Subreddit Recommender

Deshantar Parajuli
Texas A&M University
400 Bizzell St, College Station, TX 77843
deshantar99@tamu.edu

Jayesh Tripathi
Texas A&M University
400 Bizzell St
jtjayesh98@tamu.edu

Abstract

Reddit is one of the largest discussion based sites where people of different interests share their thoughts on a common topic. With over 330 million monthly users, and over a million subreddits, it is a daunting experience for a first time visit. We have created a recommendation system which will recommend a subreddit to a user based on their query. This experimental classifier-based recommendation system has been trained with four different classifiers, each displaying its own recommendation to the user. After analysing each algorithm, we have decided that Naive Bayes provides the best result given the evaluation metrics we set for the project.

0.1. Change in Project

Our initial problem statement was to create an algorithm that would help us predict a certain students chance of getting admission in a given university, and while we were looking into which was the right algorithm to use, we ran into the issue of getting the right data. Texas A&M University could not provide us with the data on the right time, due to the ongoing CoViD-19 pandemic, and the data which we finally received was average data of first time students, already admitted at Texas A&M University. Thus, we decided to change our approach and look for datasets that were readily available and could be used to develop working algorithms. We were able to gain access to Reddit's API, which provided us information on all of the posts across many subreddits in real time. We decided on developing an system that would recommend subreddits for reddit users, based on topic based queries.

1. Problem statement

Reddit ranks as the 5th most visited site in United States, boasting over 330 million montly users. This means there are thousands of new users visiting the site everyday. The front page of Reddit contains popular posts made by users from various subreddits in the website. Each subreddit is a unique discussion board based on the certain topic described by the subreddit's name. When a new user observes the thousands of different subreddits circulating through the front page every hour, it can be very intimidating for them to find their own community. Reddit had a feature where it would automatically subscribe new users to a list of popular subreddits, however, in the recent years they have removed this feature, thus leaving the new users on their own to discover their own community. To solve this issue, we have created a classifer-based recommendation system which allows users to construct a topic based query which will be used by the system to display the recommended subreddit.

Traditionally, search engines and recommendation systems aren't built using classifiers because there are an infinite number of queries a user can create, and it's very difficult map each query to a class. However, reddit has a finite number of subreddits, and by carefully selecting a few hundred general subreddits, we can cover a large number of topics a user can be interested in. To train our models, we can use Reddit's API to get post title and its respective subreddit. By using these datapoints, we can train multiple classifiers to predict the subreddit given a post's title. However, this post title prediction is not the goal of our experimental project. Majority of the post titles contain certain terms related to the topic of the subreddit, which can be tracked using a bag-of-words abstract data type which saves each term and its frequency. With this assumption in mind, we hope that given a query that is carefully constructed with keywords pretaining to a certain topic, our model will fetch the subreddit where the frequency of the terms is used the most.

2. Preliminary Literature survey

2.1. Recommender System in Practice

Giving examples of recommendation system used by various companies, like Netflix and Pandora, Houtao Deng provides a clear explanation on the different types of recommendation algorithms and their benefits. He dives into topics such as content-based recommendation, nearest-neighbor, and even deep learning embedding to highlight how companies create a delightful user experience while driving incremental revenue. [1]

2.2. Comprehensive Guide to build a Recommendation Engine from scratch

Pulkit Sharma presents a step-by-step guide for creating a simple recommendation engine from scratch in python. He uses methods such as collaborative filtering models and matrix factorization to create a recommendation system on the MovieLens dataset. He also provides a comprehensive guide on evaluation metrics for recommendation engines by using values such as recall, RMSE, NDCG, and more. This guide allows the developer to capture the the past behavior of a customer and recommend products the users are most likely to buy. [2]

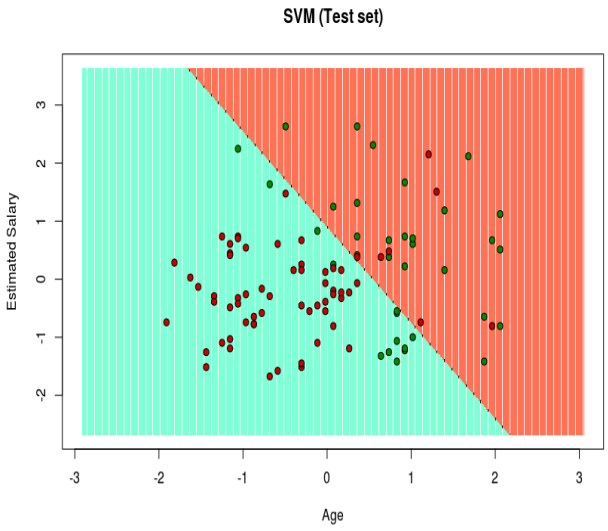

Figure 1. Classification Model

3. Methods

3.1. Data Collection

We obtain the data for the top 1000 posts, from 149 different subreddits, from the Reddit API. Using the scikit-learn's model_selection library, we split the data to 70-30 ratio, for training and testing phases. For good coverage, the subreddits selected are of various topics. The data includes the post titles, the body, url and the subreddit. There were many discrepencies in the dataset, as not all posts included a body and some posts included the metadata for various sorts of attachments. Therefore, we used the post titles, which was a constant for all the entries, to train our model. Data fitting and normalization was handled by the library. We used used nltk library for lemmatizer and stopwords.

3.2. Data Storage and Transformation

When using libraries in scikit-learn, the most important part is the process of transformation and vectorization of the data points. Once all of the steps have been completed, using the library to train the model is a very simple process.

3.2.1 Data Collection from Reddit

First part of the process was to collect the post data points from Reddit. We needed to compile a list of subreddits which would cover a vast majority of the topics frequently discussed in Reddit. To accomplish this, we used "redditlist.com"[6], which keeps an updated list of the most popular subreddits, fastest growing subreddits, and other important metrics that are essential when compiling the subreddit list. We also took inspiration from the old subreddits that Reddit would automatically subscribe new users to. We made sure to not include too many subreddits based on the same topic. For example, we included "soccer", "nfl", "nba", and "sports", but not other specialized sports because we didn't want to get too specific with our lists. However, we included subreddits about soccer, football, and basketball because they are very popular topics and demand their own label in our model, unlike other less popular sports. Our goal is cover as many topics as we can with a few subreddits. We also added a few subreddits that were of interest to us personally, most of which were related to tech and tech career. To increase the coverage, we can add more subreddits to the mix but at the cost of accuracy and precision of the model.

3.2.2 Data Transformation and Vectorization

Once the Reddit data has been downloaded and saved to csv files, the next step is to split the data into train and test sets. To improve the efficiency of the program, we decided to create a separate script to clean the data and save them to train and test csv files. The data is split using scikit-learn's model_selection, which randomly splits a data array into specified sections. For this project, we decided to set aside 70% of the data per file in the train set, and 30% in the test set.

Next, nltk library was used to tokenize each term in the post's title, remove any stopwords, and finally lemmatized

so we could improve the efficiency of our model. To encode the labels in the dataset, we used LabelEncoder module from sklearn, to fit and transform each class to a numeric value. Next, the lemmatized post title terms are fitted and transformed using tf-idf vectorizor (term frequency-inverse document frequency). Now that both data and label are fitted and transformed appropriately, we can now fit them using classification models from scikit-learn and start predicting with our model.

3.3. Naive Bayes Classifier

Naive Bayes classifer is a probabilistic based machine learning algorithm. Based on the Bayesian classifier model, the Naive Bayes classifier treats every event as an independent event and makes it decision on the Maximum A Posteriori (MAP). It has been used as an effective text based classifier, and has been used for spam detection model in GMAIL.

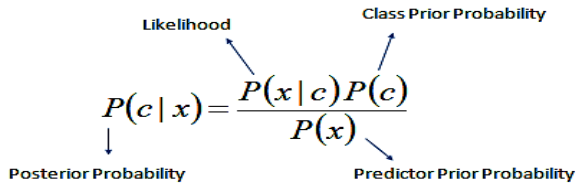

Figure 2. Bayes Model

We are using the Naive Bayes Classifier to classify the bag of words; in our scenario, the post titles, as related to a query or not. Naive Bayes Classifier, would treat every word as an independent event. Thus, for a Naive Bayes Classifier, the appearance of the word sports and soccer are not related. This allows us to treat every word with its independent identity, making the order not necessary. Using this the classifier creates a matrix of words and their appearance in their respective post titles. For the training dataset, we have the post-titles that have already been classified, and this is used to create different probabilities. With the probabilities for the post-titles, based on their association with a certain characteristics, we can easily calculate the prior and the likelihood possibilities for each word, for both the classifications.

Let us say we have the following 3 cases of title, which we have to train, for the sports related searches, with their classification score (for training purposes)

- Messi is the best footall player. (True)
- Messi is a football player. (True)
- Player is a movie. (False)

Messi	is	the	best	player	a	football	movie
1	1	1	1	1	1	0	0
1	1	0	0	1	1	1	0
0	1	0	0	1	1	0	1

With this data, we can create the following data-matrix

For each classification, we have the conditional probabilities, based on the classification of the title. The formula we used to calculate the conditional probability is n+1/k+20, where n is the frequency of the word in that classification, and k is the total number of words in that classification. So for the word,Messi, the conditional probability that it appears in the positive classification is 0.08, and the probability that it appears in the five titles is 0.04. Using this, we can calculate the posterior probability, for the word Messi is 0.0032, for the positive classification. Thus, we will have posterior for every word, and every title will have a posterior value, which is the multiple of all the posterior of the words. Based on the maximum values for a keyword, we suggest the best subreddit to the user.

3.4. Support Vector Machines

With SVM, a machine learning model can have better efficiency in multi-dimensions. SVM, unlike Naive Bayes, is non probabilistic in nature, and works to increase the maximum margins between the edge cases of the two classification groups. SVM works by creating a hyperplane, between the data points, that could clearly and distinctly classify the dataset, in two classification. To identify the best hyperplane, it uses a technique of maximising the margins, between the two sets of data points. By maximising the distance from the nearest points to the hyperplanes, known as support vectors, we can identify the final hyperplane, that divides the dataset into two distinct classifications. One of the biggest advantages comes from the fact that SVM only uses a subset of the whole dataset, the edge cases or known as support vectors. This makes the whole algorithm highly memory efficient, as it does not have to keep a track of data points that are not in the support vector subset. This also helps us in ignoring the outliers, or the extreme cases of certain classification groups. In mathematical models, outliers may have affected the prediction model by increasing the variance, however, since usually outliers are not part of the support vector subset, they dont affect the identification process of the hyperplane, between the two classification groups. Finally, since SVMs are dynamic in nature and depend on reinforcements provided by classification, as it happens, they are unlikely to be prone to misclassification. This increases the efficiency of the model.

”One-versus-all” technique is used to extend a two-class classifier into a multiclass classifier [4]. Each classifier is compared to other classes, this making K-1 SVM classi-

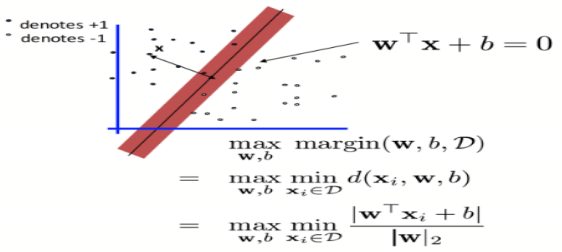

Figure 3. Support Vector Machine

fiers. "One-versus-one" technique is also used to create multiclass classifiers. This is done when each class is compared to all the other classes, using binary SVM classifiers, thus creating K*K-1/2 classifiers. This however leads to an ambiguities, which are demonstrated in the figure below

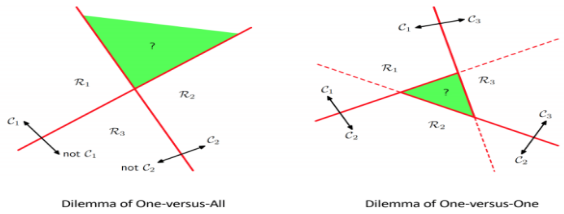

Figure 4. Multiclass classifiers

To resolve the issues, we refered to the scikit-learn's SVM library, and were able to train the model for 60 subreddits for the first iteration, and 149 subreddits for the final iteration.

3.5. Rocchio

Rocchio is a relevance based, feedback driven algorithm. Rocchio works with a dynamic framework, where it relies on the queries by the user, and then creates its own queries, by identifying the relevant and non-relevant data. It uses the training data with the help of weights and the classification, it creates new queries, based on the new dataset. In the end, from an initial query, we end up with a query, from where we are able to rank the data points, which are the nearest to the final query, which are the positive classification and the farthest away are the negative classification. Rocchio is one of the better algorithms for classification of articles and bags of words. A simple query, whose vector representation may be sparse, can be turned into a query set, which may have a multi-dimensional dense dataset.

$$\overrightarrow{Q_m} = \left(a \cdot \overrightarrow{Q_o}\right) + \left(b \cdot \frac{1}{|D_r|} \cdot \sum_{\overrightarrow{D_j} \in D_r} \overrightarrow{D_j}\right) - \left(c \cdot \frac{1}{|D_{nr}|} \cdot \sum_{\overrightarrow{D_k} \in D_{nr}} \overrightarrow{D_k}\right)$$

Figure 5. Rocchio Feedback Formula

For example lets take the example used in Naive Bayes Algorithm, with three titles.

- Messi is the best footall player. (True)
- Messi is a football player. (True)
- Player is a movie. (False)

With this data, we can create the following data-matrix

Messi	is	the	best	player	a	football	movie
1	1	1	1	1	1	0	0
1	1	0	0	1	1	1	0
0	1	0	0	1	1	0	1

Let the positive weight for positive classification of titles be 1, and negative weight for non classification be -0.5. Based on the query soccer, our new query will be

$$Q_1 = [0,0,0,0,1,0,0,0] + \frac{1 * [1,1,1,1,1,1,0,0] + 1 * [1,1,0,0,1,1,1,0]}{2} - 0.5 * [0,1,0,0,0,1,1,1]$$

$$Q_1 = [1,0.5,0.5,1,2,0.5,0,-0.5]$$

With more iterations and a bigger training dataset, we will get a better refined query, through which we can calculate the cosine distance for unclassified titles, and based on the score, we can classify them.

3.6. KNN

KNN does not require a training phase. It is considered as the simplest Supervised Machine Learning Algorithm. For every data point, it looks for the K nearest-neighbors. Based on the data it collects, it decides, whether it belongs to that category or not. This is one of the most efficient and simplest Algorithm for classification. For our model, for every key word, we looked for the cosine similarity between the word in the query and the word frequency in that subreddit. Cosine similarity is calculated by calculating the cosine distance. Two sets of articles may be different in size, but may not be very different on the language used and topic. Cosine distance removes the length of the articles from the equation of comparison, and helps us compare the two articles mathematically. Using the cosine distance, we can classify any new data points by comparing its nearest K neighbours, and thus predict where it is going to lie in the classification. However, despite the simplicity and high accuracy of the algorithm, KNN takes a lot of time, and resources. With a bigger dataset, it is not possible to train and run the KNN, in a given time frame. So, we had to trade off on accuracy, for the time.

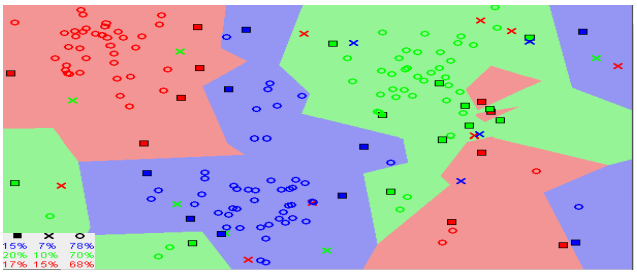

Figure 6. K nearest neighbours

4. Experiments and Results

4.1. Naive Bayes Classifier

In our limited compuatational capacity, the Naive Bayes Classifier worked the best, as a solution. The Naive Bayes Model, use the post-title for prior and likelihood probabilities. For the first iteration, we trained the dataset for 60 subreddits and the results are shown below:

	Train	Test
Accuracy	0.6235813	0.3925642

For the second iteration, we increased the number of subreddits to 149 subreddits, and the test accuracy was actually worse than the results for 60 subreddits. While the model's performance in terms of the metrics decreased, its performance in terms of result coverage increased. This means that the user can now search amoung a larger set of subreddits and get a better set of results as long as their query is constructed properly. This was the original goal of our project. Our goal is not to make an overall search engine for Reddit. The goal our application seeks to achieve is to recommend subreddits given a query with the user's interests. When using this criteria, using 149 subreddits actually performs better than 60 subreddits. Our results for train data performs much better than validation or test. This overfitting is bound to happen because our model is trained based on post title and its respective subreddit. Post titles can contain a lot of filler words (more than our list of stop words can prevent), and not every title is always related to the topics of the subreddit. But our goal is to train the model using post titles to get learn enough keywords on the topic so that it can make good predictions when given a query with topic based keywords. Below is the result for 149 subreddits:

	Train	Test
Accuracy	0.51769143	0.38167495

4.2. SVM

We used scikit-learn's multiclass svm library to train our model with 149 subreddits. It performed very well in terms of accuracy as shown by the table below. While the library didn't provide a clean way to extract other metrics, it can be seen by the accuracy measure that the multiclass svm performed better than naive bayes. Despite this, the model took too long to train. Our computational capacity, especially in the given scenario of CoViD-19 pandemic, also hampered our ability to fully test the limits of an SVM based model. This can be seen by looking at the size of each locally saved trained models. In /local/ directory, each trained model is saved for future use. While all other models get up to 11 MB, SVM is 224 MB. The training time for the model with extremely long (over 1 hr), but with better computing power, we would further explore SVM as it gave very accurate results in both training and testing metrics, and in the evaluation of the way we look to use the trained model.

	Train	Test
Accuracy	0.6006165	0.3678147

4.3. KNN

We decided to decrease the coverage of the project and ran the KNN for a lesser number of subreddits. Despite the time it took, as we did not have a good coverage, the algorithm couldn't pass most of the test cases, as the coverage was low. The low accuracy can be explained because there were not any cases in the near vicinity of our query vector.

	Train	Test
Accuracy	0.33420144	0.12696147

Due to the time KNN models take to predict and the low accuracy, we decided that this algorithm is not suitable for creating a predicition model for our project.

4.4. Rocchio

The results for rocchio were very low, so it is deemed unfit for our recommendation model. The following is the result for 149 subreddits. As you can see, the evaluation metrics are very poor, and it performs poorly when testing with actual user queries.

	Train	Test
Accuracy	0.34598575	0.30180787

5. Result Analysis

Our final model was based on the Naive Bayes classifier algorithm. The reasons behind our choice are the evaluation results, as well as the simplicity of the model. It also gave great results when we made our own realistic queries when testing the application. Despite the high accuracy, we aren't using the SVM because the complexity increases with the

increase in the number of classes. The locally saved model is already 224 Mb for 149 subreddits. Expanding the number of subreddits will drastically increase the complexity. In comparison to others though, KNN gave the worst results, in terms to train and test scores as well as our evaluation using topic based queries. For example, when entering the query "Best movies to watch" (the topic here is movies to watch), NB, SVM, and Rocchio recommend 'movies' subreddit, however, KNN recommends 'Joe Rogan'. Testing the validity of this application is difficult as there isn't one specific answer that is expected for each topic based query. This can be seen with the query "Best tech jobs". NB and SVM recommend "cscareerquestions", which is the best subreddit for tech jobs. However, KNN recommends 'GetEmployed' and Rocchio recommends 'Jobs', which technically aren't wrong. In test and train scores, KNN and Rocchio would guess incorrectly, yet their recommendation is still very relevant to the topic searched. This is one of the issues with this experimental project; it is very difficult to evaluate the model when there can be multiple classes that a given query could potentially belong to. This issue will be furthur discussed in the future works section.

Because Rocchio and KNN perform poorly in train/test scores, NB and SVM are the more trusted models. Yet, we have still decided to display Rocchio and KNN just to display the limitation of those models. Thus the best models to train would be Naive Bayes and SVM. As seen in the results, both Naive Bayes and SVM's training results are a lot higher than their test results, thus showing that the model might be overfitted. However, this is expected because the post titles don't always do the best job at describing the subreddits, yet we still had to use them to train our model.

6. Conclusion

Reddit is one of the most popular websites in the internet, with thousands of new users joining the site every day. When they first land in the frontpage, they may be intimidated by the variety of subreddits. Reddit had a feature to automatically subscribe new users to certain subreddits, however, they have now removed that feature. To solve this issue, we have created an experimental classification based recommendation system.

This recommendation algorithm for Reddit data uses the post titles to train dataset based on different classification algorithms. We used the classification algorithms, for a search-engine like algorithm, because we took advantage of the limited number of subreddits, as compared to the infinite number of queries for a regular search-engine. We are not building a model to predict subreddits given a post title, even though the model is trained to do so. We are using this trained model to predict a subreddit given a topic based query from the user. We hoped that by training the model using post titles, enough keywords about the topic was the subreddit was saved to recommend a subreddit related to the topic given by the user. To train the model, we used the Reddit's API to get the datapoints for a 1000 different posts on 149 different subreddits. Thus, our dataset had 149000 different datapoints. We used 4 different classification algorithm, Naive Bayes Classification algorithm, Support Vector Machine Algorithm, K Nearest Neighbors Algorithm and the Rocchio feedback based classification model. Different algorithms have trade-off for different features, while some have good coverage, others are most efficient. KNN and SVM provided us with the highest accuracy, but as we increased the coverage, their time efficiency decreased. We came to the final conclusion that the classification model best suited was the Naive Bayes Algorithm. This is because of it efficiency when training the model. Given a higher computational power, we would use SVM to train our model.

7. Future Work

Our application performs very well given that we have condensed the thousands of subreddits into 149. In the future, we can increase the number of subreddit to increase the overall coverage of the application. In our application, we have only included the SFW subreddits. While this might be for the best in terms of our application, there are many controversial discussion subreddits that have also been excluded. In the future, we hope try to include majority of the popular topics in reddit in our labels. While we have placed more importance on Naive Bayes because of its simplicity in training, SVM provided us with the best results, so in the future, we would explore SVM even more to make our model as accurate as possible.

One issue we faced during prediction was that a query could potentially relate to multiple subreddits. As discussed in the result analysis, a query about tech jobs should recommend 'cscareerquestions', however, 'jobs' and 'GetEmployed' are still very relevant to the topic of tech jobs. One potential solution could be to group all of the subreddits related to 'jobs' into one class. Thus when a query given is related to jobs, our model could output a list of subreddits that are related to jobs. However, categorizing thousands of subreddits into class can be a difficult task, and other solutions can also be looked at in the future.

Another change we could make to the project is to include the post body along with post titles when training our model. The reason we only included titles was because all post included a title but not every post included a body, so this gave us a way to make all of the data points constant. Another reason was that adding body would make our model extremely complex, adding to an already long training time. Now that our theory of using classifiers as a search engine has had great success, we can expand this experiment by adding post body, and therefore create a more

robost system.

8. Task Assignment

Code: Deshantar Parajuli
Report: Deshantar Parajuli, Jayesh Tripathi

9. Programming Sources

- Python: We will be making our model in Python, due to its simplicity and accessibility, while also having various machine learning libraries.
- Scikit-learn: We used Scikit-learn's multiclass SVM library to train a model

References

[1] Recommender Systems in Practice https://towardsdatascience.com/recommender-systems-in-practice-cef9033bb23a

[2] Machine Learning Basics with the K-Nearest Neighbors Algorithm https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

[3] Comprehensive guide to build a Recommendation Engine from scratch https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/

[4] Introduction to Naive Bayes Classification https://towardsdatascience.com/introduction-to-naive-bayes-classification-4cffabb1ae54

[5] Multiclass SVMs https://nlp.stanford.edu/IR-book/html/htmledition/multiclass-svms-1.html