Personalized Medicine: Redefining Cancer Treatment

Susie Elangbam, Hosung Hwang, Quan Le, Varun Verma, Eric Stone

May 7, 2019

1 Introduction

1.1 Problem

This project revisits a competition that was hosted on Kaggle. With the collaboration of researchers at Memorial Sloan Kettering Cancer Center, people are seeking ways to use genetic sequencing data to better improve our understanding of cancer and produce more targeted treatments. Researchers, specifically, are trying to classify one of 9 classes of genetic mutation. Currently, the interpretation is manually classfied, using evidence from clinical text. Thus, this project seeks to develop a way to classify the genetic mutation using known ML algorithms in order to help reduce the tedious process.

1.2 Structure of the Data

Since this is from a Kaggle competition, the data utilized in training and testing the models are from Kaggle. Because the clinical field primarily label genetic material with strings to classify mutation, data is primarily strings. In particular, the data comes in 3 types: Gene, Variation, and text files, which seem to be passages clinical text. Similarly, since the text file field comes from a clinical passage, some amount of data cleaning is required. Likewise, to understand the distribution of the data, the data has been charted in these figures in the repository: class.png, gene.png, and variation.png.

1.3 Survey of Related Works

As this is a classification problem with string inputs there are many known models such Bag of Words, Neural Network, etc. From looking at known kernels, possible suggested algorithms are Bag of Words, TF-IDF, Word2Vec with reported accuracies of 50.4%, 55.1%, 57.8%, respectively.[1] This project's ap-

[1]This kernel has simple implementations of these methods with no data cleaning. Bag of Words Has slight difference in implementation than this project's approach

proach for this problem consist of 3 different methods: Bag of Words, generic Neural Network, and LSTM Neural Network.

2 Method

2.1 Bag of Words

Doc2vec, also called a paragraph vector, is a NLP way to to represent documents by encoding them as numeric feature vectors. To build a document vector using a distributed bag of words approach, a document vector is trained by being forced to predict random words from the paragraph as its output. To build this model, we built dataframes of the source data by combining the document feature files and abstract text files into a testing and training dataframe was determined by the data providers at Kaggle. The test of the abstracts was processing into Spacy document objects and then the stop words were removed using the base Spacy english language model. The source data was then converted into tagged document objects where the classification of the mutation type was attached to the associated text data. Two types of document were created for each document, one used the text from the paper abstract and one used the concatenation of the gene and the variation type. The tagged documents were trained into a document vector using the Doc2Vec model in Gensim and then classified using a linear regression model. This training was done at three different alpha values (0.002, 0.001, 0.003) with 0.002 being the recommended default value and for multiple training epochs (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 30, 40, 60, 80, 100).

2.2 Neural Network

In order to set up the neural network, the data for neural network requires building a bag of words. First, we built a dictionary from the 3321 samples. Each key of the dictionary is a unique word in the original database, and the value of the dictionary is the frequency of occurrence of each word in the original database. Then, we removed the keys whose length was equal to 1 from the dictionary in order to optimize the code running time when training the neural network and do away with extraneous entities in the data set. Next, we filtered out any key which had a frequency that ranked less than 10 or more than 90. The results are a final bag of words that has 21,362 keys, which are all in lowercase. Finally, we built a 2D array. The number of rows was the number of samples, and the number of columns was the size of bag of words. Each element was the frequency of a word in a sample. After the data been built, we looked into the algorithms and techniques for the Neural Network. In this approach, we built a forward-propagation neural network by utilizing the tensorflow library to determine the accuracy of our method in relation to the question at hand. In order to optimize the running time of the neural network training, we used sixty threads concurrently in our Google Colab code implementation, significantly improving the training time from several hours to just under an hour. To train the forward-propagation neural network, we used a deep network approach using a total of four layers, consisting of one input layer, two hidden layers, and one output layer, with the input and hidden layers using the sigmoid activation function. The mini-batches consisted of chunks of one hundred elements within the dataset and ten epochs, or iterations over samples, were utilized over the course of time in which the algorithm ran. The layers were then configured as followed: Layer 1 as the input layer utilizing the tensorflow function tf.keras.layers.Flatten, Layer 2 as the hidden layer with 4096 nodes using the sigmoid activation function utilizing the tensorflow function tf.keras.layers.Dense, Layer 3 as the hidden layer with 512 nodes using the sigmoid activation function utilizing the tensorflow function tf.keras.layers.Dense, and Layer 4 as the output layer with 1 node using the sigmoid activation function utilizing the tensorflow function tf.keras.layers.Dense

2.3 LSTM Neural Network

We also utilized a LSTM Neural Network as this architecture has shown success in natural language processing types of problems. For this approach, we first translated up each word into a token value and selected the 2,000 most often occurring tokens. Thus, each word was mapped to a dictionary with its corresponding token value so we could translate the text to a sequence of integers. We also padded the sequences to a length of 2,000 as it was important for each sequence to have the same length for efficient matrix operations in training time. Then we initialized the Sequential model from keras to create a linear stack of layers and added an embedding layer, an LSTM layer and the final Dense output layer. We added an embedding layer to avoid the sparsity issue that occur in natural language processing due to the sheer amount of words in the English language and because embedding layers have proven to understand relationships in text data. The LSTM layer specified its output dimension as 196 outputs and the dropout as .25 of the neurons. The final dense layer consisted of 9 output layers to represent 9 of the different classes of gene mutations and this used a softmax activation function. The loss function used to find the parameters was the categorical cross entropy loss function and the adam optimizer was utilized to converge the gradient quicker. After training the data on 8 epochs with a batch size of 32, we applied the trained neural to the testing data obtaining an accuracy.

3 Results

Once we tested all the models we have found that the bag of words, neural network, and LSTM, has an accuracy of 80%, 18.5%, and 54.5%, respectively. In particular it seems that the generic neural network did the worst while the LSTM neural network did the best. With a 54.5% as the best result we got for the neural network, it seems that we have not done better than the researched

TF-IDF model or the Bag of Words models. When obtaining an accuracy of 54.5% with the LSTM model, it was also noted that validation loss did not decrease after epoch 6. For the neural network model, the accuracy of training amounted to 18.48%. It did not improve over the course of future epochs and the accuracy of validation amounted to 13.80% on each epoch. It did not get higher in the next epochs. This approach needs to improve many aspects to get better accuracy such as lower the bag of words and better configuration for the neural network. Finally both our Bag of Words model seem to do better than both neural networks. For the lower values of alpha, 0.001 and 0.002, the model trained on the abstract text (AT) outperformed the model trained on just the gene and variant (GV), but only by about 10% at best with the GV trained model consistently having about 70% accuracy vs about 80% accuracy on the F1 score. For both models the GV accuracy was consistent for the number of training epochs with a slight drift downward as the number increased. The AT model however, had its accuracy drop by about half around 20 epochs for the alpha of 0.002 and at round 10 epochs for the 0.001 alpha. The model trained at an alpha of 0.003 had a slightly lower overall average and its AT model dropped of at around 8 training epochs. From this, for use of this model, our tests suggest that training on the abstract text data is best with a small alpha value and a small number of training epochs will produce results as good as a greater number of training epochs without risk of a drop off. If you are pressed for time and compute resources, train the model on just the gene involved and the variation type will produce results of almost the same accuracy with no risk of significant drop off using significantly fewer computational resources.

4 Conclusion

At the end of the day, we have not seem to found a model can compare to the tedious process that the researchers go through when categorizing mutuations with our best result yeilding about a 80% accuracy. But while this is not a direct solution, the models we constructed could be modified further to get better results in the future.

References

[1] Basic NLP: Bag of Words, TF-IDF, Word2Vec, LSTM
https://www.kaggle.com/reiinakano/basic-nlp-bag-of-words-tf-idf-word2vec-lstm