ReviewIt

Version 1.3

April 13th, 2016

Comp Sci 2XB3 McMaster University

Group 26

John Peng | Nathan Mangaoil | Dragan Visekruna | Nicolas Tristani

Abstract

Due to the advent of social networking, people now have the means to broadcast their views and opinions to far wider audience than ever before. Innovations in mobile technologies in past decade have this even easier. One consequence of this trend is a surge in consumer reviews of commercial products/services, due to web platforms such as Yelp facilitate making this process as seamless as possible. This has very important implications for aspiring businesses, whose' operations are greatly impacted by online reviews. The restaurant business is arguably one of the industries most affected by their customers' reviews. Numerous studies have shown that a Yelp review can significantly impact a consumer's demand for particular restaurant. Thus paying attention to a business's reviews should be a top priority for managers operating in today's food industry.

Introduction

Our project is a simple concept with a complex architecture. We have created a web application that restaurant owners can use to see just how good, or bad their restaurant is doing. The concept behind it is straightforward: a user visits the website, enters his or her name, writes their review, and hits submit.
From here, this is where the magic happens. The user submitted review will be taken in through a server, and fed to our main Java application. Our Java application will accept the review, parse it, split it by words that our algorithm deems useful, and based on all these specifications and more, it will return a score between 1 star, and 5 stars.
The score will be assigned to the review and will be returned to the Manager's Portal webpage, where the owner of the restaurant can see the name of the user, his or her review, and the score that our algorithm has deemed appropriate.

This project greatly revolves around ***Bayes' Theorem.***The simple statement of Bayes' theorem is
shown here:

$$P(Score|Text) = \frac{P(Text|Score) * P(Score)}{P(Text)}$$

Here is how Bayes' Theorem is defined online:

> *"In probability theory and statistics,* ***Bayes' theorem*** *describes the probability of an event, based on conditions that might be related to the event. For example, suppose one is interested in whether a person has cancer, and knows the person's age. If cancer is related to age, then, using Bayes' Theorem, information about their age can be used to more accurately assess the probability that they have cancer."*

Simply put, the theorem takes into account events and outcomes related to whatever is being evaluated, and creates a probability of what it's outcome could be by comparing it to the already evaluated outcomes.

Using this definition, we decided we could implement this into our program. We thought this could be used to make an innovative, more unique way to critic restaurants. And more importantly, help owners improve the quality of their business.

Our idea was to use the amazon review dataset as the conditions Bayes' theorem would base itself on. Then given a review, each word in the review will be check and then run in Bayes' theorem. Some words have a higher probability of being a positive review than others. After each word has been checked, Bayes' theorem and predict a score out of five as to how the customer perceived their experience.

TrainBayes:
Description: This is the main implementation of the Bayesian Prediction algorithm, and relies heavily on the Table class to store relevant information. Include methods to train a Bayesian Classifier, as well as a method to classify a review based on the trained model.

Field	**Modifier**	**Description**
Table table	Private	A Table object that holds the list of words and their associated probabilities used for training the model, as well methods setter/getter methods
ArrayList<Review> reviews	Private	Stores the Review objects used in training the model

Public interfaces

public TrainBayes()
This is the default constructor method, which reads the data from a text file (stored in local memory) and builds a list of Review objects. It will then iterate through each of these Review objects, and add each word to Table, under the respective scores of the Review. For example, when a Review containing "I like fish" with a score of 5 is read, the words "I", "like" and "fish" will be added to their respective word counts in the Table object, under the score of 5.

public int classify(String[] arr, int evidence)
The bread and butter of our application. This method utilizes the famous Bayes Formula classify a review into a score of 1-5. Each of the priors are calculated individually using methods defined in table. Once that's done, we apply Bayes theorem to calculate the probabilities of each review belonging to score of 1-5, find the one with the highest probability, then return that score.

Review

Description: This class has the ADT that will hold the reviews themselves. Each review will five elements: Product ID, Helpfulness, score, summary, and text.

Field	Modifier	Description
String productId	Public	This will hold the product ID of the review
String helpfulness	Public	This holds the helpfulness value of the review
Int score	Public	This holds the score of the review
Int time	Public	This holds the time at which the review was written
String summary	Public	This holds the summary of the review
String text	Public	This holds the text itself from the review

Public Interfaces

public void Review(String id, String help, int score,int time, String summary, String text)
This takes in the inputs required to create an ADT. Each input will be assigned to its respective field. For example, String ID becomes the ProductId of this review ADT.

public Boolean isComplete()
Checks to make sure that the review is valid (has all elements of the ADT).

public void printReview()
Prints out all the elements of the review to the user if so desired.

Notes: Although we have fields defined productID, helpfulness, time, score, summary and text, we actually only use score and text in our application. The reason for this is because the original format of the Amazon review dataset included these fields, so initially we implemented the Review object to be as similar to the actual Review as possible.

ReadFile

Description/: This class is used to simply read the reviews from file, and then assign them to our ADT we created in the Review class.

Field	Modifier	Description
ArrayList<Review> Reviews	Private	This holds the reviews and all its inputs after they've been read.

Public Interfaces

public ReadFile()
This read the file that contains the reviews. After the scanner located the file, It begins to isolate the inputs and puts them in their correct places in the review ADT.

public ArrayList<Review> getReviews()
Simply returns the reviews that have been read.

Table

Description/Traceability: This class is crucial in the implementation of our main class TrainBayes. It keeps track of every single word used in to train the Bayes model, as well as providing methods to quickly/manipulate them.

Field	Modifier	Description
int total	Private	Total number of words processed
HashMap<String,Integer[]> words	Private	String mapped to an integer array that keeps track of how many times the string was repeated in the review
int[] evidence	Private	Holds the count of the total word given scores 1-5 mapped to the index of the array

Public Interfaces

public double probEvent(String word, int score)
This method is used to get the probability of a word, and update its score in the HashMap. It returns a double.

public void updateTotal()
This method is used to update the total number of words processed.

public int total()
This method is used to return the number of total words processed.

public void updateScore(int score)
This method is used to update the count of the words appearing for a particular score, in the evidence array

public void addElement(String word, int score)
This method is used to add a new word to our existing words HashMap with a corresponding score.

public int sumRow(String word)
This method returns the sum of the row where the word is located in the HashMap.

public HashMap<String,Integer[]> table()
This method is used to return the HashMap.

public int sumCol(int score)
This method is used to return the total number of reviews of the given *score*.

public void printTable()
This method is used to print the table, which will be formatted by System.out.println statements. It will also print out the words and their corresponding score from the HashMap.

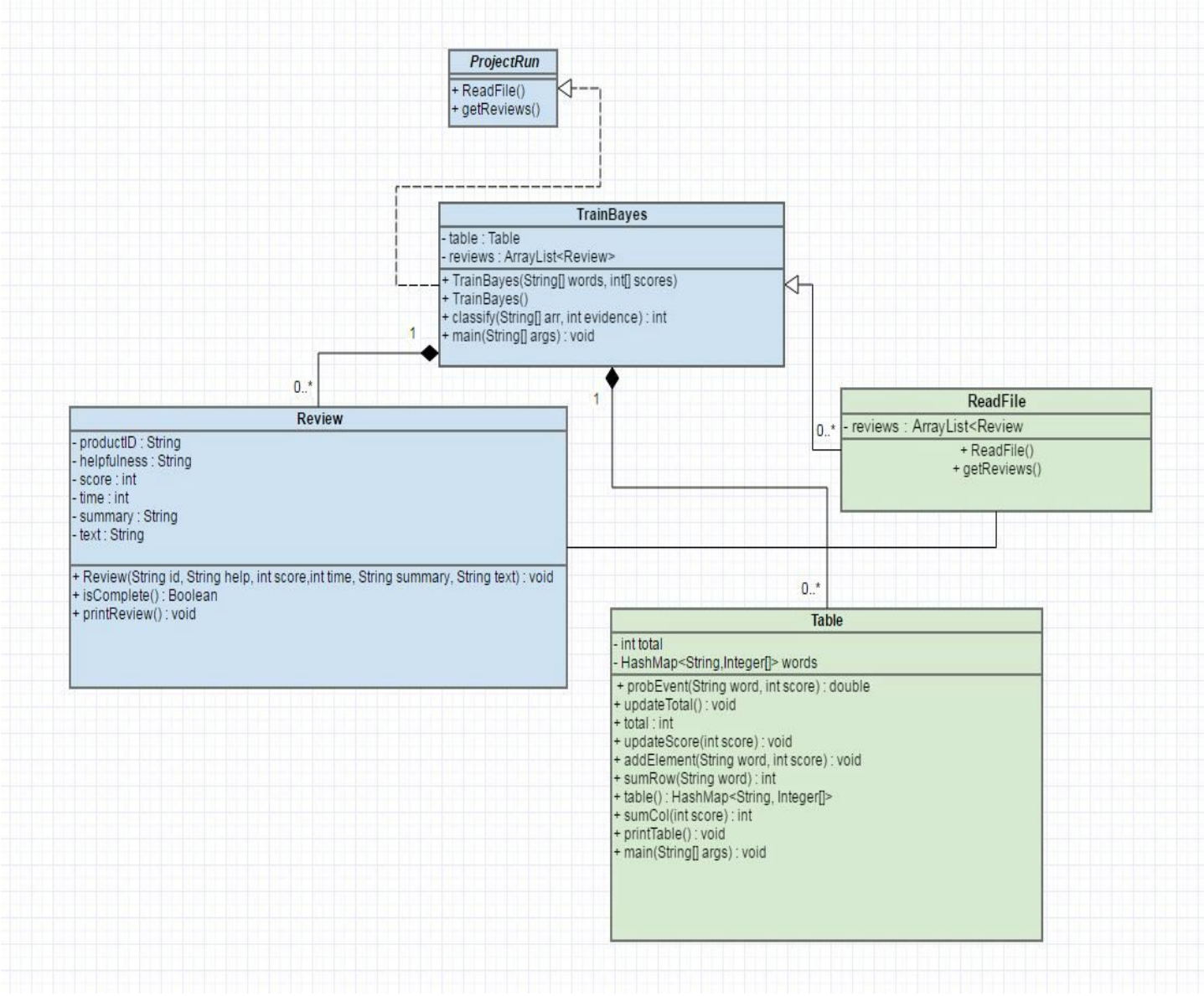
Internal Review:

1) **The dataset:** One of the main things that we realize after we've implemented the Bayes algorithm was that the dataset that we used was actually really poorly suited to our task at hand. The main reason being that the dataset contained reviews regarding very general food items, from diverse categories such as Jumbo peanuts to canned tomatoes. The wide range of different products resulted in reviews that contained a very large vocabulary set. This has 2 main disadvantages: 1) Bayes takes a very long time to iterate through the reviews, due to the large amount of distinct words it has to check against and, 2) many of these words are only relevant with respect to the actual product being reviewed, and cannot be generalized to evaluating restaurants (ie. "jumbo-shrimp", "eskimo", and "Wheatgrass" were just some of these words, which clearly contain no relevance outside of their own respective review category).

2) **Dilution of meaning**:The large dataset also "diluted" the effective weight of each word. For instance, words such as "good" or "terrible", which usually carries a highly negative/positive significance with regards to reviews, have their effects significantly reduced in such a big dataset. This is because when Bayes calculates the probability of a word occurring in a review of a particular score, it aggregates the probability amongst the entire word set, which in this case is very large. Thus, having a dataset with many different words effectively diminishes the effects of the really important ones. We realized this fact too late into our project, and thus was not able to alter implementation to account for this effect. In retrospect, we could 1) included a pre-processing step which filtered out "useless" (in the sense that they don't contribute any information to the negativeness/positiveness of a review) before we trained our model **or/and** 2) artificially inflate the value of those really significant words.

3) **Performance:** We used Java HasMap to map a word to its appearance in reviews from 1-5. One way to improve to performance of algorithm would be to implement a hashing function that hashed keys in ascending lexicographic order, then sort them with MSD. This would have made it a lot faster when checking words in the review against strings in the HashMap. In our current implementation, we just use the built in method, keySet().contains() to accomplish this task. However, since the built in hashing function was not designed to specifically hash strings, the above modification to the Java HashMap would have yielded a performance improvement.

Uses Relationship (whole program)



Traceback to requirements

1: A way to read a Dataset for the program
2: An ADT to hold the Dataset information
3: Class to hold algorithm
4: Class to hold information about words (probability of positive or negative)

Requirement	**Modules**
1	ReadFile.java
2	Review.java
3	TrainBayes.java
4	Table.java

Contributions

Name	**Role**	**Contributions**	**Comments**
John Peng	Lead - Algorithm	Created Bayes' algorithm that works for inputted reviews	Lead the program design
Nathan Mangaoil	Server	Created TomCat server	Linked work to a working server
Dragan Visekruna	Webpage	Created Html files	Created the actual UI the user and manager sees and interact with
Nicolas Tristani	Documentation/ Presentation	Documentation	Analyzed all codes and algorithms