Amazon Food Review

Mithun R , 1MS13CS414,
Department of Computer Science and Engineering,
M.S.R.I.T, Bangalore, India
Ganesh M Hegde,1MS12CS033,
Department of Computer Science and Engineering,
M.S.R.I.T, Bangalore, India

Charan Roa, 1MS13CS406
Department of Computer Science and Engineering,
M.S.R.I.T, Bangalore, India
Karthik M, 1MS13CS407
Department of Computer Science and Engineering,
M.S.R.I.T, Bangalore, India

Abstract: Recommending products to consumers means not only understanding their *tastes*, but also understanding their level of *experience*. Thus our goal in this paper is to recommend products that a user will enjoy *now*,while acknowledging that their tastes may have changed over time, and may change again in the future. We model how tastes change due to the very act of consuming more products. We develop a latent factor recommendation system that explicitly accounts for each user's level of experience. We find that such a model not only leads to better recommendations, but also allows us to study the role of user experience and expertise on a novel data set of fifteen million beer, wine, food reviews.

There are broadly three types of approaches for sentiment classification of texts: (a) using a machine learning based text classifier -such as Naïve Bayes - with suitable feature selection scheme; (b) using the unsupervised semantic orientation scheme of extracting relevant n-grams of the text and then labeling them either as positive or negative and consequentially the document; and (c) using the SentiWordNet based publicly available library that provides positive, negative and neutral scores for words.

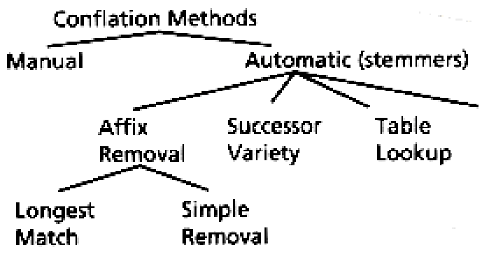
So this analysis is done on the datasets that contains food reviews from the Amazon product dump. This helps the user to better decisions whilst ordering tasted food.

1. INTRODUCTION

In order to predict how a user will respond to a product, we must understand the tastes of the user and the properties of the product. We must also understand how these properties change and evolve over time.This paper highlights three different mechanisms that cause perceptions of products to change. Firstly, such change may be tied to the age of the *product*. Secondly, it may be tied to the age of the *user*. Thirdly, it may be tied to the state of the *community* the user belongs to.Our goal in this paper is to propose models for such mechanisms, in order to assess which of them best captures the temporal dynamics present in real product rating data.However, few works have studied the *personal development* of users, that is, how users' tastes change and evolve as they gain knowledge, maturity, and experience. A user may have to be exposed to many films before they can fully appreciate (by awarding ita high rating) *Citizen Kane*; a user may not appreciate a *Romanée Conti* (the 'Citizen Kane' of red wine) until they have been exposed to many inferior reds;Developing new models that take into account this novel viewpoint of user evolution is one of our main contributions. experience is some quality that users gain over time, as they consume, rate, and review additional products. The underlying hypothesis that we aim to model is that users with similar levels of experience will rate products in similar ways, even if their ratings are temporally far apart. We learn latent-factor recommender systems for different experience levels, so that users 'progress' between recommender systems as they gain experience.

. 2. DESIGN

A. Streaming algorithm



A. Taxonamy of the streaming algorithm

Stemming is used to improve retrieval effectiveness and to reduce the size of indexing files. Several approaches to stemming are described--table lookup, affix removal, successor variety, and n-gram. Empirical studies of stemming are summarized. One technique for improving IR performance is to provide searchers with ways of finding morphological variants of search terms. If, for example, a searcher enters the term *stemming* as part of a query, it is likely that he or she will also be interested in such variants as *stemmed* and *stem*. We use the term *conflation*, meaning the act of fusing or combining, as the general term for the process of matching morphological term variants.here are four automatic approaches. Affix removal algorithms remove suffixes and/or prefixes from terms leaving a *stem*. These algorithms sometimes also transform the resultant stem. The name stemmer derives from this method, which is the most common.

1. Using the *cutoff method*, some cutoff value is selected for successor varieties and a boundary is identified whenever the cutoff value is reached. The problem with this method is how to select the cutoff value--if it is too small, incorrect cuts will be made; if too large, correct cuts will be missed.

2. With the *peak and plateau method*, a segment break is made after a character whose successor variety exceeds that of the character immediately preceding it and the character immediately following it. This method removes the need for the cutoff value to be selected.

3. In the *complete word method* method, a break is made after a segment if the segment is a complete word in the corpus.

4. The *entropy method* takes advantage of the distribution of successor variety letters. The method works as follows. Let $|D_{\alpha i}|$ be the number of words in a text body beginning with the i length sequence of letters α. Let $|D_{\alpha ij}|$ be the number of words in $D_{\alpha i}$ with the successor j. The probability that a member of $D_{\alpha i}$ has the successor j is given by $\frac{|D_{\alpha ij}|}{|D_{\alpha i}|}$. The entropy of $|D_{\alpha i}|$ is

$$H_{\alpha i} = \sum_{p=1}^{26} - \frac{|D_{\alpha ij}|}{|D_{\alpha i}|} \cdot \log_2 \frac{|D_{\alpha ij}|}{|D_{\alpha i}|}$$

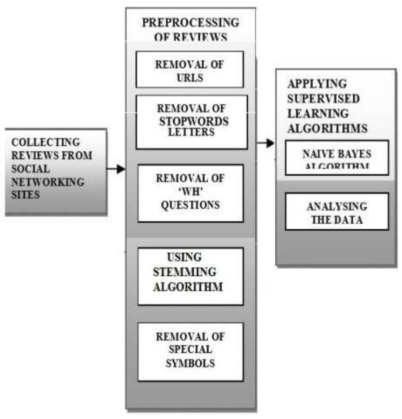
Using this equation, a set of entropy measures can be determined for a word.

Once the unique digrams for the word pair have been identified and counted, a similarity measure based on them is computed. The similarity measure used was Dice's coefficient, which is defined as

$$S = \frac{2C}{A + B}$$

where *A* is the number of unique digrams in the first word, *B* the number of unique digrams in the second, and *C* the number of unique digrams shared by *A* and *B*. For the example above, Dice's coefficient would equal (2 x 6)/(7 + 8) = .80.

B. Naive bayes classifier


COLLECTING REVIEWS FROM SOCIAL NETWORKING SITES
PREPROCESSING OF REVIEWS
REMOVAL OF URLS
REMOVAL OF STOPWORDS LETTERS
REMOVAL OF 'WH' QUESTIONS
USING STEMMING ALGORITHM
REMOVAL OF SPECIAL SYMBOLS
APPLYING SUPERVISED LEARNING ALGORITHMS
NAIVE BAYES ALGORITHM
ANALYSING THE DATA

B. Naïve Bayes Classifier

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

With a multinomial event model, samples (feature vectors) represent the frequencies with which certain events have been generated by a multinomial $(p_1, \dots, p_n)$ where p_i is the probability that event i occurs (or K such multinomials in the multiclass case). A feature vector $\mathbf{x} = (x_1, \dots, x_n)$ is then a histogram, with x_i counting the number of times event i was observed in a particular instance. This is the event model typically used for document classification, with events representing the occurrence of a word in a single document (see bag of words assumption). The likelihood of observing a histogram x is given by

$$p(\mathbf{x}|C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i p_{ki}{}^{x_i}$$

The multinomial naive Bayes classifier becomes a linear classifier when expressed in log-space

$$\begin{aligned} \log p(C_k|\mathbf{x}) &\propto \log \left(p(C_k) \prod_{i=1}^{n} p_{ki}{}^{x_i} \right) \\ &= \log p(C_k) + \sum_{i=1}^{n} x_i \cdot \log p_{ki} \\ &= b + \mathbf{w}_k^\top \mathbf{x} \end{aligned}$$

Where $b = \log p(C_k)$ and $w_{ki} = \log p_{ki}$

If a given class and feature value never occurs together in the training data, then the frequency-based probability estimate will be zero. This is problematic because it will wipe out all information in the other probabilities when they are multiplied. Therefore, it is often desirable to incorporate a small-sample correction, called pseudocount, in all probability estimates such that no probability is ever set to be exactly zero. This way of regularizing naive Bayes is called Laplace smoothing when the pseudocount is one, and Lidstone smoothing in the general case.

Here is a worked example of naive Bayesian classification to the document classification problem. Consider the problem of classifying documents by their content, for example into spam and non-spam e-mails. Imagine that documents are drawn from a number of classes of documents which can be modelled as sets of words where the (independent) probability that the i-th word of a given document occurs in a document from class C can be written as

$$p(w_i|C)$$

Then the probability that a given document D contains all of the words w_i, given a class C, is

$$p(D|C) = \prod_i p(w_i|C)$$

The question that we desire to answer is: "what is the probability that a given document D belongs to a given class C?" In other words, what is $p(C|D)$?

Now by definition

$p(D|C) = \frac{p(D \cap C)}{p(C)}$ and $p(C|D) = \frac{p(D \cap C)}{p(D)}$

Bayes' theorem manipulates these into a statement of probability in terms of likelihood.

$$p(C|D) = \frac{p(C)}{p(D)} p(D|C)$$

Assume for the moment that there are only two mutually exclusive classes, S and ¬S (e.g. positive and negative review), such that every element is in either one or the other;

$p(D|S) = \prod_i p(w_i|S)$ and $p(D|\neg S) = \prod_i p(w_i|\neg S)$

Using the Bayesian result above, we can write:

$p(S|D) = \frac{p(S)}{p(D)} \prod_i p(w_i|S)$ and $p(\neg S|D) = \frac{p(\neg S)}{p(D)} \prod_i p(w_i|\neg S)$

Dividing one by the other gives:

$$\frac{p(S|D)}{p(\neg S|D)} = \frac{p(S)}{p(\neg S)} \prod_i \frac{p(w_i|S)}{p(w_i|\neg S)}$$

Thus, the probability ratio p(S | D) / p(¬S | D) can be expressed in terms of a series of likelihood ratios. The actual probability p(S | D) can be easily computed from log (p(S | D) / p(¬S | D)) based on the observation that p(S | D) + p(¬S | D) = 1.

Taking the logarithm of all these ratios, we have:

$$\ln \frac{p(S|D)}{p(\neg S|D)} = \ln \frac{p(S)}{p(\neg S)} + \sum_i \ln \frac{p(w_i|S)}{p(w_i|\neg S)}$$

Finally, the document can be classified as follows. It is positive review if $p(S|D) > p(\neg S|D)$(i.e., $\ln \frac{p(S|D)}{p(\neg S|D)} > 0$), otherwise it is a negative review.

3. IMPLEMENTATION

There are 3 modules used in this project for the analysis

- Sentiment Analysis module
- Entity Analysis module
- Filter Module

A. Implementation of the Modules. (Modules description)

- Sentiment Analysis Module: **Sentiment analysis** (also known as **opinion mining**) refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials. Sentiment analysis is widely applied to reviews and social media for a variety of applications, ranging from marketing to customer service. Generally speaking, sentiment analysis aims to determine the attitude of a speaker or a writer with respect to some topic or the overall contextual polarity of a document. The attitude may be his or her judgment or evaluation (see appraisal theory), affective state (that is to say, the emotional state of the author when writing), or the intended emotional communication (that is to say, the emotional effect the author wishes to have on the reader).
- Entity Analysis module extracts the attributes i.e positive or negative review using the modules.
- Filter module classifies the entities from the module for the sentiment opinion.

B. Algorithm design

1) Porter's Algorithm (stemming)

Step 1 : Gets rid of plurals and -ed or -ing suffixes.

Step 2 : Turns terminal y to i when there is another vowel in the stem.

Step 3 : Maps double suffixes to single ones: -ization, -ational, etc.

Step 4 : Deals with suffixes, -full, -ness etc.

Step 5 : Takes off -ant, -ence, etc.

Removes a final –e.

2) Naive bayse Algorithm

The Naive Bayes classifier is a simple probabilistic classifier which is based on Bayes theorem with strong and naïve independence assumptions. It is one of the most basic text classification techniques with various applications in email spam detection, personal email sorting, document categorization,Bayes Theorem p(h/D)= p(D/h) P(h)/ p(D)

P(h) : Prior probability of hypothesis h

P(D) : Prior probability of training data D

P(h/D) : Probability of h given D

P(D/h) : Probability of D given h

1. **Handle Data**: Load the data from CSV file and split it into training and test datasets
2. **Summarize Data**: summarize the properties in the training dataset so that we can calculate probabilities and make predictions.
3. **Make a Prediction**: Use the summaries of the dataset to generate a single prediction.

4. **Make Predictions**: Generate predictions given a test dataset and a summarized training dataset.

5. **Evaluate Accuracy**: Evaluate the accuracy of predictions made for a test dataset as the percentage correct out of all predictions made.

6. **Tie it Together**: Use all of the code elements to present a complete and standalone implementation of the Naive Bayes algorithm.

4. SCOPE AND FUTURE WORK

The scope of this project aims to satisfy the user community to make their own analysis of this classified data and also can be used for further research topic.

An interesting finding of our work is that beginners and experts have the same *polarity* in their opinions, but that experts give more 'extreme' ratings: they rate the top products more highly, and the bottom roducts more harshly. Thus naively, we might conclude that we should simply recommend both groups of users the same products: nobody likes adjunct lagers, so what does it matter if beginners dislike them *less*? The counter to this argument is that in order to *fully* appreciate a product (by giving it the highest rating), auser must first become an expert.

5. CONCLUSION

Users' tastes and preferences change and evolve over time. Shifting trends in the community, the arrival of new products, and even changes in users' social networks may influence their rating behavior. At the same time, users' tastes may change simply through the act of consuming additional products, as they gain knowledge and experience. Existing models consider temporal effects at the level of products and communities, but neglect the *personal development* of users: users who rate products at the same time may have less in common than users who rate products at *different* times, but who are at the same stage in their personal evolution. We developed models for such notions of user evolution, in order to assess which best captures the dynamics present in product rating data. We found that modeling users' personal evolution, or 'experience', not only helps us to discover 'acquired tastes' in product rating systems, but more importantly, it allows us to discover *when* users acquire them.

6.REFERENCE

[1] O. Alonso, P. Devanbu, and M. Gertz. Expertise identification and visualization from CVS. In *Working Conference on Mining Software Repositories*, 2008.

[2] R. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *ICDM*, 2007.

[3] J. Bennett and S. Lanning. The Netflix prize. In *KDD Cup and Workshop*, 2007.

[4] L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *SPIRE*, 2000

.