appear closer together in the subspace.

3.2.1 One Hot Encoding and Witten Bell Generation

After the documents were tokenized, the text was piped into the transformer $T1$ which applied a series of reduction operations to remove punctuation and ultimately break the words into their stems. The result of $T1$ consisted of a set of unique stems D_n where n represents the order of which the the document was processed in. After $T1$ was applied to all documents, the union of all the sets was taken to form the vocabulary V. V was then fed into the *FreqDist* class from the *nltk.probability* package to allow the subsequent usage of the *WittenBellProbDist* class. By allocating $|V|$ bins, and using the *FreqDist* of D_n, we ended up with an instance of the *WittenBellProbDist* capable of generating ${}_VC_S$ unique sets, where S represents the cardinality of the generated set. The described method was used to prevent clustering of the same words, and ultimately prevent the model from developing an aptitude for classifing an input based on the frequency of its instances instead of the existence of an instance.

After N samples were generated via the *WittenBellProbDist* method described above, we transformed the data using an instance of the *OneHotencoder* class fit on the the vocabulary V. With this each word in D had its own unique binary vector and out-of-vocabulary (OOV) words were treated as the 0 vector with size $|V|$.

3.2.2 FastText Model

In the alternative preprocessor, we used the FastText model. Unlike the former method, no data generation was done. Instead, all documents were first tokenized by sentence using an instance of the *PunktSentenceTokenizer*.

The sentences were then appended to a singular file, which served as the corpus for the FastText model. Using an implementation of the FastText model by *gensim*, we trained an unsupervised word embeddings model capable of translating words into a R dimension vector. After the FastText model was trained, each word in the vocublary was projected onto the embedding space, resulting in vectors of cardinality R.

While each word had the same length, the sentence length varied. As a result, a transformer T_2 was created to pad the sequence given. The algorithm applied consisted of retrieving the vector that resulted of a coordinate-wise max or min. Alternating between max and min operations, we injected the sequence with the resulting vector from the operating described.

3.3 Training & Achitecture

As there existed little data, only 20% of the input was withheld for validation. In this section, we describe the reasoning for the architecture we decided upon. It should be noted that the architecture did not depend on the pre-processor. Hence, there is no difference in the architecture of the classifier between the two implementations.

3.3.1 Layers

The primary components of the CNN consisted of 2D Convolution layers, Max Pooling layers, a Flatten layer, Dense layers, Dropout layers and finally a Dense output layer. The convolution layers were used as a means of determining the most critical dimensions responsible in classfying an ailment while the max pooling layers were used to drop dimensions with little information. After the final pooling layer, the output was Flatten layer was used to reduce the resulting feature map onto a single column matrix, allowing it to be transformed in later operations. The resulting output was then passed to a series of alternating Dense and Dropout layers. The Dense layers were added to ensure that the CNN would be able to fine tune its mechanisms, likely finding small details in the activation map provided and over time, make better classifications. The Dropout layers were used to prevent overfitting as a result of the small dataset provided. Finally, the last Dense layer was used to reduce the learned values onto a probability distribution matching the cardinality of the classification size.

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	[(None, 56, 32, 1)]	0
conv2d_10 (Conv2D)	(None, 54, 30, 3)	30
max_pooling2d_10 (MaxPoolin g2D)	(None, 27, 15, 3)	0
conv2d_11 (Conv2D)	(None, 13, 7, 3)	84
max_pooling2d_11 (MaxPoolin g2D)	(None, 7, 4, 3)	0
dense_25 (Dense)	(None, 7, 4, 64)	256
dropout_12 (Dropout)	(None, 7, 4, 64)	0
dense_26 (Dense)	(None, 7, 4, 64)	4160
dropout_13 (Dropout)	(None, 7, 4, 64)	0
dense_27 (Dense)	(None, 7, 4, 64)	4160
dropout_14 (Dropout)	(None, 7, 4, 64)	0
dense_28 (Dense)	(None, 7, 4, 128)	8320
dropout_15 (Dropout)	(None, 7, 4, 128)	0
flatten_5 (Flatten)	(None, 3584)	0
dense_29 (Dense)	(None, 3)	10755

Figure 3.1: Fast Text Model Preprocessor - CNN

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 56, 4210, 1)]	0
conv2d_2 (Conv2D)	(None, 54, 4208, 3)	30
max_pooling2d_2 (MaxPooling 2D)	(None, 27, 2104, 3)	0
conv2d_3 (Conv2D)	(None, 13, 1051, 3)	84
max_pooling2d_3 (MaxPooling 2D)	(None, 7, 526, 3)	0
dense_5 (Dense)	(None, 7, 526, 64)	256
dropout_4 (Dropout)	(None, 7, 526, 64)	0
dense_6 (Dense)	(None, 7, 526, 64)	4160
dropout_5 (Dropout)	(None, 7, 526, 64)	0
dense_7 (Dense)	(None, 7, 526, 64)	4160
dropout_6 (Dropout)	(None, 7, 526, 64)	0
dense_8 (Dense)	(None, 7, 526, 128)	8320
dropout_7 (Dropout)	(None, 7, 526, 128)	0
flatten_1 (Flatten)	(None, 471296)	0
dense_9 (Dense)	(None, 3)	1413891

Figure 3.2: One Hot Encoding Preprocessor - CNN

3.4 First Implementation

3.4.1 Results

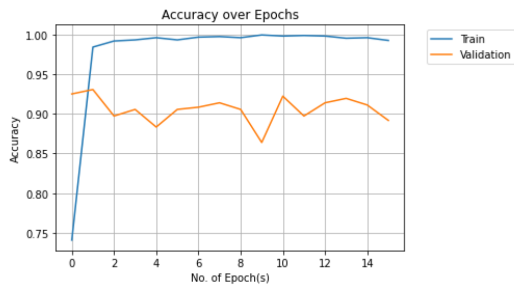


Figure 3.3: One Hot Encoding Preprocessor - CNN Accuracy

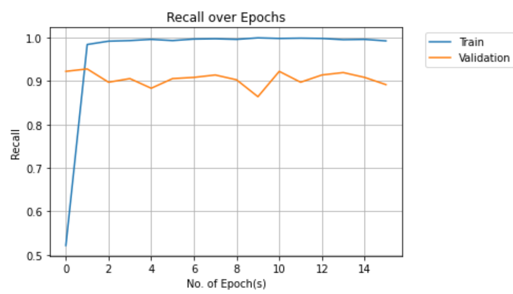


Figure 3.4: One Hot Encoding Preprocessor - CNN Recall

When observing the accuracy over the number of epochs in the first implementation, we found that at the first epoch, validation accuracy was higher than training accuracy with values of 0.925 and 0.74 respectively. For the second epoch and onwards, however, training accuracy varied around 0.99 and validation accuracy hovered around 0.9. 3.3 and 3.4, however, are measures of how well the model performed in regards to classifying the randomized training data. In 3.5, we mapped the model's capacity to classify our written descriptions as either depression, a migraine or tetanus.

As observed in Figure 3, the model was highly effective at classing descriptions of depression, migraines and tetanus. Depression possessed a precision of 0.83 and a recall of 1.0 while tetanus had a precision and recall of 1 and

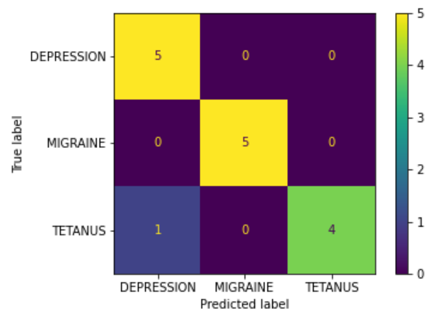


Figure 3.5: One Hot Encoding Preprocessor - CNN Confusion Matrix

0.8 respectively. For migraines, however, the model returned a precision and recall both of 1.

3.4.2 Discussion

There are several factors that play into the model's ability to label these descriptions accurately. It is evident that the neural network was capable of recognizing the particular types of words commonly used when describing the three illnesses as scraped from medical websites. An example of an accurately labelled symptom description tested was "my head feels like it's spinning, pain in head, pounding throbbing". Keywords that the model may have recognized in this specific instance were "head", "spinning", "pounding" and "throbbing". The 750 generated instances for each label used to train the model appear to have provided a generalized representation of the language used to describe symptoms to make the model perform with a high level of precision and recall.

The use of One Hot Encoding certainly impacted the run-time performance of the model. Since every single column represented a unique word, the dimensions of the sentence matrix were 56x4210. This was a result of the 4210 unique stems of words in the corpus. This large number of features slowed the model dramatically, and led to the epoch training time to increase. In an attempt to reduce dimensionality and provide the model with some form of semantic meaning, we deecided to use word embeddings in the next implementation.

3.5 Second Implementation

3.5.1 Results

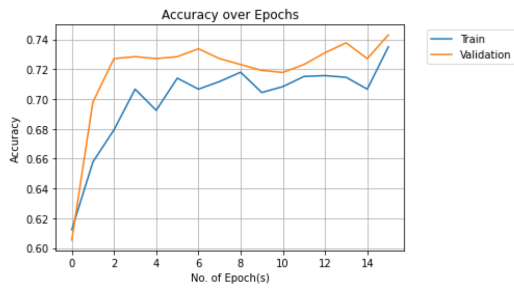


Figure 3.6: FastText Preprocessor - CNN Accuracy

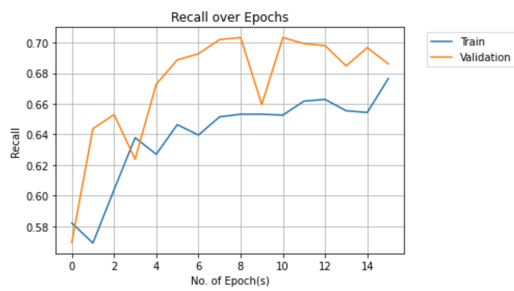
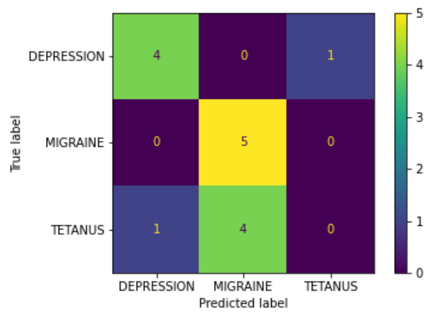


Figure 3.7: FastText Preprocessor - CNN Recall

For the second implementation, we observed that in 3.6 and 3.7, the validation accuracy and recall was greater than that of training accuracy. We believe this can be attributed to the regularization we used in the model to avoid overfitting, which includes the four dropout layers that we added. This means that the model at validation is more general and robust, which would in turn lead to a greater validation accuracy and recall than training.

At the first epoch, the accuracy for the validation and training data are both approximately 0.62, while recall is 0.57 and 0.58 respectively. The accuracy for validation then rose and hovered around 0.73, while trraining oscilated around 0.71. In regards to recall, the validation data experienced sudden drops at the third and ninth epochs, but otherwise rose to about

0.7, while the training data hovered around 0.65. Let us observe the models capacity to classify our own written descriptions in 3.8.

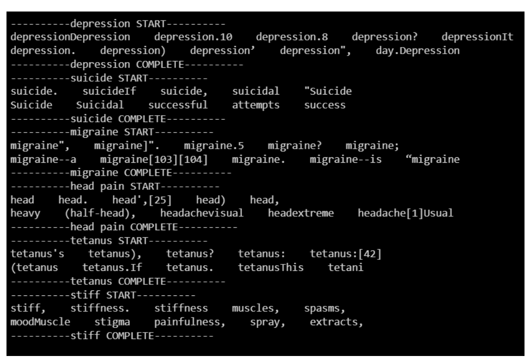


O

Figure 3.8: FastText Preprocessor - CNN Confusion Matrix

In 3.8, we see that the model was fairly effective at classifying depression, which had a precision and recall of 0.8, but struggled to accurately label migraines and tetanus. When classifying migraines, the model had a precision of 0.56 and recall of 1, however tetanus had a precision and recall of 0.

3.5.2 Discussion



```
----------depression START----------
depressionDepression    depression.10    depression.8    depression?    depressionIt
depression.    depression)    depression’    depression",    day.Depression
----------depression COMPLETE----------
----------suicide START----------
suicide.    suicideIf    suicide,    suicidal    "Suicide
Suicide    Suicidal    successful    attempts    success
----------suicide COMPLETE----------
----------migraine START----------
migraine",    migraine]".    migraine.5    migraine?    migraine;
migraine--a    migraine[103][104]    migraine.    migraine--is    “migraine
----------migraine COMPLETE----------
----------head pain START----------
head    head.    head',[25]    head)    head,
heavy    (half-head),    headachevisual    headextreme    headache[1]Usual
----------head pain COMPLETE----------
----------tetanus START----------
tetanus's    tetanus),    tetanus?    tetanus:    tetanus:[42]
(tetanus    tetanus.If    tetanus.    tetanusThis    tetani
----------tetanus COMPLETE----------
----------stiff START----------
stiff,    stiffness.    stiffness    muscles,    spasms,
moodMuscle    stigma    painfulness,    spray,    extracts,
----------stiff COMPLETE----------
```

Figure 3.9: The 10 most proximate words for depression, suicide, tetanus, stiff, migraine, head pain (in order).

The intent behind the use of the FastText model primarily lied in a belief, the retention of semantic meaning would allow the model to connect terminology from the general domain to terminology fom the medical domain.

While there was some preliminary indication that similar words were in close proximity, most words only differed in punctuation as seen in Figure 3.9. Simply put, the semantic meaning of each sentence was likely lost during processing.

Furthermore, some sentences contained largely irrelevent bodies, such as citations or questionaires. Thus, the dataset was diluted with elements that potentially strayed the CNN from making correct classifications. This can be seen in the model's inability to correctly classify custom Tetanus sentences, where a majority of the citations where found. In retrospective, the drop in accuracy could have been prevented through stricter filtering of the data prior to feeding it to FastText model.

Chapter 4

Conclusion

Through our examinations, we have tested the efficacy of a Convolutional Neural Network's ability to classify descriptions of symptoms of an ailment as their most probable diagnoses. In so doing, we created two implementations of a CNN, one of which utilized generated training instances and One Hot Encoding while the other used sentences from the corpus and word embeddings. Through testing the two models ability to classify our written accounts of symptoms, we found that the first implementation was more effective at accurately labelling descriptions than the second. Going forward, there are several ways in which the model can become more robust. The addition of more ailments and associated web pages would provide more variety in the possible classifications the model can discern.

References

Guarino, A. (2019, Jun). *Study finds 89% of us citizens turn to google before their doctor.* Wect News 6. Retrieved from https://www.wect.com/2019/06/24/study-finds-us-citizens-turn-google-before-their-doctor/