Drug Rating Prediction Based on the Drug Review Dataset from UCI Machine Learning Repository

Ting-Chou Lin
A53317103
til002@eng.ucsd.edu

Yufei Mao
A53268374
ymao@ucsd.edu

Yen-Yi Wu
A53318422
yew001@eng.ucsd.edu

ACM Reference Format:
Ting-Chou Lin, Yufei Mao, and Yen-Yi Wu. 2019. Drug Rating Prediction Based on the Drug Review Dataset from UCI Machine Learning Repository. In *Proceedings of Assignment2 (CSE258)*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 DATASET

We found an interesting dataset at UCI Machine Learning Repository, called *"Drug Review Dataset (Drugs.com) Data Set"* [7]. The dataset was obtained by crawling online pharmaceutical review sites, *Drugs.com*. The dataset contains 215063 number of samples. As shown in Table 1, the dataset provides patient ratings from one to ten showing overall patient satisfaction, patient reviews on specific drugs along with related conditions and the date of recording, and finally the number of other users who found that such review is useful.

Table 1: Dataset Attribute

Attribute	Type	Description
drugName	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
usefulCount	numerical	number of users who found review useful

Exploratory Data Analysis

First of all, we would like to understand how is the distribution of the user ratings. As shown in Figure 1, rating ten is the most frequent ratings that user gave, which has 68005 samples. Then, the followings are the rating nine and rating one, which has 36708 and 28918 samples, respectively. That is to say, the total number of samples with rating one, nine, or ten is 133631, which is over half of the number of the overall samples. Hence, we can conclude that the drug users like to give the extreme ratings. Also, if users are not satisfied, they would tend to give the lowest rating, one star, but not two or three stars.

Besides, we would like to see if different dates effect the user ratings. Note that the overall user ratings' mean is 6.99 and the standard deviation is 3.28. As shown in Figure 2, we can observe that the differences from average ratings of each year is huge. Year 2008 has the highest average rating, 8.93 and year 2017 has the lowest average rating, which is 6.04. Even so, there is no obvious long-term temporal effect of years to the ratings, which means, we would need one-hot encoding of each year as features if we would like to predict the user rating. Similarly, the recorded days and months of reviews has the similar views, as shown in Figure 3 and Figure 4, respectively.

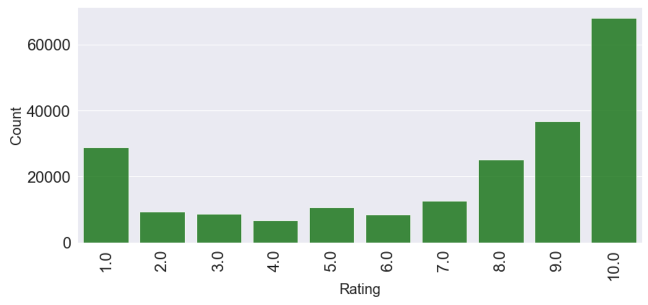

Figure 1: Distribution of numbers of different ratings.

Furthermore, we would like to know what is the correlation between useful counts and user ratings. As shown in Figure 5, we can find that the number of useful counts is highly correlated with the user rating. In other words, the number of useful counts can be a good feature for predicting the user rating.

Last but not least, we do the sentiment analysis on the review text. Here, we use *nltk.sentiment.vader* as our sentiment intensity analyzer. Figure 6 shows the rough distribution of the scores of the sentiment intensity. In addition, we would

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CSE258, Assignment2, UCSD
© 2019 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

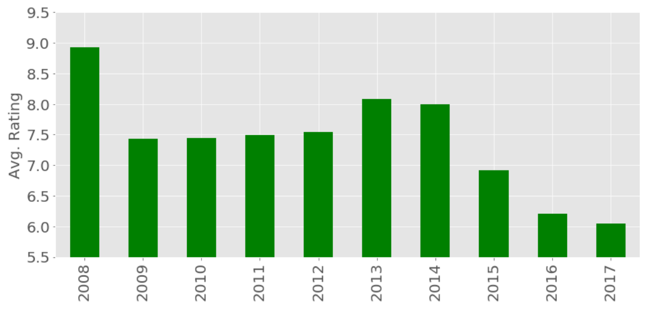

Figure 2: Average user rating of each year.

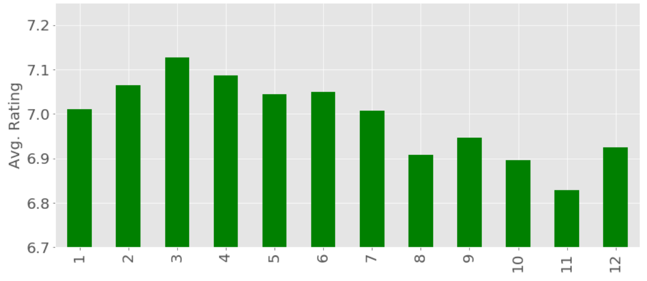

Figure 3: Average user rating of each month.

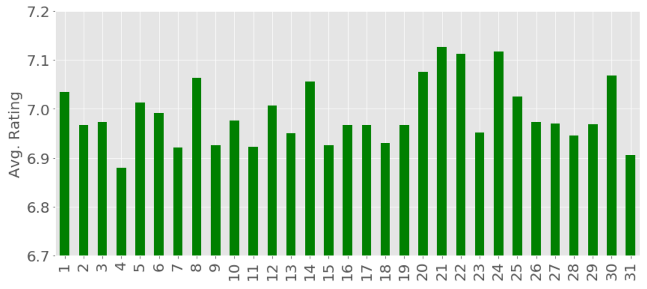

Figure 4: Average user rating of each day.

like to learn the relationship between the sentiment intensity scores of the review text and the user ratings. As shown in Figure 7, we can observe that the average sentiment intensity score of text review is highly correlated with the user rating. Hence, the text review could be a helpful feature when predicting the user rating.

Figure 8 shows the word cloud. We find that some words are daily words like "first","given", which are not predictive for rating, some words looks predictive for rating like "limited","quick","improvement", and there are also medical words like "antidepressants" and meaningless words like "ve". To make daily common words less influential to our model, we could use TF-IDF scores as features. To simplify our training process, we could ignore some words which are strange and rare and just use the most N frequent words, where N differs

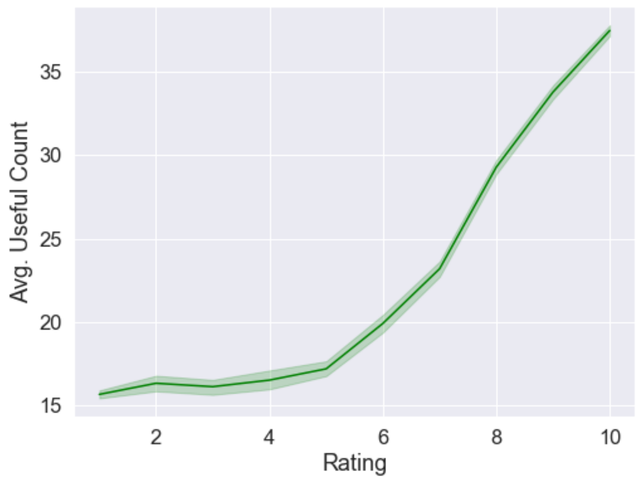

Figure 5: Average useful counts by different ratings.

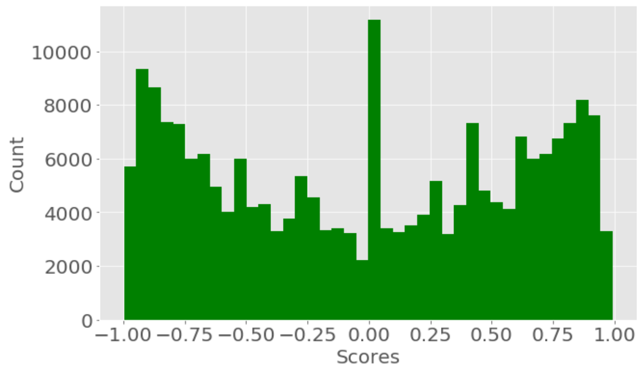

Figure 6: Histogram of the scores of sentiment intensity. Positive values are positive valence, negative value are negative valence.

Table 2: Drug Review Data set Basic Attributes

Attribute	Value
Length of data set	215063
Average rating	6.99
Unique conditions	917
Unique uni-grams in reviews	88495
Unique bi-grams in reviews	1225707

from 3000 to 45000. Some attributes of data set are shown in the Table 2.

2 PREDICTIVE TASK

After doing the exploratory analysis of the data, we find that predicting the user rating based on the users' reviews and other features would be an interesting and challenging

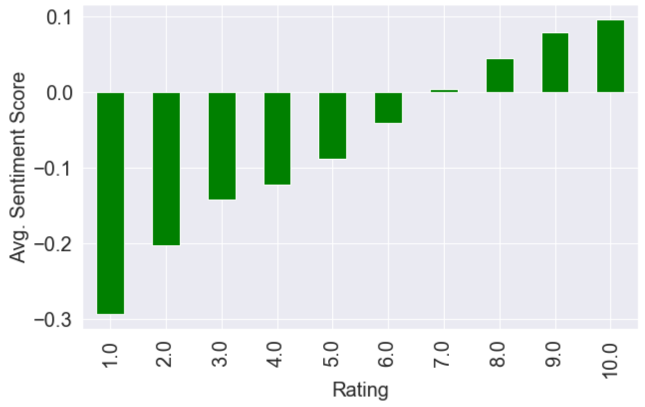

Figure 7: Average sentiment intensity score of review text by different user ratings.

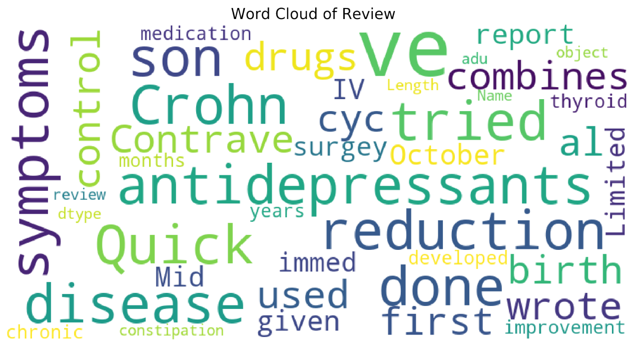

Figure 8: Word cloud for drug review dataset

problem. The study [5] shows that sentiment analysis is often domain-dependent, because the polarity of single terms can be different depending on the context they are used in. Moreover, the language expressions in online forums are highly informal and user-expressed medical concepts are often nontechnical, descriptive, and challenging to extract. The formal problem formulation of our rating prediction problem is defined below:

- Given attributes in the data of, *condition, review, date* and *usefulCount*, predict the *rating* so that the root-mean-square error ($RMSE$) is minimized and the coefficient of determination (R^2) is maximized accordingly.

Model Evaluation

We take five percents of the overall data (about 15000 of records) as test set, perform different models on this test set, and compute the following criteria to evaluate our trained models:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}\left(yi - \widehat{yi}\right)^2 \tag{1}$$

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(yi - \widehat{yi}\right)^2} \tag{2}$$

$$R^2 = \frac{ExplainedVariation}{TotalVariation} \tag{3}$$

Mean Squared Error(MSE) measures the average squared difference between the estimated values and the actual value. Root Mean Squared Error(RMSE) is the square root of the mean squared error. R-Squared is a statistical measure that indicates how much variation of a dependent variable is explained by the independent variable(s) in a regression model.

Data Pre-processing and Feature Generation

In order to facilitate the rating prediction task, several data pre-processing is needed, including the the handle of missing values and the review text pre-processing.

Missing Values Removal. We find that there are about 0.55% of the drug conditions have the following improper texts, *"3</span> users found this comment helpful."*. After removing these samples, the overall number of samples is 213869.

Review Text Pre-processing. We remove the stopwords in review texts and then perform stemming. *nltk.corpus.stopwords* is used. Note that we remove several negative words in the stopwords, like aren't or doesn't, because these words could be very predictive and highly correlated to the sentiment. After then, *nltk.stem.snowball.SnowballStemmer* is used for text stemming.

Baselines

Linear Regression using Review Length as Feature. The first baseline we designed is to use review text length as the feature. After removing punctuation and doing stemming, the total number of words in each review is counted, then we train a linear regression model. The features for this model are [1, text length]. The $RMSE$ and R^2 on test set for this baseline are 3.2619 and 0.0018.

Average Rating for Condition. As ratings may be relative to conditions, this baseline takes one-hot encoding for conditions as input features. The output is the average rating for a specific condition. For example, the average rating for the condition "von Willebrand's Disease" is 10. Given the record with condition of "von Willebrand's Disease", this predictor will predict the rating to be 10. The average rating for each condition is calculated by using data in the training set. The $RMSE$ and R^2 on test set for this baseline are 2.6149 and 0.0848.

Ridge Regression with 3000 Unigrams. This baseline uses ridge regression, taking [Bag-of-words count, one-hot-condition, usefulCount] as features. First of all, 3000 most frequent unigrams are extracted. Then, for each text in the training set,

Table 3: Selected Features

Feature	Encoding	Dimension
condition	integer label and one-hot	916
review	bag-of-words or TF-IDF	3000–45000
rating	float	1
year	one-hot	10
month	one-hot	12
day	one-hot	31
usefulCount	integer	1

the number of each unigram appearing in the text is counted and vectorized, used as the feature of Bag-of-words count. Besides, we use one-hot encoding to represent conditions. The *RMSE* and R^2 on test set for this baseline are 1.9788 and 0.4372.

Feature Generation

As shown in Table 3, we encode the data attribute into various features to ease the following rating predicting task. First, we use *sklearn.preprocessing.LabelEncoder* [12] to label the attribute *condition*. Then, *sklearn.preprocessing.OneHotEncoder* is used for one-hot encoding the attribute *date* and the labeled attribute *condition*. For the encoding of the pre-processed review text, *sklearn.feature_extraction.text.TfidVectorizer* and *sklearn.feature_extraction.text.CountVectorizer* are used.

Bag-of-words. In the Bag-of-Words model, first, most frequent n words are related with a unique ID, wordID, where n varies from 3000 to 45000. Number of each word in the text are counted. We vectorize these numbers by their wordIDs, and put this list into feature vector. In our experiment, not only uni-grams are counted, bi-grams, tri-grams are also counted using the similar method.

TF-IDF. In the TF-IDF model, the TF-IDF score of each word in the text are calculated and used as part of the feature. To calculate TF-IDF, we use following equations:

$$IDF(t, D) = log(N/DF(t, D)) \tag{4}$$

$$TF - IDF(t, d, D) = TF(t, d) * IDF(t, D). \tag{5}$$

where TF(w,d) means number of times word w appears in the document d, DF(w,D) means number of documents that contains w. Similarly, we give each word a unique ID and serialize the TF-IDF scores by their wordIDs.

Appropriate Models

In order to predict the user rating, we think that the appropriate models would be linear regression, ridge regression, gradient boosting regression and random forest regression. The details of these models are discussed in Section 3. Note that because the text reviews usually contains numerous noises and meaningless words, we think that the decision tree regression model will fall into overfitting easily and thus is not suitable for this prediting task. Also, since we don't have the user information (i.e. user ID for each review) in the dataset, the cosine and Pearson similarities are not suitable for the predicting task on this dataset.

3 SELECTED MODELS

Intuitively, regarding rating predicting problem, linear regression that we learned from the class first comes to our mind. To begin with, we deploy linear regression model and try different features (more details are in Section 5). We then adopt ridge regression, which is also called regularized model. Regularization is the process of penalizing model complexity during training. However, the solution quality of both linear regression and ridge regression are not good enough to give a promising predicting result, say, R^2 over 0.7. We search online and find that gradient boosting and random forest models based on decision tree algorithm might be useful, because more trees give a more robust model and prevent overfitting.

In this section, we introduce four models (linear regression, ridge regression, gradient boosting and random forest) that we use in this report and how we optimize them. We also present comparisons between these models and report unsuccessful attempts and scalability issue. Finally, we explain the overfitting issue when training the model.

Linear Regression

As professor gave in the lecture, given a response vector $y \in R^n$ and a predictor matrix $X \in R^{n \times p}$, the linear regression coefficients are defined as:

$$\widehat{\theta}^{linear} = argmin_{\theta \in R^p} \frac{1}{N} ||y - X\theta||_2^2. \tag{6}$$

$||y - X\theta||_2^2$ represents as loss (MSE). The optimization problem is to find a set of $\widehat{\theta}^{linear}$, that can minimize the MSE.

Ridge Regression

Ridge regression is like least squares but shrinks the estimated coefficients towards zero [8]. Given a response vector $y \in R^n$ and a predictor matrix $X \in R^{n \times p}$, the ridge regression coefficients are defined as:

$$\widehat{\beta}^{ridge} = argmin_{\beta \in R^p} ||y - X\beta||_2^2 + \lambda ||\beta||_2^2. \tag{7}$$

$||y - X\beta||_2^2$ represents as loss; $||\beta||_2^2$ represents as penalty. λ is a tuning parameter, which controls the strength of the penalty term. When $\lambda = 0$, we get the linear regression estimate; while when $\lambda = \infty$, we get $\beta^{ridge} = 0$. For λ in between, we are balancing two ideas: fitting a linear model of y on X, and shrinking the coefficients. Like linear regression, the optimization problem is to find a set of $\widehat{\beta}^{ridge}$, that can minimize the loss. We here can utilize cross-validation to tune λ.

Gradient Boosting

Gradient boosting is a gradient descent algorithm. We compare several gradient boosting algorithms and find that gradient tree boosting [3] would be best one to adopt due to its effectiveness. Like other boosting methods, gradient boosting combines weak "learners" into a single strong learner in an iterative fashion. It is easiest to explain in the least-squares regression setting, where the goal is to "teach" a model F to predict values of the form $\hat{y} = F(x)$ by minimizing the MSE.

At each stage m, $1 \leq m \leq M$, of gradient boosting, it may be assumed that there is some imperfect model F_m. The gradient boosting algorithm improves on F_m by constructing a new model that adds an estimator h to provide a better model: $F_{m+1}(x) = F_m(x)+h(x)$. To find h, the gradient boosting solution starts with the observation that a perfect h would imply $h(x) = y - F_m(x)$. Therefore, gradient boosting will fit h to the residual $y - F_m(x)$. As in other boosting variants, each F_{m+1} attempts to correct the errors of its predecessor F_m.

Generic gradient boosting at the m-th step would fit a decision tree $h_m(x)$ to pseudo-residuals. J_m be the number of its leaves. The tree partitions the input space into J_m disjoint regions $R_{1m}, \ldots, R_{J_m m}$ and predicts a constant value in each region. Using the indicator notation, the output of $h_m(x)$ for input x can be written as the sum:

$$h_m(x) = \sum_{j=1}^{J_m} b_{jm} \mathbf{1}_{R_{jm}}(x), \tag{8}$$

where b_{jm} is the value predicted in the region R_{jm}.

Then the coefficients b_{jm} are multiplied by some value γ_m, chosen using line search so as to minimize the loss function, and the model is updated as follows:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x), \tag{9}$$

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)). \tag{10}$$

It chooses a separate optimal value γ_{jm} for each of the tree's regions, instead of a single γ_m for the whole tree. The coefficients b_{jm} from the tree-fitting procedure can be then simply discarded and the model update rule becomes:

$$F_m(x) = F_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}_{R_{jm}}(x), \tag{11}$$

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma) \tag{12}$$

Here, we can optimize the model by tuning the depth of the trees and iterations M (i.e., the number of trees). On the other hand, we can also tune the learning rate. The learning rate definition is as below:

$$F_m(x) = F_{m-1}(x) + \nu \cdot \gamma_m h_m(x), \quad 0 < \nu \leq 1, \tag{13}$$

where parameter ν is called the *learning rate.*

Random Forest

The training algorithm for random forests applies the general technique of bootstrap aggregating to tree learners [10]. Given a training set $X = x_1, ..., x_n$ with responses $Y = y_1, ..., y_n$, bagging repeatedly selects a random sample with replacement of the training set and fits trees to these samples. After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} f_b(x') \tag{14}$$

or by taking the majority vote in the case of classification trees.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees; bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on x':

$$\sigma = \sqrt{\frac{\sum_{b=1}^{B} (f_b(x') - \hat{f})^2}{B - 1}}. \tag{15}$$

We here adopt cross-validation to find a better the number of samples/trees, B.

Comparison

The advantages of linear/ridge regressions are their properties that make them easy to interpret and deploy; while the weakness is that they only model relationships between dependent and independent variables that are linear. The strengths of gradient boosting solve linear regression's weakness. Gradient boosting can optimize on different loss functions and provides several hyperparameter tuning options that make the function fit very flexible. Where as the weaknesses of gradient boosting are computationally expensive and less interpretable. Also, gradient boosting will continue improving to minimize all errors, which will overemphasize outliers and cause overfitting. The strengths of random forest are: (1) they provide a reliable feature importance estimate (2) the predictive performance can compete with the best supervised learning algorithms. On the other hand, random forests also have a few disadvantages: (1) an ensemble model is inherently less interpretable than an individual decision tree (2) training a large number of deep trees can have high computational costs and use a lot of memory. The following two sub-sections, we compare both regressions and gradient boosting with random forest.

Linear/Ridge Regression versus Random Forest. In Regression analysis we fit a curve/line to the data points, in such a manner that the error between the value of fitted regression model at each point is and the actual value is minimized. Where as in random forest we make multiple decision trees. To classify a new object based on attributes, each tree gives a classification and we say the tree “votes” for that class. The forest chooses the classification having the most votes (over all the trees in the forest) and in case of regression, it takes the average of outputs by different trees. Random Forest is able to discover more complex relation at the cost of time. We cannot identify obvious linear relationship from our dataset in Section 1. Therefore, we conjecture that the outcome of the random forest model would be better than the outcome of the regression. The experimental result in Section 5 proves our surmise.

Gradient Boosting versus Random Forest. Like random forests, gradient boosting is a set of decision trees. The two main differences are:

- How trees are built: random forests builds each tree independently while gradient boosting builds one tree at a time. This additive model (ensemble) works in a forward stage-wise manner, introducing a weak learner to improve the shortcomings of existing weak learners.
- Combining results: random forests combine results at the end of the process (by averaging or "majority rules") while gradient boosting combines results along the way.

Since our dataset contains numerous statistical noise, we conjecture that the outcome of the gradient forest might easily result in overfitting. On the other hand, because gradient boosting tends to be harder to tune the parameters than random forests, we here infer the result of the random forest would be better.

Scalability and Unsuccessful Attempts

We tried to use Kernel ridge Regression (KRR) and Support Vector Regression (SVR) with kernel trick to learn the non-linear function in the original space. However, both of the models ran into the scalability problems because of our relatively large data (i.e. over 150k). Even if running the task on a server with 126GB memory, both of the models still reported errors of memory allocation. To address this issue, we can use a smaller dataset to train the model, but need to aware of the potential underfitting. In addition to KRR and SVR models, we also ran into scalability issue when training the gradient boosting model, which is shown in Section 5.

Table 4: Comparison of training and Testing MSE with different # word representing features (W).

Model	W	Training MSE	Testing MSE
Bag-of-Words	20000	4.251992	5.220429
	25000	4.000093	5.190206
	30000	3.75456	5.187375
	35000	3.528651	5.17876
	40000	3.308389	5.182842
TF-IDF	20000	3.800098	4.733297
	25000	3.552582	4.711584
	30000	3.321771	4.686358
	35000	3.106133	4.652426
	40000	2.899742	4.633232
	45000	2.706859	4.622013
	50000	2.515409	4.684884

Overfitting

When applying Bag-of-Words model and TF-IDF model to generate features for linear regression model, we run in to an issue of overfitting. In our experiment, we keep increasing number of words which are used as features from 20000 with a step of 5000. As shown in Table 4, the best number of textual representation features for Bag-of-Words model is 35000, and the best for TF-IDF model is 45000. If the number of textual representation features is higher than this best one, then the training MSE keeps decreasing, but the test *MSE* increases, which is a phenomenon of overfitting. To address overfitting, we choose appropriate number of textual representation features of the trained models by cross validation.

4 LITERATURE REVIEW

To the best of our knowledge, non of existing work predicts exact users' ratings based on the drug review dataset [7] that we use in this report. In this section, we review some previous works which used the same dataset. We then survey other similar datasets and related works in the past. Finally, we summarize the state-of-the-art methods currently employed to study this type of data and compare other works with our finding.

Preliminary on Dataset [7] and Related Works

We find that all the works used the same dataset tackled different problems. Gräßer et al. provided the drug review dataset [7] and presented a methodology to predict patients' satisfactions. They divided dataset into three categories (negative, neutral, positive) based on patients' ratings shown in Table 5. They applied a n-grams approach to represent patients' reviews. That means unigrams, bigrams as well as trigrams were used to derive features. Then, they adopted logistic regression to train their model. The experimental result shows 92.24% accuracy. The work [2] by Apurva presented k-means clustering algorithm. Their model clustered the unlabeled drug reviews dataset in order to group the drugs with similar usage and benefits. Chen et al. [4] proposed an implementation for automatic sentiment classification of drug reviews employing fuzzy rough feature selection. Fuzzy-rough feature selection provided a means by which discrete or real-valued noisy data (or a mixture of both) can be effectively reduced without the need for user-supplied information. The Figure 9 shows the word cloud by adopting fuzzy rough feature selection. The work [9] analyzed drug review dataset [7] and other online reviews. Their objective is to improve accessability of relevant information via modern social media. They presented a deep learning strategy based on the idea of a tweet's footprint to improve search and navigation in social media platforms and an efficient searching algorithm. Moreover, they proposed a deep search strategy that establishes a relationship between the status of a tweet and its longevity measured in terms of engagement lifespan since posting. Deep search utilised a convolutional neural network for textual n-grams features extraction and meta-features from the tweet to train a fully connected network on a vast number of tweets.

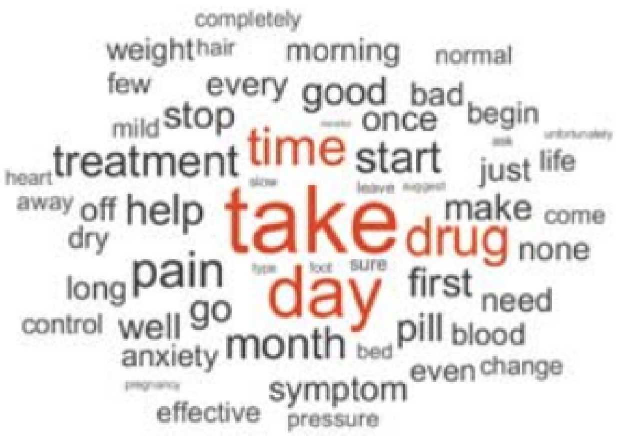

Figure 9: Word cloud by adopting fuzzy rough feature selection [4].

Similar Dataset

The work [6] crawled drugs from the same website *Drugs.com* and other drug website *Druglib.com*. They created a data-mining model to evaluate the effectiveness and detected potential side effects from online customer reviews on specific prescriptive drugs. The study utilized text parsing, text filtering, text topic, and text clustering and supervised learning algorithm for building multiple predictive models to identify the optimal model for reviews. K et al. [11] collected review data from two different website *livewell.pk* and *kaymu.pk* [1]. Their research aimed to use lexicon based approach for sentiment classification according to positive, negative or neutral type of polarity.

Summary

Most of works in the same dataset and similar dataset focus on sentiment analysis [7][2][4][6][11]. The works [7] and [11] employed sentiment analysis to predict users' satisfactory, while our work would like to predict exact rating based on reviews and other features such as useful count on the reviews. [11] differs from ours in a way that they broke sentences in a review and applied sentiment method to detect sentiment related with the sentences in a review, while we use the whole review text. The work [6] gave a clear sentiment classification word cloud. It seems that their method can preserve more positive and negative words. But since our work and [6] use different dataset, it cannot be a fair comparison.

The state-of-the-art methods currently employed to study this type of data would be deep learning like [9]. A deep learning technique learn categories incrementally through it's hidden layer architecture, defining low-level categories like letters first then little higher level categories like words and then higher level categories like sentences. Because deep learning is beyond the course's scope, we did not try this method.

From the Section 1, we find that TF-IDF and useful count have a great influence on rating, but non of existing works used TF-IDF and useful count to predict users' satisfactory. On the other hand, we find that using mixed n-gram method can improve result a lot, which is the same as the work's [7] finding. We believe that the original work [7] simplified the problem a lot, since they divided all rating into three categories without scientific reason or proof. That's why they can achieve 92.24% accuracy.

5 EXPERIMENTAL RESULT

As mentioned in Section 2 and Section 3, we conduct various experiments of different combination of models and features on the user rating predicting task. In our experiments, we use different *Python* packages as listed.

Table 5: Data Description of [7]

#Train	#Test	#conditions	#drugs	rating	label	%
161297	53766	836	3654	rating ≤ 4	-1	25
				4<rating<7	0	9
				rating≥ 7	1	66

- *sklearn.linear_model.LinearRegression* is used to train a simple linear regression model [12].
- *sklearn.linear_model.Ridge* is used to train a regularized linear regression model [12].
- *sklearn.ensemble.RandomForestRegressor* is used to train a random forest regression model [12].
- *xgboost.XGBRegressor* is used to train a gradient boosting model [3].

Table 6 shows out experimental results of various models with different combination of features. As our surmise in Section 3, the result of random forest regression model outperforms the linear and ridge regression models, because the dataset does not contain obvious linear relationships. Furthermore, random forest model also beat the gradient boosting model in terms of the solution quality, which is consistent with our guessing in Section 3. Here, we think the reason is that because of the limitation of computing time and computing resources, we did not well-tune the gradient boosting model by parameters like learning rates and tree-depth. In addition to directly comparison on the solution quality between models, we also perform numerous experiments within each models. The interesting findings and the further improvements are mentioned as follows:

Interpretation of Parameters in Linear Regression

When training the linear regression model with bag-of-words textual representation, we would like to know the most positive and most negative words (features) to help us check if the model is reasonable. As shown in Figure 10, we can observe that most of the words are medical terminologies, and bi-gram word ’bleed complet’ seems reasonable. Nevertheless, if we look at the words of top-twenty positive coefficients, we can find the bi-grams words like ’clear problem’, ’deal breaker’, ’game change’ and ’life changer’, which are reasonable by common sense. On the other hand, as shown in Figure 11, similarly, there are many medical terminologies, but we can still discover bi-gram words like ’no benefit’ and ’no better’ as expected.

Bag-of-words Versus TF-IDF Using Linear Regression

When training the linear regression model, we are interested in whether different textual representation models would give us different solutions. As shown in Figure 12, we can discover that although the number of textual representation

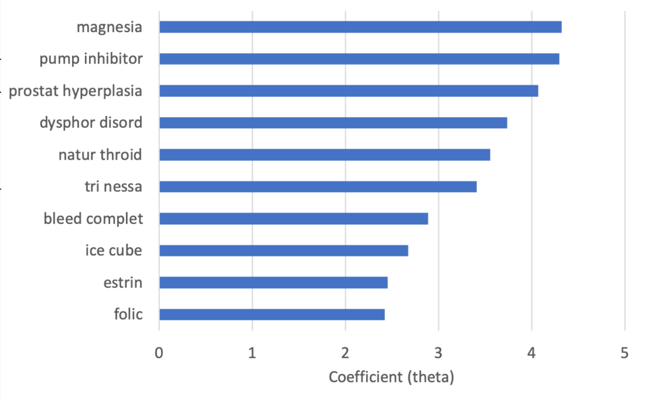

Figure 10: Line chart of top-ten positive coefficients in linear regression with 35000 of bag-of-words features.

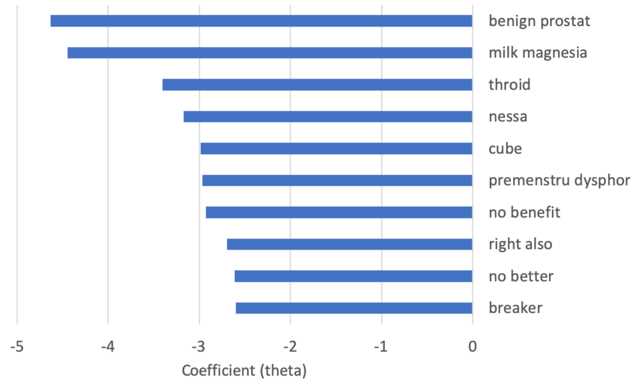

Figure 11: Line chart of top-ten negative coefficients in linear regression with 35000 of bag-of-words features.

features does not help a lot on improving R^2, simply changing from bag-of-words model to TF-IDF model can enhance the solution quality R^2 by about 0.05.

Bi- Versus Tri-gram Model Using Linear Regression

When training the linear regression model, we are also interested in whether different combination of N-gram models can improves the solution quality. As shown in Figure 13, we can observe that the number of TF-IDF features can help enhance the R^2 a little bit but seems converged. However, *Uni- + Bi-gram* and *Uni- + Bi- + Tri-gram* models do not have great differences in terms of the solution quality.

Gradient Boosting and Scalability Issue

As shown in Figure 14, the number of TF-IDF features does not have major effect on the solution quality. Still, the number of trees to fit in the gradient boosting can improve the R^2 gradually by about 0.15. Moreover, based on the trend in

Table 6: Comparison of the various models with different combination of features ("W": number of textual representation features, "C": regularization strength of ridge Regression, "N": number of trees in the random forest/gradient boosting models).

Model	Features	Parameters	*RMSE*	R^2
Baseline #1	Review Length	-	3.2619	0.0018
Baseline #2	Average Rating by Condition	-	2.6149	0.0848
Baseline #3	Bag-of-words (uni-gram)	W = 3000	1.9788	0.4372
Linear Regression	Bag-of-words (uni- & bi-gram)	W = 35000	1.3433	0.5240
	TF-IDF (uni- & bi-gram)	W = 35000	1.3030	0.5724
		W = 45000	1.2957	0.5752
	TF-IDF (uni- & bi- & tri-gram)	W = 35000	1.3025	0.5686
Ridge Regression	TF-IDF (uni- & bi-gram)	W = 35000, C = 0.05	1.4024	0.4582
		W = 35000, C = 0.001	1.5927	0.1684
Gradient Boosting Regressor	TF-IDF (uni- & bi-gram)	W = 5000, N = 100	1.4969	0.3195
		W = 5000, N = 3000	1.3010	0.5750
Random Forest Regressor	TF-IDF (uni- & bi-gram)	W = 5000, N = 100	1.0165	0.7886

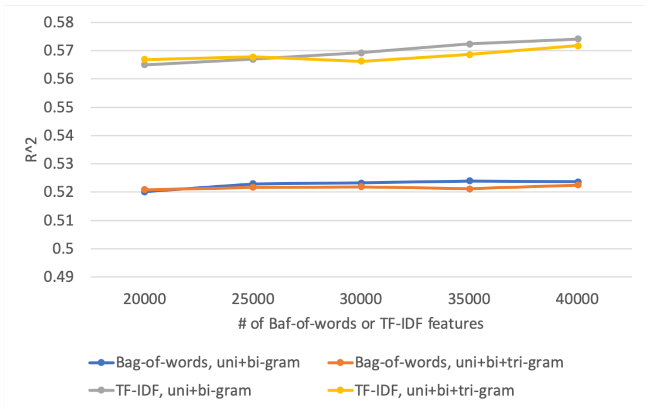

Figure 12: Line chart of experiment on different textual representation models with linear regression.

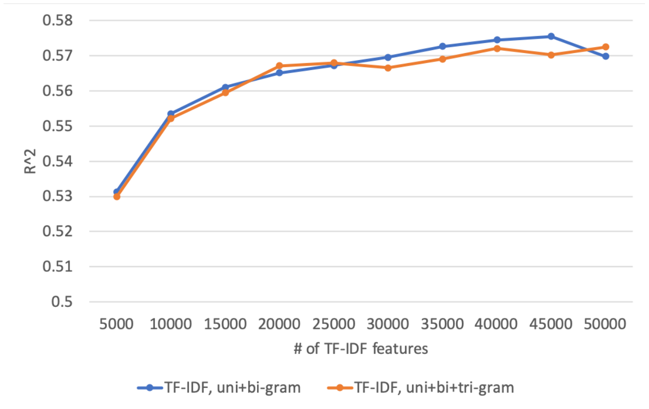

Figure 13: Line chart of experiment on different N-gram models with linear regression.

Figure 14, there is still room for the gradient boosting to improve. In other words, we can keep increasing the number of trees to fit in the model until the convergence if we have more computing resource and computing time.

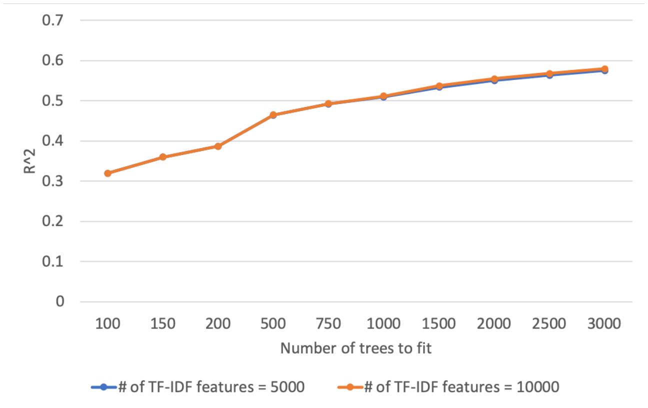

Figure 14: Line chart of gradient boosting experiment.

Random Forest and Feature Importance

As shown in Figure 15, the number of TF-IDF features does not have major effect on the solution quality. Nevertheless, the number of trees in the forest does, which improves the R^2 by about 0.05. Also, Figure 16 and Figure 17 show that the importance of various textual representation features. We can observe that most of the useful features are uni-gram model and have strong sentiment. On the contrary, most of the useless features are bi-gram model and do not contain heavy tendency.

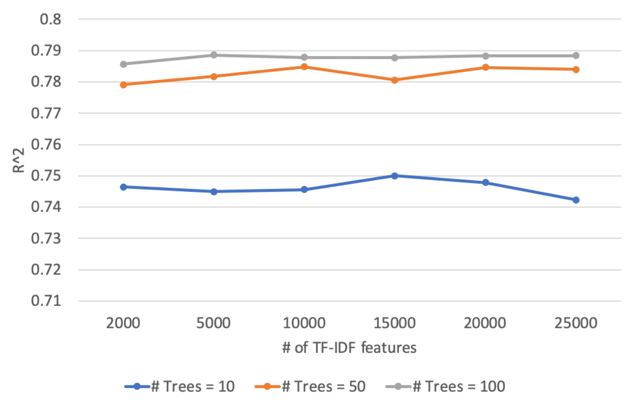

Figure 15: Line chart of random forest experiment.

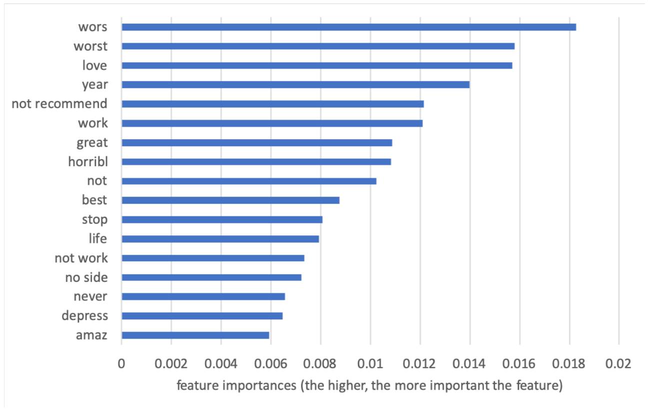

Figure 16: Seventeen most important features in the random forest model.

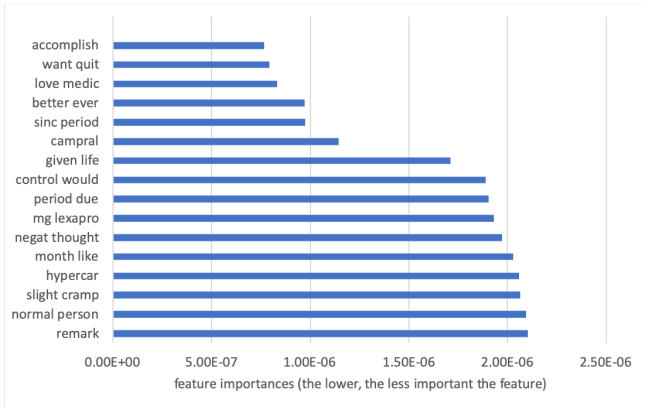

Figure 17: Seventeen least important features in the random forest model.

Predictive Words Using Ridge Regression

Using ridge regression, we analyze the results of the model's parameters, and get the answers of which words are most predictive for low ratings and which words are most predictive

Table 7: Predictive words

Unigram	High Ratings	excellent, years, saved, great, amazing, works, best, miracle, love, i
Unigram	Low Ratings	disappointed, not, worse, this, waste, useless, worst, gained, ruined, horrible
Bigram	High Ratings	feel great, thank you, is amazing, works wonders, love this, words great, really works, highly recommend, the condom, saved my
Bigram	Low Ratings	never again, not recommend, not worth, nothing for, no relief, been bleeding, my vagina, will never, not for, didn't help

for high ratings. The top ten unigrams and bigrams for high ratings and low ratings are shown in the Table 7. These results are consistent with our intuition, and the certain words show that the customers care about drugs' effectiveness (e.g., really works) as well as price (e.g., saved).

Effectiveness of condition and usefulCount Features

Using linear regression model, we compare different features. When taking [bag-of-words, one-hot-date] as feature, we get test R^2 around 0.51. After adding two more features, one-hot-condition and usefulCount, the test R^2 increases to around 0.52, which means adding these two features makes the model slightly more accurate. The coefficient for usefulCount is positive, which means the more usefulCount number a review gets, the higher the rating might be.

Further Improvements

- We believe that employing more sophisticated features and applying more powerful machine learning models, e.g. deep learning approaches, can further improve our current solution.
- In the bag of word model, the words count is dependent with the text length, so maybe it's better to remove the effect of text length by dividing the words count by the text length.

REFERENCES

[1] Minara P Anto, Mejo Antony, KM Muhsina, Nivya Johny, Vinay James, and Aswathy Wilson. 2016. Product rating using sentiment analysis. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*. IEEE, 3458–3462.

[2] Bhargava Apurva. 2019. Grouping of Medicinal Drugs Used for Similar Symptoms by Mining Clusters from Drug Benefits Reviews. *Available at SSRN 3356314* (2019).

[3] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 785–794.

[4] Tianhua Chen, Pan Su, Changjing Shang, Richard Hill, Hengshan Zhang, and Qiang Shen. 2019. Sentiment Classification of Drug Reviews Using Fuzzy-rough Feature Selection. In *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 1–6.

[5] Kerstin Denecke and Yihan Deng. 2015. Sentiment analysis in medical settings: New opportunities and challenges. *Artificial intelligence in medicine* 64, 1 (2015), 17–27.

[6] Thu Dinh. [n.d.]. Detecting Side Effects and Evaluating Effectiveness of Drugs from Customers' Online Reviews using Text Analytics and Data Mining Models. ([n. d.]).

[7] Felix Gräßer, Surya Kallumadi, Hagen Malberg, and Sebastian Zaunseder. 2018. Aspect-based sentiment analysis of drug reviews applying cross-domain and cross-data learning. In *Proceedings of the 2018 International Conference on Digital Health*. ACM, 121–125.

[8] Arthur E Hoerl and Robert W Kennard. 1970. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics* 12, 1 (1970), 55–67.

[9] Isa Inuwa-Dutse, Mark Liptrott, and Ioannis Korkontzelos. 2019. A deep semantic search method for random tweets. *Online Social Networks and Media* 13 (2019), 100046.

[10] Andy Liaw, Matthew Wiener, et al. 2002. Classification and regression by randomForest. *R news* 2, 3 (2002), 18–22.

[11] K Mahboob and F Ali. 2018. Sentiment Analysis of Pharmaceutical Products Evaluation Based on Customer Review Mining. *J Comput Sci Syst Biol* 11 (2018), 190–194.

[12] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.