Drug Recommendation System using NLP Tasks

Samer Baslan   
Department of Computer Science  
San Jose State UniversitySan Jose, CA, USA  
samer.baslan@sjsu.edu

Jimmy Liang  
Department of Computer Science  
San Jose State UniversitySan Jose, CA, USA  
jimmy.liang@sjsu.edu

Rohan Kumar  
Department of Computer Science  
San Jose State UniversitySan Jose, CA, USA  
rohan.kumar@sjsu.edu

***Abstract*— Oftentimes, different people react in various ways towards the same drugs, making drug efficacy vary amongst large populations. With the UCI Machine Learning Drug dataset, we gather enough information about drug user reviews to create a Drug Recommendation system that eliminates the use of drugs that receive unanimously negative reviews, and suggest useful alternative drugs to people whose prescribed drug isn’t working for them. Using a collaborative filtering approach, we make use of reviews made by users that comment about the side effects and the usefulness of the drug. We utilize the power of NLP to provide drug recommendations to the user based on their symptoms.**

***Keywords—Recommendation System, NLP, Collaborative Filtering, Machine Learning, UCI ML Drugs***

# Introduction

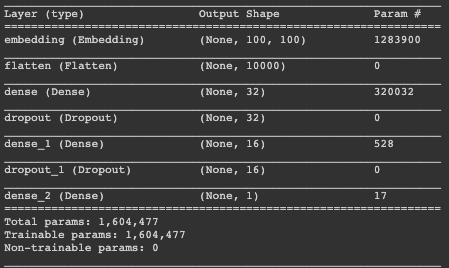
When creating Recommendation Systems, there are numerous approaches to be taken depending on the nature of the data that is collected. Mainly, Recommendation Systems could have 3 different components [1]: Candidate Generation, Scoring Systems, and Re-Ranking Systems. Given that the UCI Machine Learning Drug dataset provides ratings, reviews, and comments about user condition, we can use different approaches to classify the efficacy of a drug and whether it should be recommended or not. Specifically, we will take both the Candidate Generation approach, to offer users alternative drugs that have a high overall rating and seems to be satisfactory for their users, as well as a Scoring System approach, where we eliminate the drugs considered ineffective by unanimous decision in the dataset, and keep the drugs that have worked for a decent size of the population. Our Candidate Generation approach will make use of NLP methods to generate a candidate drug for the user given their symptoms based on the reviews from other users, whereas the Scoring System will make use of the numeric ratings given by the reviewers to determine whether a drug is considered “Good” or “Bad”.

# METHODOLOGY

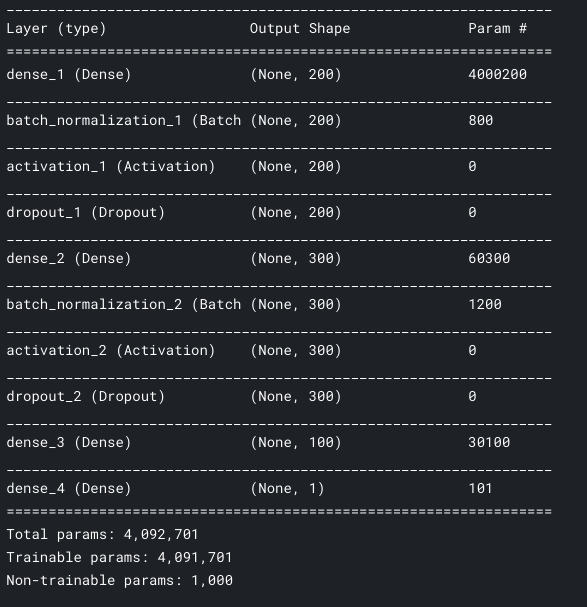
The first step we have to take before making use of the dataset is to clean and preprocess it to our use. The initial dataset’s columns are the following: [Drug Name, Rating, Effectiveness Rating, Side Effect Rating, Condition, BenefitsReview, SideEffectsReview, CommentsReview]. Given that the Effectiveness Rating and the Side Effect Rating are given in different string categories, careful mapping of values should be made to be eventually combined into a single column that combines the Benefits Rating as well as the Side Effects Rating into a single “usefulCount” feature. The Benefits Rating is made up of values of “Highly Effective”, “Considerably Effective”, “Marginally Effective”, “Moderately Effective”, and “Ineffective”. Thus, given that Highly Effective is our most desired outcome, we map that category to a value of 5, and the rest all the way from the values of 4-1. As for the Side Effects Review, the values consist of the following values: “Extremely Severe Side Effects”, “Severe Side Effects”, “Moderate Side Effects”, “Mild Side Effects”, and “No Side Effects”. Given that our most desired outcome is No Side Effects, we map that value to a 5 and other values map to 4-1, backwards. This mapping allows us to combine the ratings into a feature that takes into account the Benefits as well as the Side Effects, and can be used later in the Scoring System approach that we will take. Another important mapping to be created was the rating class of either “Good” or “Bad”. This was simply created by mapping the overall rating values to “1” for those with ratings higher than 5, and “0” otherwise. This will be our target class for the Scoring System.

We decide to make use of the reviews to feed our Scoring System approach as well. First, we start by concatenating all three review columns to preprocess with NLP techniques such as stemming and lemmatization. Second, we use the Keras Tokenizer to build our string index and transform strings into lists of integers. Then, we create a word index dictionary which maps words into integer indices to prepare for our next step; Word2Vec. We utilize Word2Vec to find the similar words in our dataset, as they will benefit us when we move on to the Candidate Generation approach when the recommendations are being made to associate the drugs with the symptoms and comments that people have about them. We also create a TSNE plot to understand the structure of the similar words that our model is picking up, as well as verify that the word connections that the model is creating are making sense.

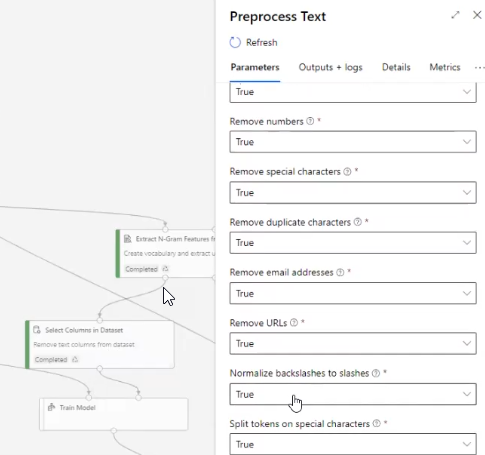


Lastly, we create an embedding of the word index and their mappings before we feed it into the model. We create a 3-layer neural network that takes in an Embeddings Layer and a Flattening Layer prior to the Dense Layers. This resulted in overfitting in the model, and thus we included the 2 Dropout layers between our Dense layers. With this architecture, we were able to achieve a training accuracy of 96%, and a validation accuracy of 0.76%. This means that the model is actually picking up on whether a drug is considered useful or not a significant amount of times. 

The Candidate Generation approach will also require its separate set of preprocessing techniques to make use of the Benefits Reviews, Side Effects Reviews, and Comments Reviews, as they all contain useful text that can benefit our recommendation system. Thus, first we concatenate all three review columns to preprocess with NLP techniques before feeding it into our model that will generate the recommendations. We base the next steps on the efforts of Kaggle user “Hyun Woo Kim” [2]. We first perform basic NLP methods such as transforming into lower case, removing numbers/special characters, stemming and lemmatization. Then, we pass the cleaned, combined reviews into a CountVectorizer with n gram range of (4, 4). Then, we split our data into training and test sets, and create the model. The model architecture consisted of 3 sets of Dense, Batch Normalization, Activation, and Dropout layers. Here, we run into a major error that prevented us from creating recommendations for a while, as this API worked in previous versions of Keras and Tensorflow, but currently produces a cryptic SpareTensorError, for which there was no available documentation for fixes and many attempts have been made to fix with no success. We move on to a different approach.

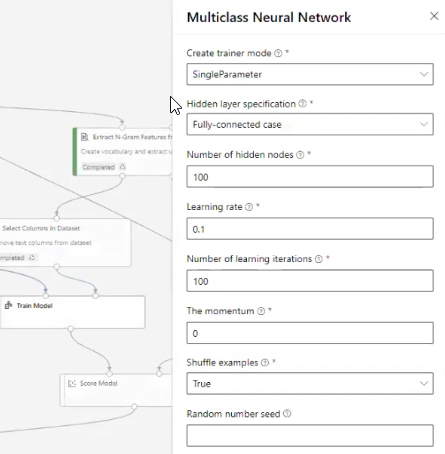


As the eventual model and inference will reside on the Azure Machine Learning environment, we decided to leverage the ML capabilities offered by Azure to replicate the python code steps and model creation inside of Azure. This also allowed us to produce an inference pipeline for serving up predictions. Within the Azure ML Dashboard, we were able to recreate the data cleaning workflow, stripping stop words, special characters, removing URLs, in a single block.

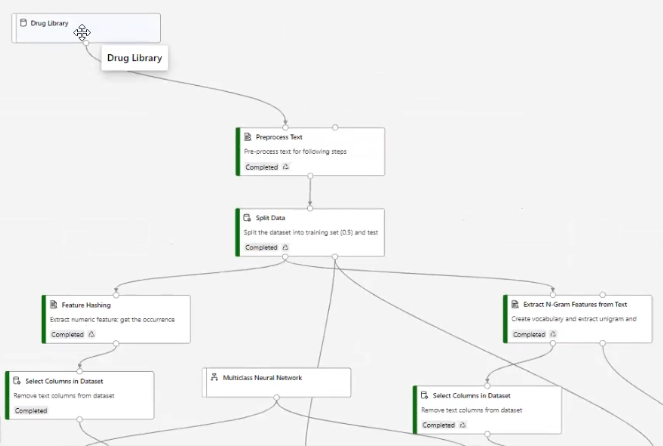


The output of this block is then passed as inputs to the next block, which we have defined as feature hashing and extracting n-grams features from the text columns with n=2. This processing created an embeddings of the features, which we have selected as the conditions that a particular drug is treating.

We then defined a multiclass neural network as a model to train. Azure has the option to try multiple parameters, but since we already know what we wanted to try, we selected the single parameter option. We could have defined our own model architecture to fully mimic our notebooks as described above, but this required the definition to be in a Net# language that we are not familiar with. Given the time constraints we went ahead with a classic fully connected neural network with 100 hidden nodes.



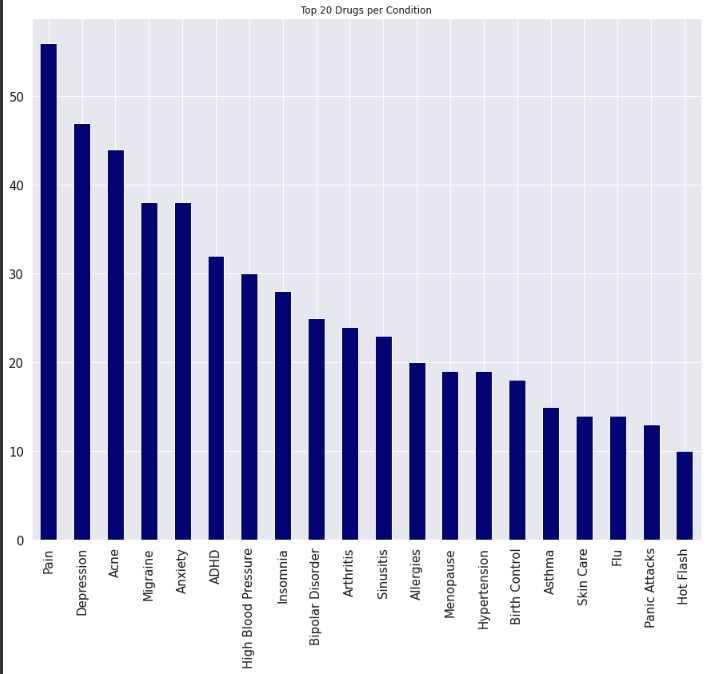
We continued creating the training pipeline by connecting the appropriate inputs and outputs of each block module. Our data source, the drug library dataset described earlier in this document, was added to our data store and selected as the top node. Our training pipeline, then looks like this: With the dataset as the source, we passed the data into the text preprocessing step. We did a split of the data (50% split to be able to train 2 separate models to compare accuracy) and created feature hashing and n-gram from texts. The whole training step takes ~2 hours given our limited CPU machine.



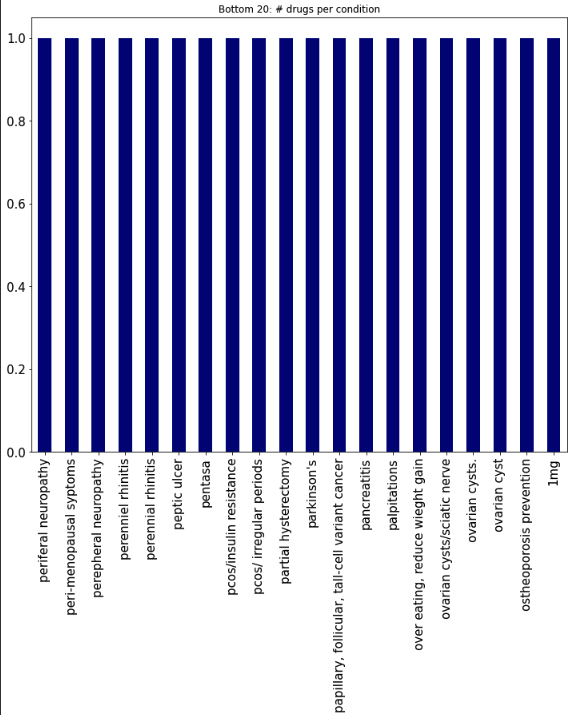
Once the model is trained, we converted the data cleaning steps as well as the word embedding steps into an inference pipeline, this allows for the text string sent in for prediction to go through the same data preparation steps for inference. We were then able to deploy the model using a container instance and expose a REST API for inference. This allows our simple Flask app to send a phrase describing a symptom and receive a prediction of the closest matching drug name as a prediction.

# DATA ANALYSIS

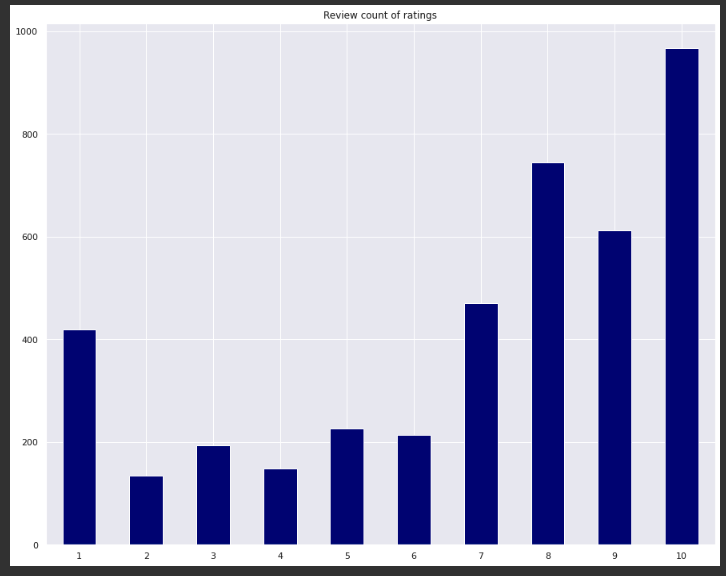
To understand the structure of the data, we perform some data analysis and visualizations that will benefit us throughout the project and give us some insight into our data. Given that the conditions of the users were input with the freedom of using any language to describe symptoms, we needed to perform some manual preprocessing to combine conditions that contained either common misspellings, or used a verbose description of their symptoms. Also, oftentimes the same conditions can be described using different words. To overcome that, we had to search for specific patterns in the conditions column. In the end, we were able to identify the top 20 most common conditions:



The most prevalent conditions are Pain, Depression, and Acne with counts of over 40. If we look at the bottom 20 drugs, we see that they’re mostly an issue of preprocessing. They all have counts of 1, since no other row in the data contains the same text. Our condition cleaning function was able to capture the most common misspellings/verbose texts, but not all of them.

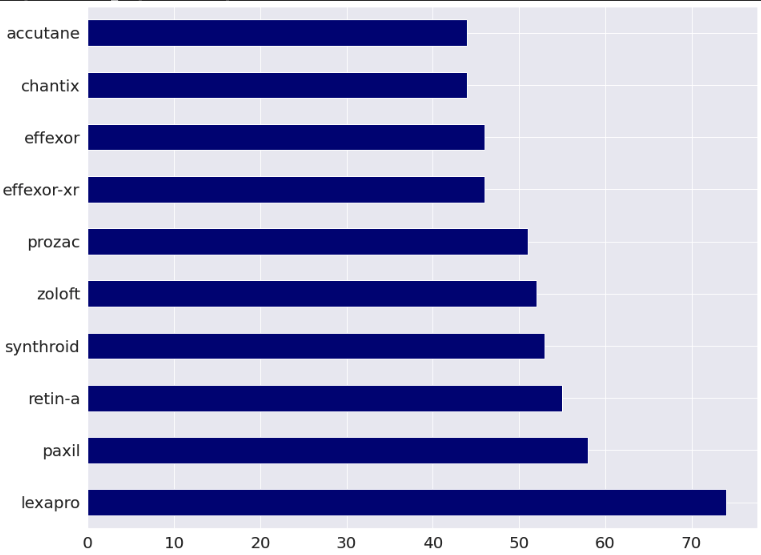


Additionally, it is important to check to see the most prevalent rating classes, to see how the data is distributed. In our case, we can see that the majority of the drugs were given a full rating of 10, which means that we can expect to see more “Good” drugs than “Bad” ones.



We can expect this to have some effect on the classification bias, but oversampling techniques would not have worked due to the fact that our system is based on text reviews, and undersampling would have not been useful as we don’t have enough data to discard.

Another useful characteristic of the dataset to understand is the highest available drugs in the dataset. We see here that the drug with the most reviews in this dataset is “Lexapro”, with a count of over 70, whereas most of the other drugs lie in the 10-40 mark.



##### Acknowledgment

We would like to acknowledge UCI for providing the dataset that we used for this project. We would also like to acknowledge Professor Eranti and the San Jose State MSSE program for their guidance and support.

##### References

1. Roy, A., 2021. Introduction To Recommender Systems- 1: Content-Based Filtering And Collaborative Filtering. [online] Medium. Available at: <https://towardsdatascience.com/introduction-to-recommender-systems-1-971bd274f421> [Accessed 15 May 2021].
2. H. W. Kim, “Recommendation Medicines by using a review.” Bucheon-si, Gyeonggi-do, Bucheon-si, Gyeonggi-do, 01-Dec-2018.