MediBuddy – Health Assistant

**Team Members: -**

Nirmal Narayanaswamy Karunanithi (A0215406N)

Tushar Metha (A0215466A)

Aayush Mathur (A0215470M)

Contents

[Overview 2](#_Toc68293762)

[Design 2](#_Toc68293763)

[Data understanding 2](#_Toc68293764)

[Data cleaning 2](#_Toc68293765)

[Exploratory Data Analysis 2](#_Toc68293766)

[Sentiment analysis 2](#_Toc68293767)

[User Interface 2](#_Toc68293768)

# Overview

This chatbot aims to make information regarding drugs much faster and easier to use for both patients and physicians. MediBuddy is powered by a sophisticated machine learning system that offers increasingly accurate responses to user questions based on behaviours that it “learns” by interacting with human beings.

# Data understanding

It was data from <https://archive.ics.uci.edu/ml/datasets/Drug+Review+Dataset+%28Drugs.com%29> and crawled reviews from online pharmaceutical review sites.

These are additional explanations for variables.

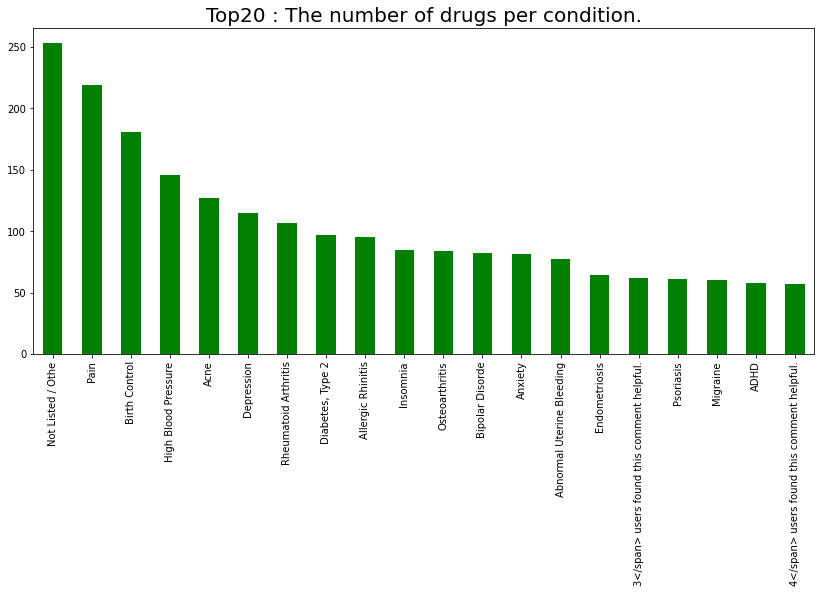
* drug Name (categorical): name of drug
* condition (categorical): name of condition
* review (text): patient review
* rating (numerical): 10-star patient rating
* date (date): date of review entry
* useful Count (numerical): number of users who found review useful

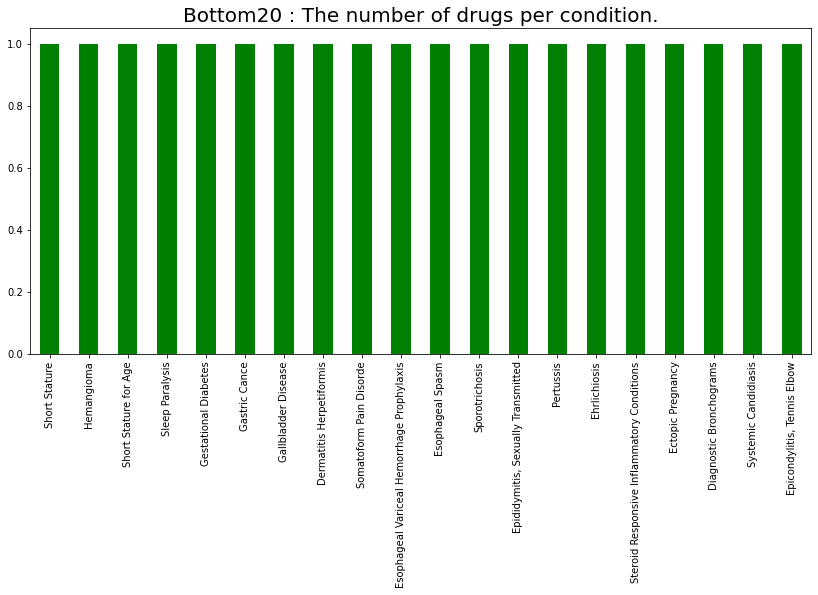
The structure of the data is that a patient with a unique ID purchases a drug that meets his condition and writes a review and rating for the drug he/she purchased on the date. Afterwards, if the others read that review and find it helpful, they will click useful Count, which will add 1 for the variable.

The UCI ML Drug Review dataset provides patient reviews on specific drugs along with related conditions and a 10-star patient rating system reflecting overall patient satisfaction. The data was sourced by web scrapping multiple online pharmaceutical and medical review sites. This data was published in a study on sentiment analysis of drug experience over multiple facets, ex. sentiments learned on specific aspects such as effectiveness and side effects.

# Exploratory Data Analysis

We start with some basic visualisations where we are looking at the top and bottom 20 number of drugs per condition. From the graphs we can see that common conditions like pain and birth control have more than 200 drugs available. For more specific and complicated conditions there is only one drug available.





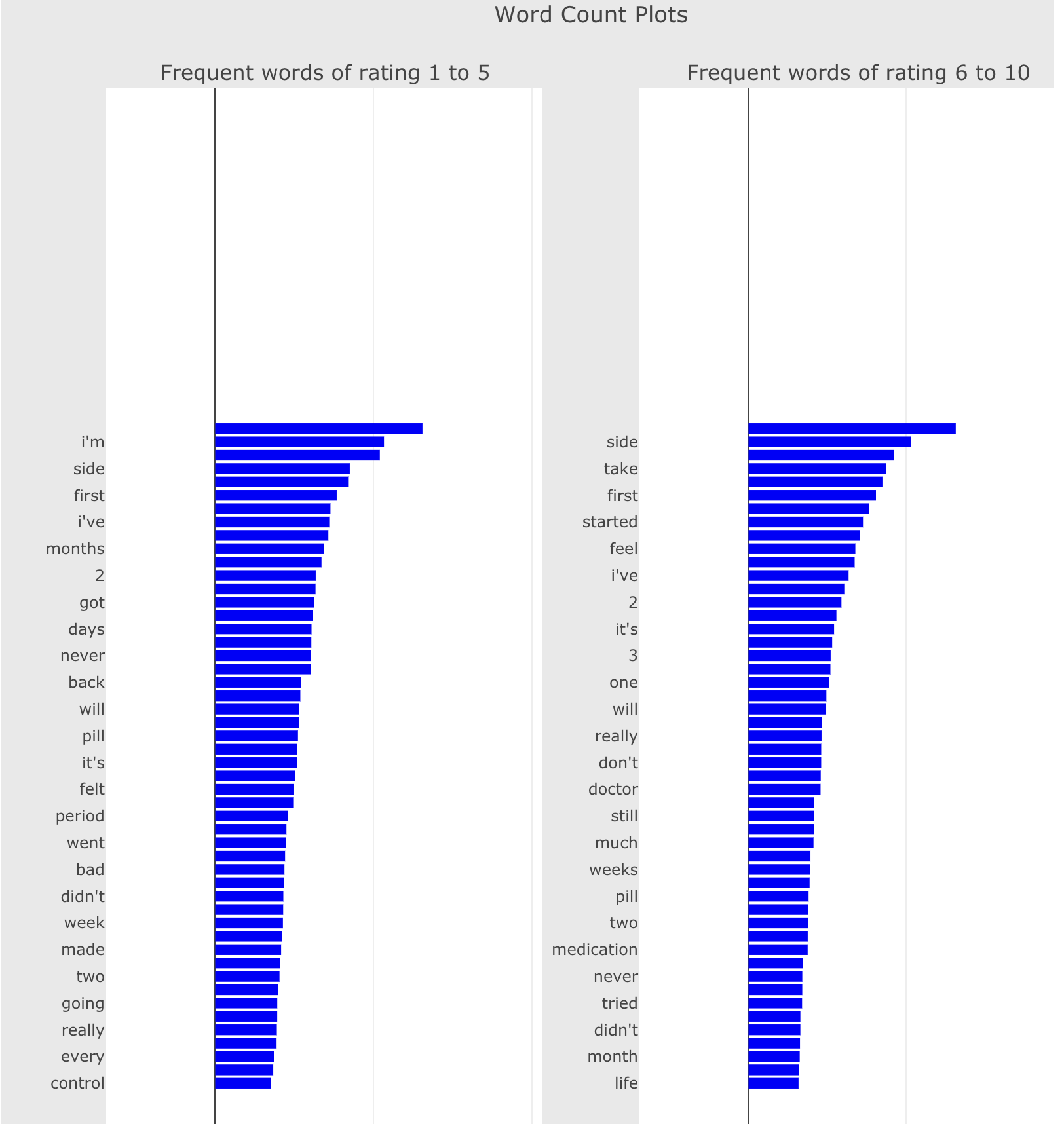
Missing values - Checking the missing values across the columns we find that the condition column has around 1200 missing values which is around 0.55% of the total values. Hence, we can go ahead and delete these rows because it is an exceedingly small percentage.



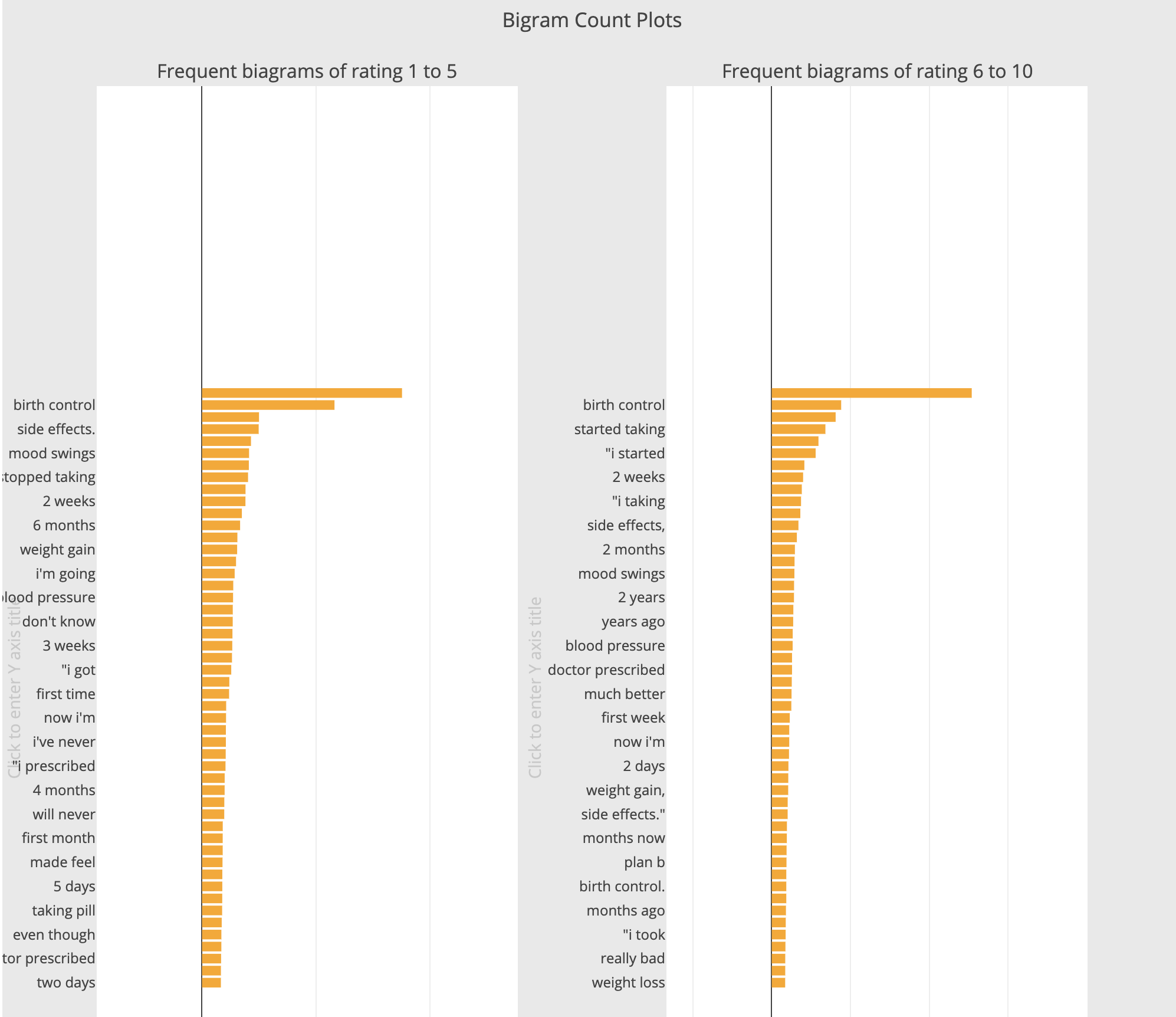


This is the word cloud created from all the medical reviews given by users for the drugs. The bigger the words the greater number of times they appear in our datasets.

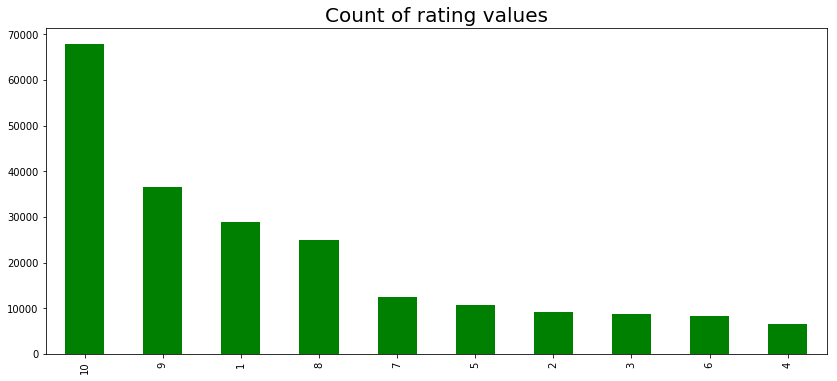
Using frequency of words in the review we are going to categorize 1-5 as negative and 6-10 as positive, and then look at 1 4 grams to see which corpus better categorizes emotions.

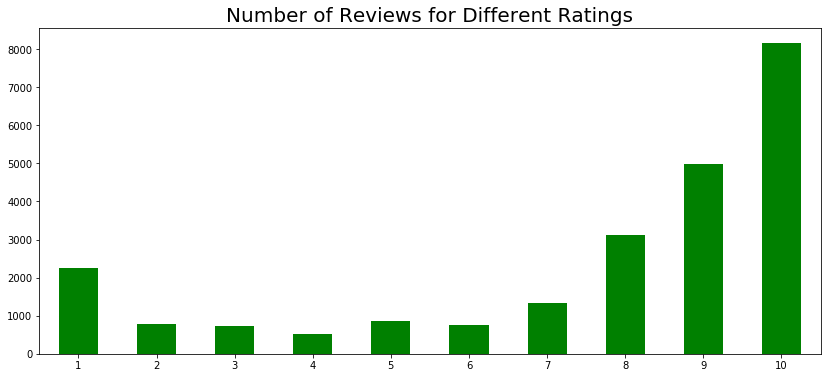


When you use 1-gram, you will find that the top five terms all have the same material, even though the left (negative) and right (positive) orders are reversed. This means that when we interpret the text with a single corpus, the emotion is not well classified. As a result, we will broaden the corpus.

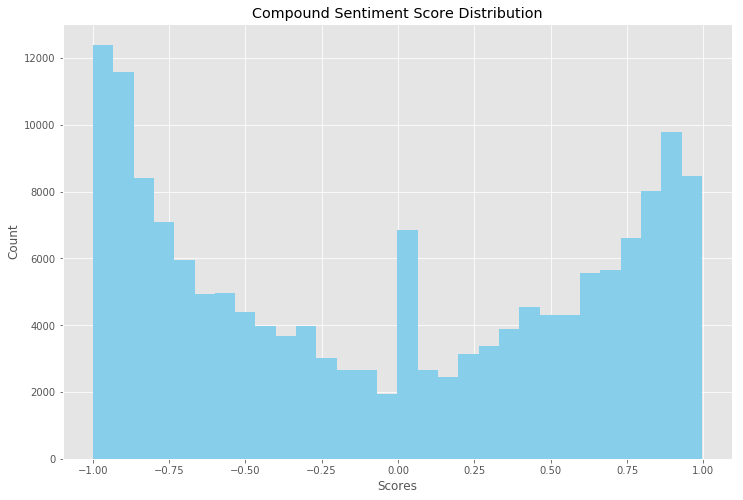


Similarly, the contents of the top five corpora in 2-gram are similar, making positive and negative classification difficult. Furthermore, the terms 'side effects' and 'side effects.' are interpreted differently, necessitating pre-processing of review results. However, as you can see, this is a great way to categorize feelings than previous 1-grams, such as side effects, weight gain, and highly recommend.





The most popular choices are 10, 9, 1, and 8, with 10 being more than twice as common as the others. We can see that the percentage of positives is greater than the percentage of negatives, and that people's reactions are diverse.



# Data Cleaning – Text Pre-processing

As we have lot of unwanted texts in the dataset. We need to do perform preprocessing steps to make the text much cleaner format.

1. Tokenization

Tokenization is the process of breaking down an expression, sentence, paragraph, or even an entire text document into smaller units like individual words or phrases. Tokens are the names given to each of these smaller units. We tried out tokenization of sentences with unigrams, bigrams, and trigrams.

1. Stop words

The most frequently used words in any natural language are stop words. These stop words may not add much value to the context of the document when analyzing text data and constructing NLP (Natural Language Processing) models. These words just to the noise in the data. We removed all kind of stop words from the customer reviews on the drugs.

iii. Removal of punctuation and Special characters

Since the data was sourced from online review sites there were a lot of special characters because of HTML formatting. These were removed to clean the data further.

iv. Lemmatization

The grouping of various forms of the same term to get the root words from the text we have. This will group the similar words and reduce the high curse of dimensionality. Performed Lemmatization and made the text much cleaner and proper format.

# Feature Weighting

Represents sentences with vector of numbers, to distinguish polarity among all the sentences.

i. Count Vectorizer

The Count Vectorizer gives us an encoded vector with the entire vocabulary's length and an integer count of how many times each word appeared in the text. It makes tokenizing a set of text documents and building a vocabulary of known words easy.

It usually only content words (adjectives, adverbs, nouns, verbs) considered as unigram vector features.

Performed count vectorizer approach to make text into sequences and weights calculated based upon the frequency. These weights are further used for model building.

ii. **TF-IDF**

TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is achieved by multiplying two metrics: the number of times a word appears in a document and the word's reciprocal document frequency across a series of documents.

*tf*: term frequency – number of occurrences of term *t*in document *d*

*Idf:* inverted document frequency of term *t*

*df:* the ***document frequency*** of term *t*, i.e., the number of documents that contain the term.

*N*: the total number of documents in the corpus

TF\* IDF —> TFIDF where IDF = log (N/ DF)

TD-IDF vectorizer is much better approach than earlier method where weights are calculated in different way. So, we tried out this method to check how model works I.e. better score/predictions.

iii. Word2Vec

Word2Vec is a method to construct such an embedding. It can be obtained using two methods (both involving Neural Networks): Skip Gram and Common Bag of Words (CBOW).

Word2Vec is once again a different method for calculating feature weights where we used this approach as input features in the Deep Neural Network (DNN) model.

iv. GloVe

GloVe is an unsupervised learning algorithm that generates word vector representations. The resulting representations highlight intriguing linear substructures of the word vector space, and training is based on aggregated global word-word co-occurrence statistics from a corpus.

Glove is word embedding approach where it is learnt and count the data for given features, as it has been used as input features for LSTM models and train the model.

We tried different approaches of Feature weights as input to the model build and train the dataset where we can find out how the model performs for each different method.

# Sentiment analysis - Model

1. Deep Neural Network

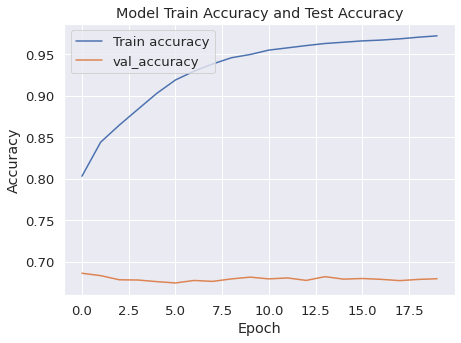
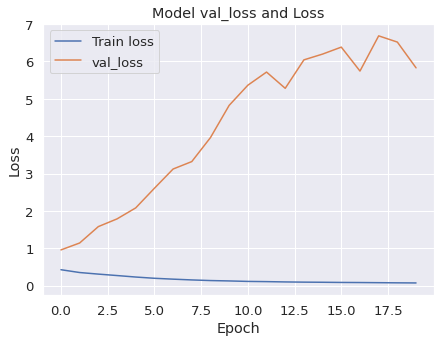
Initially, we tried with Deep Neural Network model (DNN) with below parameters.

I) Count Vectorizer Features

Input layer as **Count vectorizer features** with 200 neurons, batch normalization, "**relu**” as activation function with Dropout value as 0.5. And hidden layer 1 with 300 neurons, and hidden layer 2 with 256 neurons, and final output layer with single neuron and activation function as sigmoid.

Loss function used here, **Binary Cross Entropy** and optimizer as **Adam Function** and evaluation metrics as **Accuracy**.

Epochs: 20, Batch Size: 128

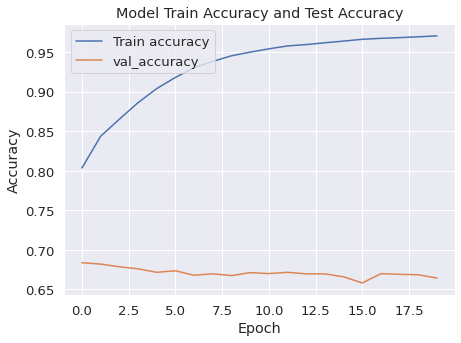
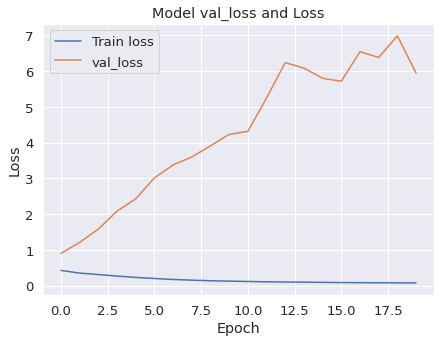
 . 

Train Accuracy: 0.9958 Testing Accuracy: 0.6796 Train Loss: 0.0226 Test Loss: 5.8295

Ii) TD-IDF Vectorizer Features

Then, later tried the model with the same network layers with input as TD-IDF vectorizer features.

Epochs: 20, Batch Size: 128

. 

Train Accuracy: 0.9953 Testing Accuracy: 0.6644 Train Loss: 0.0218 Test Loss: 5.9462

1. Bidirectional LSTM

In previous models. We tried out few combinations where train loss and training accuracy was good, but the validation accuracy and validation loss were not good enough to determine it. So, we tried with LSTM models to check whether we can be able to increase the score test accuracy and minimize the validation loss.

I) BILSTM – Output Dropout

Build a simple Bidirectional LSTM model with input as GLOVE word embedding features and output layer dropout value of 0.5.

No. Of Epochs: 10, Batch Size: 1024



Training Accuracy: 0.8779 Train Loss: 0.2896

Testing Accuracy: 0.8413 Test Loss: 0.3737

Ii) BILSTM – Input Dropout

In order tweaking the parameters of model, we tried out BILSTM model with dropout value of 0.5 in inner layer.

No. Of Epochs: 10, Batch Size: 1024



Training Accuracy: 0.8066 Train Loss: 0.4202

Testing Accuracy: 0.8154 Test Loss: 0.4206

Iii) BILSTM – Recurrent Dropout

Later, we tried the same model but with recurrent dropout of 0.5 and output dropout of 0.5. And optimizer as “Adam,” and loss function as “Binary cross entropy” and activation function as “Sigmoid” and evaluation metrics as “Accuracy”

No. Of Epochs: 10, Batch Size: 1024



Training Accuracy: 0.8291 Train Loss: 0.3856

Testing Accuracy: 0.8298 Test Loss: 0.3839

From, the above different approaches of BILSTM models, we can notice that **model with output dropout** was performing better than other approaches.

1. Transformers

I) BERT Model

Since BERT model is known for its outperformance in text classification task, we tried to fine tune the BERT model with our training dataset.

Here, we used “**Bert-base-uncased**" pretrained model and using Bert Tokenizer generated special tokens [CLS] and [Sep].

Later, perform padding and truncation using enocde\_plus method.

Create a list for input ids and attention masks and convert into tensors.

Then, tensor dataset has been made for train and test using inputs such as input id, attention masks and labels.

DataLoader created for train and validation set. **BertforSequenceClassification** is loaded with no. Of labels as 2 since it is binary classification.

AdamW is a class from the huggingface library which used as Optimizer with parameters with default value such as learning rate = 2e-5 and epsilon = 1e-5. A scheduler has been used with optimizer.

Next part, before start training the model where we perform zero gradient initially since it always clears any previously calculated gradients before performing a backward pass. PyTorch doesn't do this automatically because accumulating the gradients is "convenient while training RNNs".

Make a forward pass (evaluate the model on this training batch). Calculate the average loss by accumulating the training loss for all batches. To measure the gradients, make a backward step.

Clip the norm of the gradients to 1.0. This is to help prevent the "exploding gradients" problem.

Update parameters and take a step using the computed gradient. The optimizer dictates the "update rule"--how the parameters are modified based on their gradients, the learning rate, etc. Update the learning rate.

Calculate the average loss over all the batches. And carry out similar steps for validation set in order to find its accuracy and loss value. So, the above steps are performed for 4 Epochs repeatedly.

Please find below results obtained after fine tuning Bert model for sentiment mining classification.

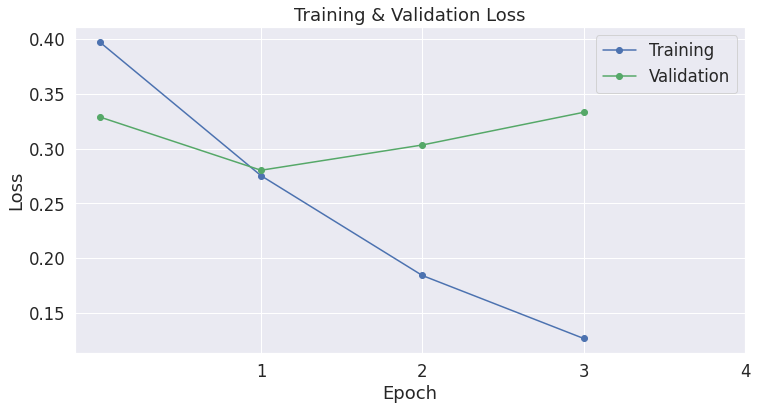
Epoch 1: Average training loss: 0.40, Validation Accuracy: 0.86, Validation Loss: 0.33

Epoch 2: Average training loss: 0.28, Validation Accuracy: 0.89, Validation Loss: 0.28

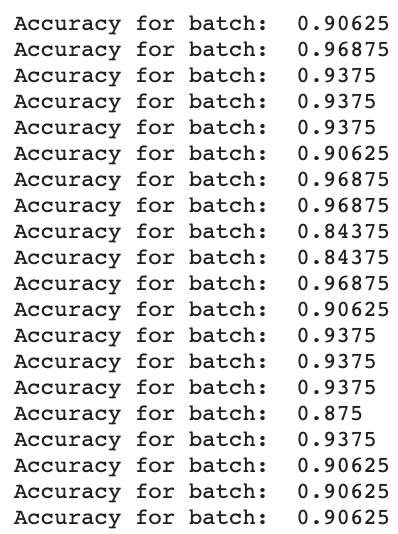
Epoch 3: Average training loss: 0.19, Validation Accuracy: 0.90, Validation Loss: 0.30

Epoch 4: Average training loss: 0.13, Validation Accuracy: 0.91, Validation Loss: 0.32

Validation Loss and Training Loss Graph

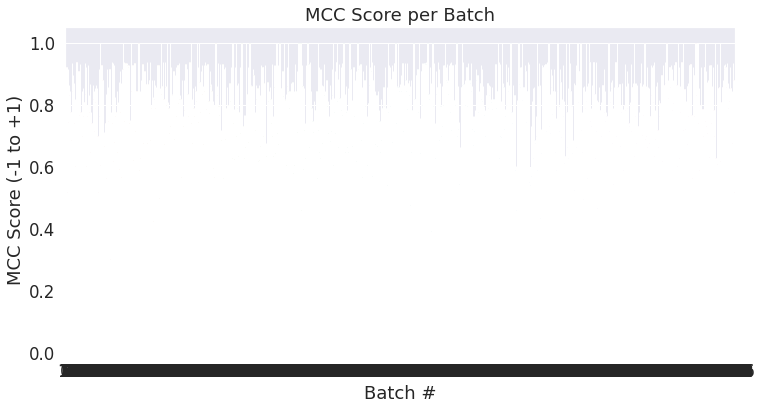


Later, predictions tried for test dataset with total batch size as 32. Here, listed below accuracy for few batches.



Overall accuracy for test set: 91%

And evaluate each test batch using Matthew's correlation coefficient



Total MCC: 0.766

# Model Comparisons

Here, listed below the accuracy and loss value for train and test set, for the various models that are tried out.

BERT and BI-LSTM (recurrent dropout) are outperformed than other approaches since here, both test and train scores are similar, low bias and low variance i.e., perfect model fit.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| MODEL COMPARSIONS | | | | | |
| MODEL NAME | APPROACH | ACCURACY | | LOSS VALUE | |
| TRAIN | TEST | TRAIN | TEST |
| DEEP NEURAL  NETWORK | COUNT VECTORIZER | 99.5% | 68% | 0.02 | 5.82 |
| TDIDF VECTORIZER | 99.5% | 66.5% | 0.02 | 5.95 |
| BI-LSTM | OUTPUT DROPOUT | 88% | 84% | 0.28 | 0.37 |
| INPUT DROPOUT | 81% | 81.5% | 0.42 | 0.42 |
| RECURRENT DROPOUT | 83% | 83% | 0.38 | 0.38 |
| TRANSFORMERS |  |  |  |  |  |
| BERT | 99% | 91% | 0.13 | 0.32 |

But BERT model takes long time duration to train the model compared to BI-LSTM model. So, in future, based on time constraint, we can choose the model and retrain it for further evaluation.

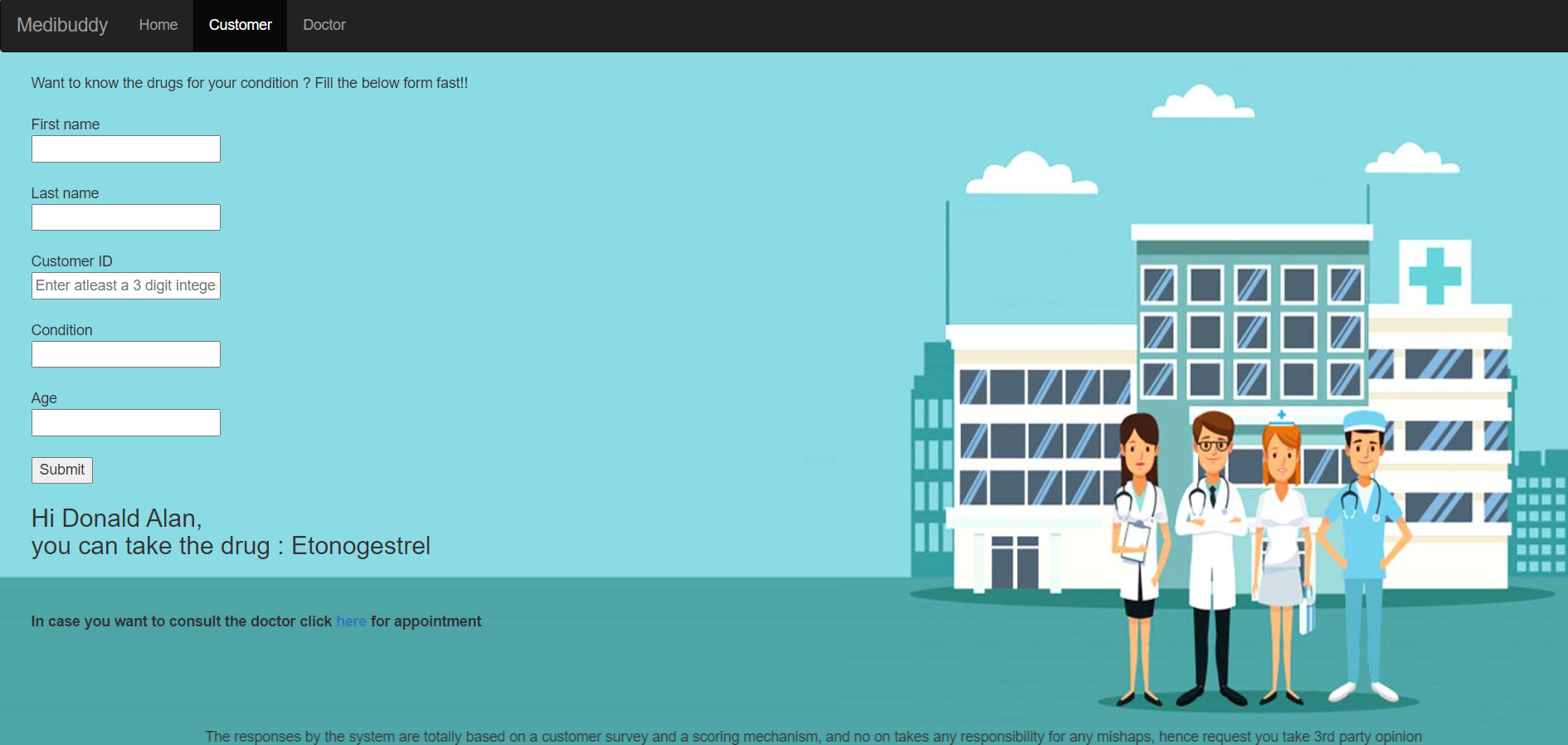
# User Interface

A user interface has also been made to facilitate 2 types of users: -

1. **Customers** 
   1. If a customer wishes to know the uses of a particular drug, he/she can just land on the page and enter the drug name, hit search, and know the conditions where the drug can be used.

Fig: Main landing Page of the Health Assist.

* 1. If the customer is having a condition and wants to know the associated drug for it, he/she can know the best drug which can be used to cure the disease based on different customers' review.

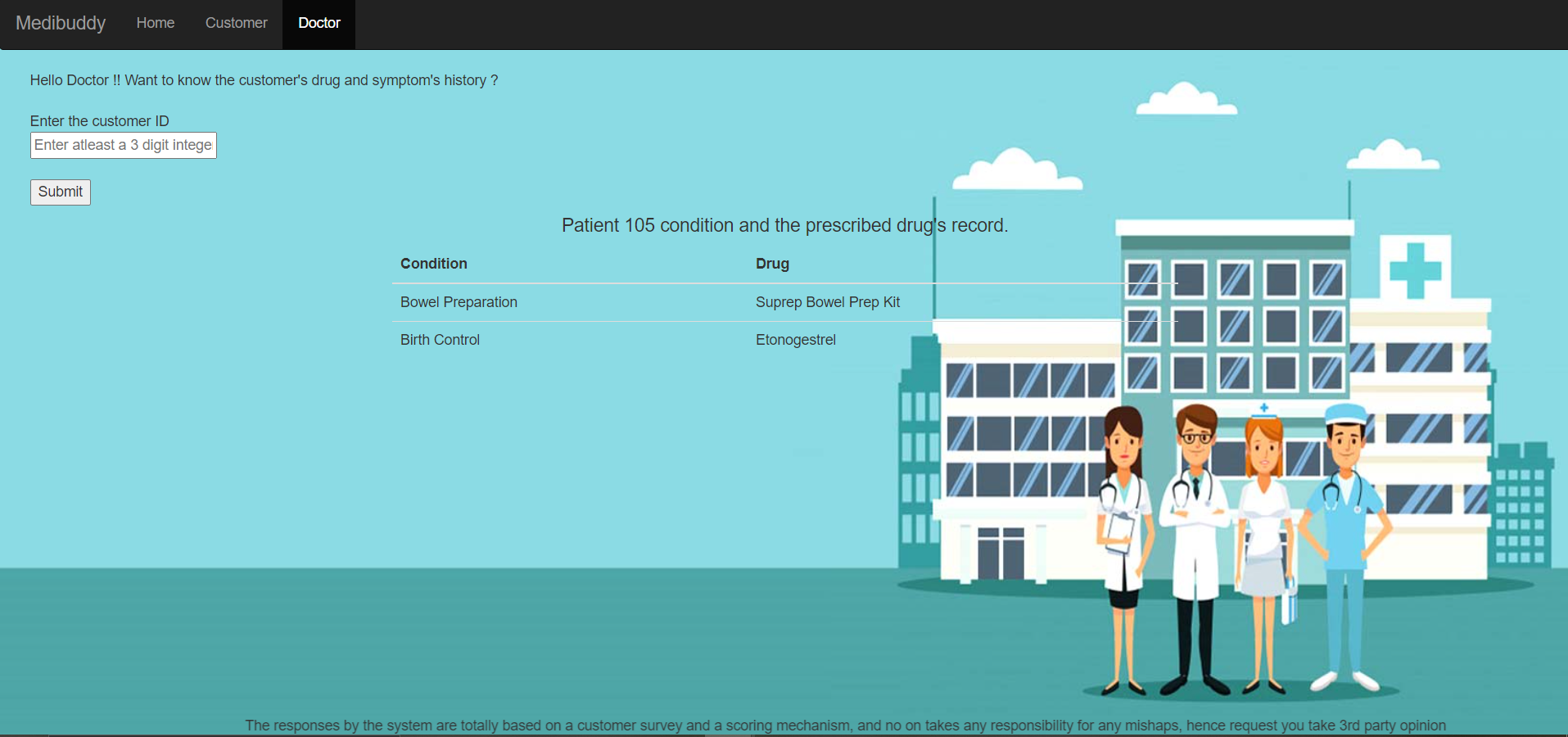
Fig: Customers Information Page

The customers can fill up the form with very minimal details as well as enter their condition and get to know the best drug based on user's recommendation.

The customers can even book an appointment by clicking the link mentioned just below which would redirect them to SingHealth’s website.

1. **Doctors**

In case a patient visits a doctor, and the doctor wants to know the patient's history of symptoms and drugs taken by the patient in the past, he/she can hop on to the doctor's tab and just enter the patient's customer I.D and get a list of drugs prescribed to the patient by the health assist.

Fig: Doctors Home Page

The condition and the drug history are provided of the requested customer id is mentioned is provided in a tabular format.

**Technical Aspects**

The user interface has been built on the following technology stack:

1. Framework: Django (Python’s web development full stack framework)
2. Database: MySQL
3. Front End: HTML, CSS, BOOTSTRAP, JQUERY
4. Back End: Python (NumPy, pandas, Transformers, Pytorch)

# References

1. <https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/>
2. <https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa>
3. <https://docs.djangoproject.com/en/3.1/>
4. <https://getbootstrap.com/docs/5.0/getting-started/introduction/>
5. <http://mccormickml.com/2019/07/22/BERT-fine-tuning/>
6. <https://huggingface.co/transformers/v2.2.0/model_doc/bert.html#transformers.BertForSequenceClassification>
7. <https://github.com/huggingface/transformers/blob/5bfcd0485ece086ebcbed2d008813037968a9e58/examples/run_glue.py#L128>
8. <https://medium.com/huggingface/how-to-build-a-state-of-the-art-conversational-ai-with-transfer-learning-2d818ac26313>
9. <https://github.com/huggingface/transfer-learning-conv-ai>
10. <https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/>
11. <https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/>
12. <https://towardsdatascience.com/bert-text-classification-using-pytorch-723dfb8b6b5b>
13. <https://paperswithcode.com/method/bilstm> - Research Papers
14. <https://medium.com/analytics-vidhya/building-a-text-classification-model-using-bilstm-c0548ace26f2>
15. <https://galhever.medium.com/sentiment-analysis-with-pytorch-part-4-lstm-bilstm-model-84447f6c4525>
16. <https://medium.com/dair-ai/deep-learning-for-nlp-an-overview-of-recent-trends-d0d8f40a776d>
17. <https://towardsdatascience.com/the-unreasonable-progress-of-deep-neural-networks-in-natural-language-processing-nlp-374443b21b00>
18. <https://www.sciencedirect.com/science/article/pii/S016971611830021X>
19. <https://cs224d.stanford.edu/>