Classification of Ailments Given Description of Symptoms

Mukhammadnaim, Urmzd[1] and MacDonald, Benjamin James[2]

[1,2]Faculty of Computer Science, Dalhousie University

December 12, 2021

List of Figures

3.1 Fast Text Model Preprocessor - CNN 8
3.2 One Hot Encoding Preprocessor - CNN 8

4.1 One Hot Encoding Preprocessor - CNN Accuracy 9
4.2 One Hot Encoding Preprocessor - CNN Recall 9
4.3 One Hot Encoding Preprocessor - CNN Confusion Matrix . . . 10
4.4 FastText Preprocessor - CNN Accuracy 11
4.5 FastText Preprocessor - CNN Recall 11
4.6 FastText Preprocessor - CNN Confusion Matrix 12
4.7 The 10 most proximate words for depression, suicide, tetanus, stiff, migraine, head pain (in order). 13

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

4 Analysis **9**
- 4.1 CNN with One Hot Encoding 9
  - 4.1.1 Results 9
  - 4.1.2 Discussion 10
- 4.2 CNN with FastText 11
  - 4.2.1 Results 11
  - 4.2.2 Discussion 13

5 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 CNNs provided a flexible model which performed well with classifying text, and has been used to classify symptoms at a sentence level to a great degree of success(Hughes, Li, Kotoulas, & Suzumura, 2017). Afterwards, we were able to observe the network's capacity to classify the dataset, and proceeded to test its ability to categorize our own descriptions of the three ailments, "depression", "migraine" and "tetanus".

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 a conversational chatbot that utilizes neural networks and decision tree classifiers to classify the ailment that a particular user is experiencing given responses to a questionnaire. Due to a sparsity of publicly available datasets, they created a file with custom patterns classified by their expected responses. They used the response file to train a neural network composed of two fully-connected layers, as well as dropout layers to prevent overfitting. The decision tree classifier was trained with a dataset of binary values indicating whether 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 neural network was able to achieve an accuracy of 95%. The decision tree classifier functioned as intended. The implementation, however, does not provide a seamless series of interactions for the user as they converse with the chatbot(Kurup & Shetty, 2022).

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 compared their accuracy to other NLP techniques. Their vectorization model was trained through the use of a dataset containing 15,000 clinical research documents. A series of clinical articles were labeled as one of 26 medical categories with 4000 sentences being randomly selected for each classification. These sentences were vectorized by the Word2vec model and used to train the CNN. The setup went on to perform with an accuracy of 68%. The model outperformed other text classification techniques such as logistic regression using bag-of-words, which possessed an accuracy of 51%. The former model was effective at displaying the success of CNNs in this context and on a larger scale with 26 individual ailments. The use of word embeddings provided the neural network greater depths of understanding for the relationships between separate words and their associations. These benefits, however, are dependent on large quantities of available data to train the Word2vec model(Hughes et al., 2017).

There is a clear 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 the context of monitoring social media for posts mentioning a drug name. It would classify them as one of three categories indicating the degree of medication. 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 in terms of sentence structure and diction. Professional text can contain a variety of rare terms which can impact a model's capacity to properly classify a description, should a large corpus be unavailable(Tokala, Gambhir, & Mukherjee, 2018).

Chapter 3

Methodology

3.1 Data Collection

One of the challenges in this endeavour was finding accessible and well documented open-source records. As medical data was difficult to come across, due to HIPAA, PIPEDA, and other laws that bar hospitals, and various medical organizations from distributing patients' health documents, we had to settle for medical encyclopedias and articles. To ensure some consistency in the data used across the desired classifications, the descriptions were pulled from the same sites. Between articles on the same website, the detail in which the article was written varied greatly. Some articles had more advertisements, others had more embedded miscellaneous content, and most of them differed in document structure. In other words, the data could not be accessed directly via its url and Document Object Model (DOM) selector. The data was collected using the top-most 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). As a result, there existed 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(Pedregosa et al., 2011) and Witten-Bell Probability Distribution(Bird, Edward, & Ewan, 2009), to retain morphological-level information and generate unique sets of words that could later be fed into the CNN. Subsequently, we analyze the use of the FastText(Rehurek & Sojka, 2011) 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 T_1 which applied a series of reduction operations to remove punctuation, ultimately breaking the words into their stems. The result of T_1 consisted of a set of unique stems D_n, where n represents the order in which the document was processed. After T_1 was applied to all documents, the union of all the sets was taken to form the vocabulary V. Subsequently, V was fed into the *FreqDist* class from the *nltk.probability* package to allow the subsequent usage of the *WittenBellProbDist*. By allocating $|V|$ bins, and using the *FreqDist* of D_n, we ended up with an instance of the *WittenBell-ProbDist*(Bird et al., 2009) 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(Bird et al., 2009) described above, we transformed the data using an instance of the *OneHotencoder*(Pedregosa et al., 2011) class fit on the the vocabulary V. As a result, 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 tokenized by sentence using an instance of the *PunktSentenceTokenizer*(Bird et al., 2009). 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*(Rehurek & Sojka, 2011), 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 vector length, the sentence length varied. As a result, a transformer T_2 was created to pad the given sequences. 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(De Boom, Van Canneyt, Demeester, & Dhoedt, 2016).

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 (Figures 3.1 and 3.2).

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 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 passed through a Flatten layer to reduce the resulting feature map onto a single column matrix, allowing it to be transformed in later operations. Afterwards, 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. Finally, the last Dense layer was used to reduce the learned values onto a probability distribution matching the cardinality of the classification size(Chollet et al., 2015).

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	[(None, 56, 32, 1)]	0
conv2d_10 (Conv2D)	(None, 54, 30, 3)	30
max_pooling2d_10 (MaxPooling2D)	(None, 27, 15, 3)	0
conv2d_11 (Conv2D)	(None, 13, 7, 3)	84
max_pooling2d_11 (MaxPooling2D)	(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 (MaxPooling2D)	(None, 27, 2104, 3)	0
conv2d_3 (Conv2D)	(None, 13, 1051, 3)	84
max_pooling2d_3 (MaxPooling2D)	(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

Chapter 4

Analysis

4.1 CNN with One Hot Encoding

4.1.1 Results

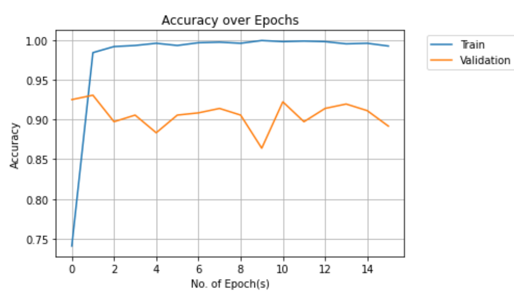

Figure 4.1: One Hot Encoding Preprocessor - CNN Accuracy

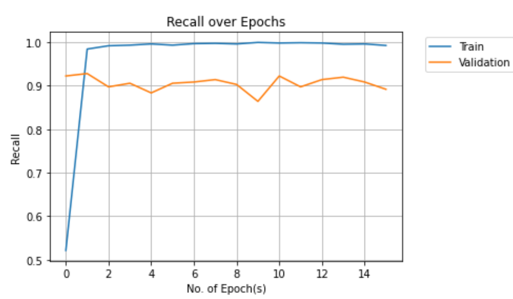

Figure 4.2: One Hot Encoding Preprocessor - CNN Recall

When observing the accuracy in the first implementation, we found that at the beginning, validation accuracy was higher than training accuracy with values of 0.925 and 0.74 respectively. From the second epoch, training accuracy oscillated around 0.99 and validation accuracy hovered around 0.9, as demonstrated by Figures 4.1 and 4.2. Afterwards, we mapped the model's capacity to classify our written descriptions as either "migraine", "depression" or "tetanus".

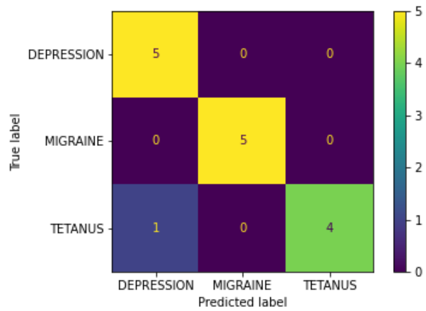

Figure 4.3: One Hot Encoding Preprocessor - CNN Confusion Matrix

The model was highly effective at classifying descriptions of depression, migraines and tetanus, as observed in Figure 4.3. Depression had a precision of 0.83 and a recall of 1.0 while tetanus had a precision and recall of 1.0 and 0.8 respectively. For migraines, the model returned both a precision and recall of 1.0.

4.1.2 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 word stems in the corpus. The large number of features slowed the model dramatically, and led to an increased training time. In an attempt to reduce dimensionality and provide the model with some form of semantic meaning, we decided to use word embeddings in the next implementation.

4.2 CNN with FastText

4.2.1 Results

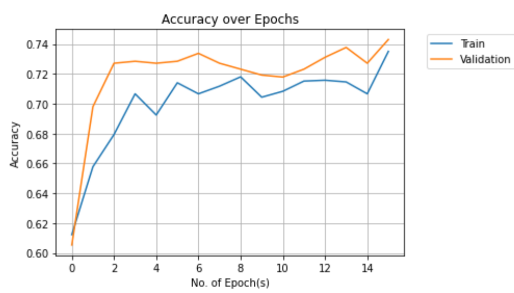

Figure 4.4: FastText Preprocessor - CNN Accuracy

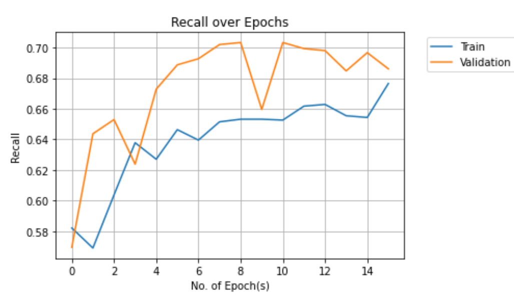

Figure 4.5: FastText Preprocessor - CNN Recall

For the second implementation, we observed that validation accuracy and recall was greater than that of the training set, as seen in Figures 4.4 and 4.5.

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 training oscillated 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.

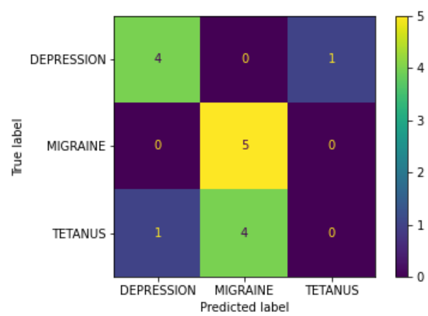

Figure 4.6: FastText Preprocessor - CNN Confusion Matrix

We see that the model had some success at classifying custom descriptions, as visible in Figure 4.6. For depression, both precision and recall were 0.8, but the CNN 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.

4.2.2 Discussion

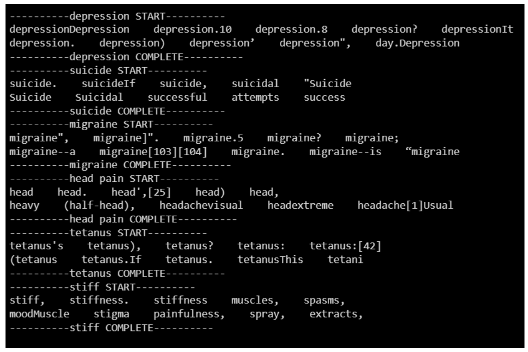

Figure 4.7: The 10 most proximate words for depression, suicide, tetanus, stiff, migraine, head pain (in order).

The intent behind the use of the FastText model primarily lied in the belief, that the retention of semantic meaning would allow the model to connect terminology from the general domain to terminology fom the medical domain(Rehurek & Sojka, 2011). While there was some preliminary indication that similar words were in close proximity, most words only differed in punctuation as seen in Figure 4.7. 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 descriptions, where a majority of miscellaneous content was found. In retrospect, the drop in accuracy could have been prevented through stricter filtering of the data prior to feeding it to the FastText model.

We believe that the validation accuracy and recall was higher than that of the training set's due to the regularization we used in the model to avoid overfitting. This includes the four dropout layers that we added. As a result, the model at validation is more general and robust, which in turn led to a greater accuracy and recall.

Chapter 5

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. 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 a FastText embedding preprocessor(Rehurek & Sojka, 2011). 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

Bird, S., Edward, L., & Ewan, K. (2009). *Natural language processing with python.* "O'Reilly Media Inc.".

Chollet, F., et al. (2015). *Keras.* `https://keras.io`.

De Boom, C., Van Canneyt, S., Demeester, T., & Dhoedt, B. (2016, Sep). Representation learning for very short texts using weighted word embedding aggregation. *Pattern Recognition Letters*, *80*, 150-156. Retrieved from `http://dx.doi.org/10.1016/j.patrec.2016.06.012` doi: 10.1016/j.patrec.2016.06.012

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/`

Hughes, M., Li, I., Kotoulas, S., & Suzumura, T. (2017). Medical text classification using convolutional neural networks. *CoRR*, *abs/1704.06841*. Retrieved from `http://arxiv.org/abs/1704.06841`

Kurup, G., & Shetty, S. D. (2022). Ai conversational chatbot for primary healthcare diagnosis using natural language processing and deep learning. In A. K. Das, J. Nayak, B. Naik, S. Dutta, & D. Pelusi (Eds.), *Computational intelligence in pattern recognition* (pp. 259–272). Singapore: Springer Singapore.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Rehurek, R., & Sojka, P. (2011). Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, *3*(2).

Tokala, S., Gambhir, V., & Mukherjee, A. (2018, October). Deep learning for social media health text classification. In *Proceedings of the 2018*

EMNLP workshop SMM4H: The 3rd social media mining for health applications workshop & shared task (pp. 61–64). Brussels, Belgium: Association for Computational Linguistics. Retrieved from https://aclanthology.org/W18-5917 doi: 10.18653/v1/W18-5917