Review-Based Search Engine for Product Recommendation

1. Introduction

This project is a review-based product search engine for recommendations. Instead of using ranking products by their average scores, we use a machine learned positiveness score for reviews as our main ranking factor. Consider the requirement of dataset size, we choose Amazon as our research object.

In this project, count-based Naive Bayes classifier and n-dimensional Gaussian Naive Bayes classifier are being used for review analysis. Base on amazon review dataset and sentiment datasets, we trained machine learning models and use them for the prediction of positiveness score.

To better present our ranking method, we build a search engine base on the titles of products, using DAAT conjunctive to analyze input queries.

1.1 File descriptions

1.1.1 ML

There are four folders under path ./ML.

1.1.1.1 dataset

There are 4 folders under path ./ML/dataset:

(1) emotion: Includes NRC dataset and SentiWord dataset, the most important two datasets we use for sentiment analysis.

(2) productData : The Amazon product dataset, with information for each product. We use this dataset to build our inverted index, lexicon and page table.

(3) productScore : Includes three product score files (positiveness score), **productScore_GA.csv**, **productScore_NB.csv** and **productScore.csv** (which is the merged version of previous two files).

(4) reviewData : Includes two important datasets, one is **kcore_5.json**, which is a review dataset that we apply our NB classifier on. The other is **Reveiew_statistic.txt**, which is a statistic file generate by script **SentiScore.py**, we use a part of this dataset to train and test our n-d Gaussian NB classifier, then apply this classifier on the full statistic dataset to get GA score.

1.1.1.2 N-D Gaussian:

Gaussian.py : Training, testing and applying the learned model on **Reveiew_statistic.txt,** generating predicted scores for all products, store these scores into a new file **productScore_GA.csv**

1.1.1.3 NB

NB.py : Training, testing and apply the learned model on **kcore_5.json**, generating average predicted scores for each product, store them into a new file **productScore_NB.csv**

1.1.1.4 plot

draw.py: Generates Figure 2.1

1.1.2 SearchEngine

There are four folders under path ./SearchEngine.

1.1.2.1 data

Required data files for search engine, includes page table, lexicon, ASCII inverted index and compressed binary inverted index.

1.1.2.2 inverted index

Using the product metadata to generate the ASCII inverted index and page table.

1.1.2.3 compress

Using Simple9 algorithm to compress ASCII inverted index into binary format, generate lexicon file.

1.1.2.4 query

Load page table, lexicon, and score file, analysis query and return a results page for product recommendation.

1.3 How to run

Make sure that these files are under path ./SearchEngine/data/

BinInvertIndex.txt
Lexicon.txt
pgTable.txt

Then use command line or IDE to run *./SearchEngine/query/query.py*

2. Part I: Review-based Machine Learning

2.1 Research

Ranking on E-commerce website

Base on the research of commonly used factors that might affect the rank of products, we found that Amazon could rank their search results by: Title, Brand, Description, Reviews, Sales rank, etc. After trying different queries, we found that the search results are not just affected by the factors listed above. Sometimes advertisements, random order, user-based(cookies) order may affect the rank. These techniques bring merchants more opportunities to sell their products - some low-related items are still possible to appear in the first page. However, this is not always a good news for customers. As customers, people usually want to purchase something when they querying on E-commerce websites (e.g. Amazon), if there are too many low related items in the first few pages, people will feel frustrated.

Customer's choice

On E-commerce website, there are two important statistical attributes that people usually use to estimate the popularity of products, the first one is average stars, another is number of reviews (number of people who purchase this product). Usually, people tend to trust those products which have high values on both of the above two attributes. However, there are exceptions, people might not trust some products with solely several high scored reviews. Normally, the more reviews a product has, the more trust it will earn from people.

2.2 Datasets

2.2.1 Review Dataset

Since we choose Amazon as our research object, we use official Amazon review dataset (http://jmcauley.ucsd.edu/data/amazon/) in our project. Considering the local storage limitation, we choose 5-core dataset(32GB) instead of the full product review dataset. But our method can be set up and apply on both datasets.

There are 9 attributes for each review: *reviewerID, asin, reviewerName, helpful, reviewText, overall, summary, unixReviewTime, reviewTime*. Since we want to find out the relationship between reviews and product's popularity, not all of these attributes are useful for us. After analyzing the meaning of each attribute, we decide to extract *asin, reviewText, overall, summary* from each review's metadata.

Attribute *“overall”* is a score in range from 1 to 5 that directly present customer’s attitude to the product. However, people cannot choose scores other than these 5 integers, which makes each score might be a little fuzzy.
Since this attribute has relative lower dimension, we would like to choose *"overall"* as the label for each review metadata, and use *"reviewText"* and *"summary"* as input data. Our goal is using the review content to find one or more machine learning models which fit the labels.

2.2.2 Sentiment Dataset

In order to analyze the relationship between review content and overall score of positiveness, we need to find some sentiment-related datasets for words which are frequently used in reviews.

Then we found these datasets:

NRC: http://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm

a) NRC-Emotion-Lexicon-Senselevel

- Includes 24206 unigrams with their sentiment labels (fear, anger, anticipate, trust, surprise, sadness, disgust, joy, positive, negative)

b) Amazon-laptops-electronics-reviews-AFFLEX-NEGLEX-unigrams

- 26577 unigrams with their positive/negative scores, and the frequency they appear in positive/negative reviews, extracted from amazon laptop reviews.

c) Amazon-laptops-electronics-reviews-AFFLEX-NEGLEX-bigrams

- 155167 bigrams with their positive/negative scores, and the frequency they appear in positive/negative reviews, extracted from amazon laptop reviews.

FBS: https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#datasets

d) pros-cons review dataset

- A small dataset (3MB) includes short reviews and their sentiment labels (positive/negative).

Sentiwordnet: http://sentiwordnet.isti.cnr.it

e) SentiWord

- 28935 unigrams with their positive and negative scores.

Among these sentiment datasets, we decide to use a) and e) due to the following reasons:

Dataset a) is a comprehensive lexicon, each term is unigram, makes them more easier to be used as input attributes. Among those sentiment labels, we can use label “positive” and “negative” in machine learning algorithm, which simplifies the problem.

Through dataset b) and dataset c) come from Amazon reviews, there are too many dirty data. A large number of “unigrams”/“bigrams” are not English words, some of them are even meaningless. What is more, this dataset is extracted from laptop reviews, which means that some words from other categories’ reviews might not be covered. So, it is hard to use this relative raw dataset to analyze other new reviews.

Through we could use dataset d) to analyze the sentiment of each word, the size of dataset probably will not lead us to an acceptable accuracy. Since we can get sentiment of words directly from dataset a), extra work is unnecessary.

Dataset e) has more details for sentiment analysis. Some words might be used both in positive and negative context, which means those words may have both positive meaning and negative meaning. This dataset scores the positiveness and negativeness for 28935 unigrams, we could use this dataset to do some analysis in a different way.

2.3 Experiments

Our goal is to predict scores for reviews, we use the content of review and summary as our input data, use “overall” as label, which is a integer score in range from 1 to 5. Considering the relative lower dimension of label, we can build a 5-class classifier for prediction.

We did experiments on several machine learning methods, and choose Content-based Naive Bayes Classifier and N-Dimensional-Gaussian Naive Bayes Classifier in our project.

2.3.1 Content-based Naive Bayes Classifier

To use content-based Naive Bayes Classifier, the input value has to be discrete. In our design, we use 0 or 1 to represent whether word *Xn* from lexicon appears in *review i.* The number of input attributes equals to the number of terms in lexicon.

We use dataset a) NRC-Emotion-Lexicon-Senselevel to prepare our lexicon. Since there are 24206 unigrams (some words have duplicates) in the dataset. The first thing we did is to extract the information we need. So we wrote a script to extract the positive/negative label for all unique words, store them into a new file “NRClexicon.txt”. After extracting, we have 8022 unique words.

Using all of these terms could be time-consuming and may cause overfitting problem, so we to reduce the dimension. Considering the most important sentiment words are adjectives and adverbs, with the help of adj/adv dataset, we extract all of advs and adjs from the original NRClexicon.txt file, get 2776 terms.

After the lexicon dataset is prepared, we start to build the NB Classifier. For each review metadata, we extract the content of “reviewText” and “summary”, compare the content with terms in lexicon, get an list of 0/1 (not appear/appear) as input data. Then we use train dataset to count the number of each class (1star - 5star), and the number of each term from lexicon appears in each class.

$$P(Ci|X) = \frac{P(X|Ci) * P(Ci)}{p(x)}$$

Using the matrix we got from previous step, we can calculate the prior $P(Ci)$ for each class, and likelihood $P(X|Ci)$ for each $term|Class$.

For some short or non-english reviews, all of words in the review content may not exist in lexicon. Since 0 may cause a lot of problems, we use add-m smoothing to prevent 0 value.

$$\frac{(number\ of\ examples\ in\ Class\ j\ with\ x_n = v) + m}{(number\ of\ examples\ in\ Class\ j) + mt}$$

During experiment, we found that a lot of valuable adjectives and adverbs are next to punctuation marks, only using space to split the content may lose many useful data. So we extract the words from those strings next to punctuation marks before comparing them with lexicon.

Base on the Bayes formula above, when we have a new review, calculate the posterior $P(Ci|X)$ for each class, then choose the class with the highest probability as our predicted result. To prevent underflow problem, we use log trick instead of the original formula.

$$\prod P(x_n|Ci) * P(Ci) = log(\sum P(x_n|Ci)) + log(P(Ci))$$

2.3.2 Parameters for Content-based Naive Bayes Classifier

a) Add-m smoothing:

Usually, the default m is 0.1, after we tried different values 1, 0.1, 0.01, 0.001..., we found that when m is smaller than 0.01, the accuracy almost converge. To prevent overfitting, we choose m = 0.01 in our model.

b) hard-margin / soft-margin

For each review metadata, the "overall" score is a integer in range from 1 to 5. Sometimes people may randomly choose 4 or 5 when they feel satisfied with a product (because there is no 4.2 or 4.7 to choose). In another word, though we choose to use the "overall" as the label, the score itself sometimes cannot reflect people's attitude precisely. For example, two people may give different scores even if they both like the product. To solve this problem and better check the performance of our learning model, we introduce soft-margin in our testing process by comparing whether the predicted result is in range $[overall - 1, overall + 1]$. For example, predicted result 2, 3, 4 are all acceptable for label 3. Using soft-margin, we can check how precisely the model can predict people's attitudes for products.

We tried different combinations during testing. If we use hard-margin (check if predicted result equals the overall score), we have around 70% accuracy when testing on training set, and 55% on test set. When we use soft-margin, we have around 90% accuracy on training set and 82% on test set.

Using soft-margin validation, we have a relative high accuracy, which means that this machine learning model can be used to predict attitude by analyzing the review content.
(All accuracies below are results of soft-margin validation by default)

c) Training set size:

In machine learning problem, the size of training set is important. Small size will lead to a bad model while large size may cause overfitting. In order to select a proper size for training set, we use a same group of 3000 reviews as our test set, compare the accuracy when using training set with different size.

After trying different size of training set, we found that when size is larger than 8000, the accuracy starts to converge. Considering the relative high dimension of

input attributes (intuitively higher dimension may need more data to train a good model), we choose 10000 reviews as our training dataset.

d) Bayes formula:

Since there are 2776 words in lexicon, the scale of likelihood $P(x_n \mid Ci)$ may be much lower than the scale of prior $P(Ci)$. For scaling, we add a parameter *m (0<m<1)* before prior.

$$log(\sum \quad P(x_n \mid Ci)) \; + \; m * log(P(Ci))$$

After trying different values, we choose m = 0.65 in our final model.

e) Lexicon size:

Lexicon size may also affect the accuracy. Using the lexicon 1.0 (2776 terms), we have 89.27% accuracy on size 10000 training set, and 82.3% on test set.

We also tried the original lexicon (8022 terms), gets an 90.33% accuracy on training set and 80.1% on test set. Obviously, the increasing of dimension leads to an overfitting problem.

Then we tried to reduce dimension. We build another filter to extract top *k* terms from lexicon 1.0 base on their frequencies in this 10000 reviews. After trying several different k (size of top words), we found k=1000 has a better performance - a 83.5% accuracy on test set.

So we choose to use this new lexicon 2.0 (1000 terms) in our model instead of lexicon 1.0 (2776 terms). There is another benefit to choose lower dimension lexicon - it is less time consuming.

2.3.3 N-Dimensional-Gaussian Naive Bayes Classifier

Different from the above method, N-Dimensional-Gaussian Naive Bayes Classifier has better time performance. However, the input data has to be real-valued and fit-gaussian distribution.

In order to get the input data, we use dataset e) SentiWord and the whole review metadata to generate a statistical dataset “Reveiew_statistic.txt”. For each line in this statistical dataset, there are 9 attributes:

1) asin: product id.

2) avgStar: The average “overall” score for each product, this attributes can be used as label in classify problem.

3) Summary_PosScore: The average positive score of summary content for each product. $(total\ positive\ score\ for\ summaries\ /\ number\ of\ sentiment\ words\ in\ summary)$

4) Summary_NegScore: The average negative score of summary content for each product. $(total\ negative\ score\ for\ summaries\ /\ number\ of\ sentiment\ words\ in\ summary)$

5) SummAvgLen: Average length of summaries for each product.

6) Review_PosScore: Average positive score of reviews for each product.

7) Review_NegScore: Average negative score of reviews for each product.

8) Review_AvgLen: Average length of reviews for each product.

9) reviewCount: Number of reviews for each product.

(here we count and calculate average score for summary and review separately because we think due to the different function of review and summary, they might have different weights in mathematical model)

Intuitively, the increasing of Summary_PosScore and Review_PosScore should be accompanied with the growth of avgStar. However, after some experiments, we found that it is not a linear functional relationship.

With the assumption of N-Dimensional-Gaussian relationship, we tried several different groups of attributes to fit N-Dimensional-Gaussian distribution, and compare their accuracies on test set.

In the experiment, we use the round value of avgStar as the label, different groups of attributes as input data to learn the sample mean matrix and covariance matrix for each class (1 to 5). With these two matrices, we can calculate likelihood for predictions. Based on Bayes formula, we should calculate the likelihood $P(x_n\ |Ci)$ separately for 5 classes, and multiply them by their corresponding prior $P(Ci)$ to get the value of $P(Ci|X)\ *\ P(Ci)$. Since $p(x)$ is same for different classes, we will choose the class with the highest $P(Ci|X)\ *\ P(Ci)$ as our predicted result.

In this classifier, we also choose soft-margin verification to estimate the accuracy, which is to compare the predicted class with the *floor(avgStar)* and *ceil(avgStar)*, the result is considered to be right if one of them is matched.

After experiments, we found that counterintuitively, the ReviewAvgLen, SummaryAvgLen and reviewCount don't have much relationship with the avgStar. When we tried to add these attributes to fit Gaussian, the accuracy is reduced. When we choose Summary_PosScore, Summary_NegScore, Review_PosScore, Review_NegScore to fit gaussian, the model has an acceptable accuracy, around 85%.

Then we try to reduce dimension by adding Summary_PosScore with Review_PosScore, and adding Summary_NegScore with Review_NegScore, using these two results to fit 2-D Gaussian. This operation increases the accuracy on test set from 85% to 87.35%. So we choose 2-D Gaussian in our final model.

Similarly, we also tried different size of training set for better performance, and choose a dataset with size 12752 for training in our final model.

2.3.4 Other experiments

In "Reveiew_statistic.txt", we have 4 real-valued attributes that can be used as input. As an assumption, we may use k-Nearest Neighbors model as the classifier.
To validate this assumption, we wrote a script to present the position (coordinate) and class for each sample.

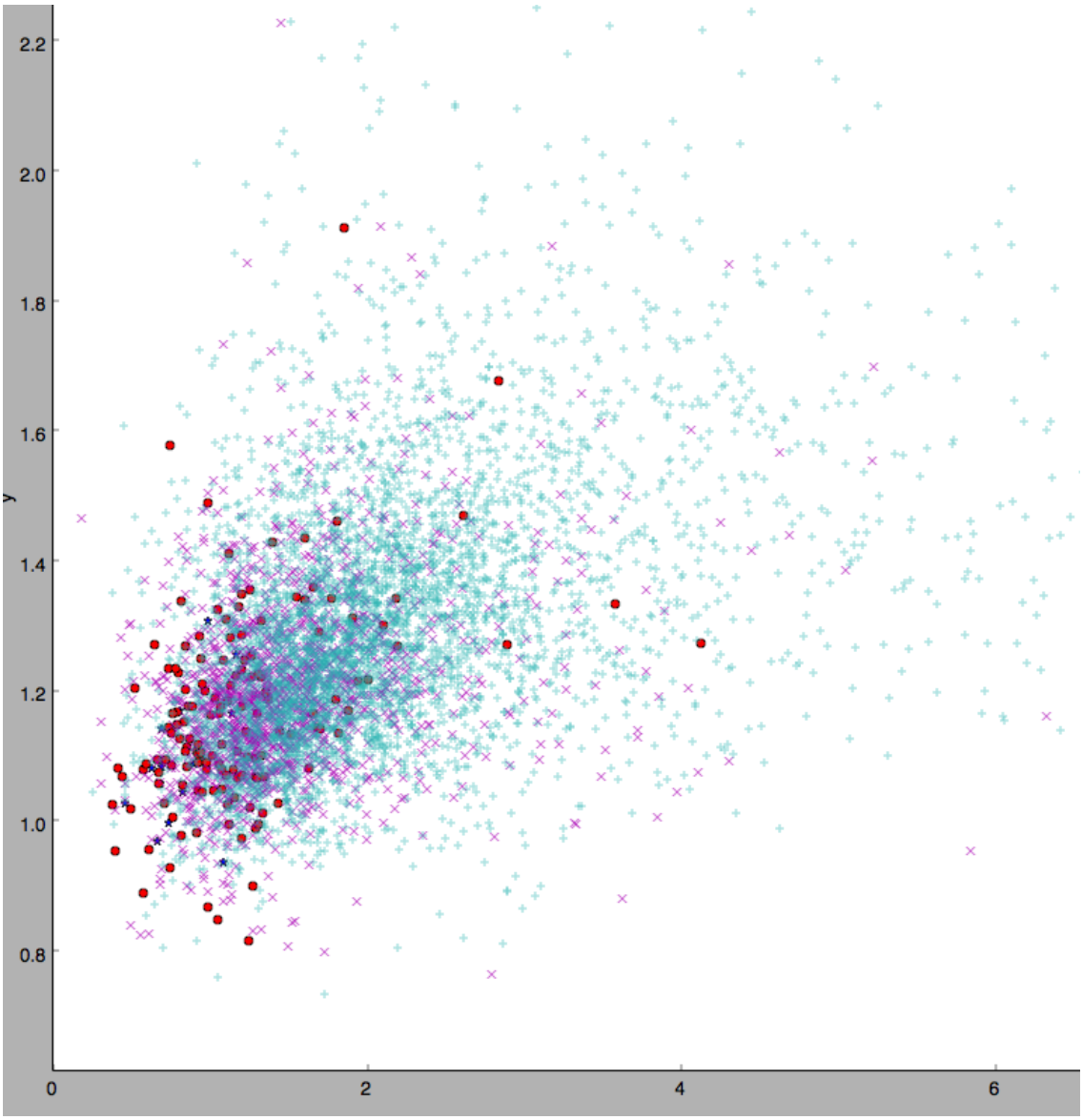

Figure 2.1 class distribution

In Figure 2.1, the green '+' represents 5 star, pink 'x' represents 4 star, red 'o' represents 3 star, and blue '*' represents 2 star.

As we can see in this diagram, there is a general rule among these samples, samples with same label are almost clustered. However, the imbalance among the quantities of classes and the large overlapping areas between each two clusters makes it hard to use k-NN algorithm.

2.4 Implementation

2.4.1 Prediction

After experiments, we built two machine learning models to predict the positiveness score for reviews.

We apply Content-based Naive Bayes Classifier on the original review dataset “kcore_5.json”, predict a score for each review, and store the average score for each product into a file “productScore_NB.csv”. There are three columns in this output file: asin, average score, review numbers.

Because running this algorithm is very time consuming, we split the dataset into 6 parts and run them separately, then merge 6 output files into one.

Similarly, we apply the trained 2-Dimension Gaussian model on dataset “Reveiew_statistic.txt”, generate a file “productScore_GA.csv”. There are four columns in this file: asin, predicted score (integer), the original average score for each product (will be showed in search result), number of reviews.

2.4.2 Product Score file

Base on the result of our experiments, these two models both have an acceptable accuracy. However, for each product, the predicted result in the second model is integer, making it hard to be used for ranking.

To solve this dilemma, in our ranking function, we decide to use both of these two scores which are generated from different models. There are two benefits: because these two scores are from different models, using both of them may have a better performance on prediction. Also, it will reduce the possibility that different products have a same score.

Then we wrote a script to merge these two files into one - “productScore.csv”. There are 5 columns in this file:

1) asin: product id.
2) NBscore: The average predicted score for each product.
3) GAscore: The predicted score for each product.
4) Original average score: The original average score for each product.
5) Review count: Number of reviews for each product.

Among these attributes, 2) NBscore, 3) GAscore and 5) Review count will be used in ranking function. 4) Original average score will be showed in the results page, in order to compare with the machine learned result.

3. Part II: Product Search Engine

3.1 Inverted Index & Page table

The first step to build a search engine is building inverted index and page table. In our project, we use dataset "http://jmcauley.ucsd.edu/data/amazon/" as the source of our product data.

For each product, there are 5 important attributes: 'asin', 'title', 'categories', 'description', 'brand'. In order to spend less time on the index building, we choose the most important attribute 'title' to build our inverted index.

Usually, in search engine for products, the frequency of terms might be irrelevant to ranking function - the ranking of products should not be decided by the frequency of terms. In our inverted index, we only store the page id (product id) for each term.

Our first step is traversing the product dataset, generate the ASCII inverted index file "InvertIndex.txt" and page table file "pgTable.txt".

The format of "InvertIndex.txt" is: ***term: docid1, docid2, ...\n***
The format of "pgTable.txt" is: ***docid, asin, title \n***

3.2 Compression & Lexicon

In order to save space and apply DAAT during query, we use Simple9 and block-wise algorithm to compress our ASCII inverted index, and generate a binary inverted index file "BinInvertIndex.txt" and a corresponding lexicon file "Lexicon.txt".

In "Lexicon.txt", we store each term and it's head & tail position in format: ***term: head, tail \n***

3.3 Query

3.3.1 Introduction

In our project, we choose to use conjunctive query instead of disjunctive query. As a search engine for product recommendation, most of people will use it to query those products they are interested in. For example, if the query is "Nikon camera", then they might not want to see Canon products in the results page.

In order to fast lookup important data during query, we load three files into memory when program is started: "Lexicon.txt", "pgTable.txt" and "productScore.csv".

For each query, we will check whether it contains special characters, if not, using a list of lowercase queried terms to apply DAAT conjunctive query and get a list of docids. Then using ranking function to calculate the score for each doc(product), and return the top 20 results.

For each returned product, these information will be showed on results page:

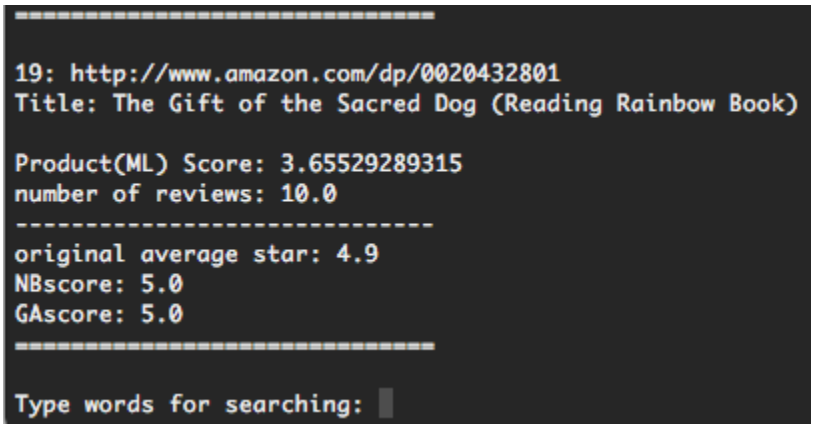

Figure 3.1

product url: The url for product, in format "https://www.amazon.com/dp/" + asin
product title: The title of product, get from page table file.
Product(ML) Score: The final product score calculated by ranking function base on NB score, GA score and number of reviews.
number of reviews: Number of reviews, get from Reveiew_statistic.txt
original average score: Original average star for this product (in the amazon review dataset). This is showed on results page in order to compare with our predicted scores, we didn't use it in prediction.
NB score: The average score(float) predicted by NB classifier.
GA score: The score(int) predicted by Gaussian NB classifier.

3.3.2 Ranking function

We use three attributes (NB score, GA score, number of reviews) as our input variables for ranking function. For each product, NB score is an average score for all predicted score base on each review content. GA score is a predicted score from 1 to 5 using the learned Gaussian model. Though these two machine learning methods have similar accuracies, NB score will become more accurate when number of reviews is relative large. Also, both of GA score and NB score will be less accurate when the number of reviews is relative small.

In order to use both of these two scores for ranking, we give them different weights. After trying different values, we choose 0.8 for NB score and 0.2 for GA score.

$$score1 = 0.8 * NB\ score + 0.2 * GA\ score$$

In our design, the review number is another important factor in ranking. First, the predicted score might be less accurate if there are only few reviews. Second, a small review number usually means that the product is not popular for most of people. For example, we usually prefer to choose a product with 4.3 star and 300 reviews rather than a product with 5 star but only one review.

For the above reasons, products with less review number should be punished. However, we don't want to punish them too much, after all, a product with (5 reviews, 4.2 star) should have higher ranking than a product with (100 reviews, 1.1 star). So we use sigmoid function to calculate punishment.

$$y = \frac{1}{1+e^{-zx}}$$

In sigmoid function, the range of y is $[0.5,1)$ for x in $[0,\infty)$, we may use this feature to punish *score1* base on number of reviews. For an appropriate punishment, we need to choose a proper value for z. After experiments, we decide to use $z = 0.1$.

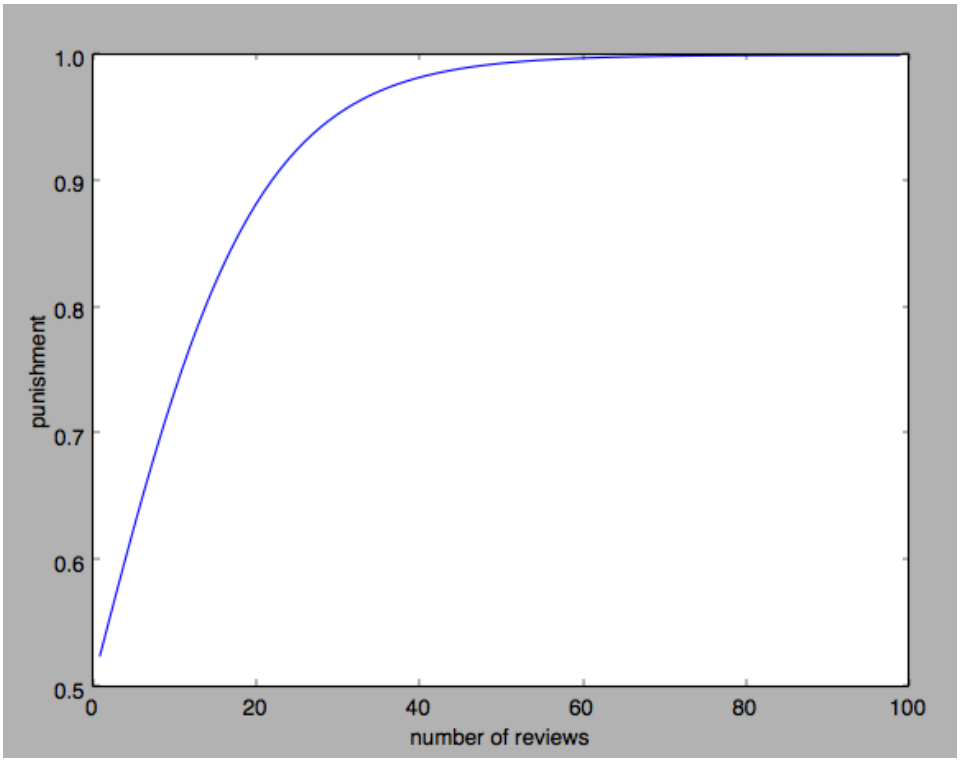

Figure 3.2 sigmoid function for z=0.1

As we can see in Figure 3.2, the less number of reviews, the more punishment. When review number reaches 40, the punishment is 1.8%, when review number reaches 60, there is nearly no punishment. In our ranking algorithm, the formula is:

$$Score2 = (0.2 * GAscore + 0.8 * NBscore) \frac{1}{1 + e^{-0.1 * review\ number}}$$

3.4 Demo

Sample query: book

```
loading lexicon...
loading pagetable...
loading score..
Type words for searching: the book
You use 0.2 seconds.
============RESULT============
0: http://www.amazon.com/dp/0001473123
Title: The Book of Revelation
------------------------------
Product(ML) Score: 4.50221973591
number of reviews: 28.0
------------------------------
original average star: 4.46428571429
NBscore: 4.72
GAscore: 5.0
==============================

1: http://www.amazon.com/dp/000171287X
Title: The Berenstains' B Book (Bright &amp; Early Books)
------------------------------
Product(ML) Score: 4.36260978713
number of reviews: 23.0
------------------------------
original average star: 4.78260869565
NBscore: 5.0
GAscore: 4.0
==============================

2: http://www.amazon.com/dp/0001473727
Title: The Greatest Book on &quot;Dispensational Truth&quot; in the World
------------------------------
Product(ML) Score: 3.44987240564
number of reviews: 8.0
------------------------------
original average star: 5.0
NBscore: 5.0
GAscore: 5.0
==============================

3: http://www.amazon.com/dp/0001472933
Title: The Book of Daniel
------------------------------
Product(ML) Score: 3.22828153113
number of reviews: 6.0
------------------------------
original average star: 4.66666666667
NBscore: 5.0
GAscore: 5.0
==============================

4: http://www.amazon.com/dp/0002246325
Title: The Sandman Book of Dreams
------------------------------
Product(ML) Score: 2.92423431452
number of reviews: 10.0
------------------------------
original average star: 4.2
NBscore: 4.0
GAscore: 4.0
==============================
```

Figure 3.3 results page for "book"

As we can see in Figure 3.3, GA score, NB score and review number will all affect ranking in different degrees.

To better present our project, we made some screenshots for other sample queries, under folder "./demo screenshots".

4. Limitations

Our inverted index and lexicon are built based on titles, which may limit the performance of query, sometimes descriptions may also have important information. For product search engine, a cascading match should be included, for example, we should build two inverted index and lexicon files, one is for title, the other is for description. For each query, we should match title's inverted index first, get top 20 products, then match description's to get another top 10 products. For a fully functional search engine, a disjunctive query processor for categories and brands is also needed.

In our machine learning part, we apply our trained models on k-core (32.7GB, a subset of full review dataset) review dataset for score prediction. However, compared to the Amazon product dataset we use, k-core review dataset covers much less products. As a result, for a relative large number of products, we can’t have predictions. In our project, for those products without predicted scores, we choose not to present them in our results page. This limitation may reduce the number of results in results page for some queries. However, this limitation can be reduced by using the full review dataset.

Reference:

https://moz.com/blog/amazon-seo-organic-search-ranking-factors#factors

http://cs.du.edu/~mitchell/mario_books/Introduction_to_Machine_Learning_-_2e_-_Ethem_Alpaydin.pdf

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html

http://jmcauley.ucsd.edu/data/amazon/

http://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm

https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#datasets

http://sentiwordnet.isti.cnr.it