Abstract— This paper outlines the concepts and technologies of big data, comparing and contrasting with traditional database technologies. It outlines a use case using e-commerce data in which customer reviews from Amazon.com were analyzed for a particular category of products, wearable fitness trackers including Fitbit, Microsoft Band, Jawbone Up, and Garmin Vivofit. Text analytics were performed on the reviews using Apache Pig to see which words are most commonly linked to positive or negative reviews. Also it tested whether the number of stars can be predicted based on the words used in the customer reviews, using the Naïve Bayes algorithm that is implemented in Apache Mahout. All scripts (Python, Pig and Linux shell) used were completed within the Cloudera Quickstart single-node virtual machine.

The results showed that approximately 83% of the time, it can be predicted if a review is positive (defined as four or five stars), and 56% of the time the actual number of stars can be predicted for this set of products. It also showed that the words used in positive and negative reviews can provide some insight into what are important aspects of the user experience with wearable fitness trackers, such as sleep monitoring, voice control, and accuracy of measurement.

Index Terms— Big Data, Hadoop, Mahout, MapReduce, Naïve Bayes, Pig, Predictive Analytics, Text Analytics

I. INTRODUCTION

BIG Data has become one of the most hotly discussed topics in business and information technology, as many organizations have implemented new capabilities to store and analyze vast amounts of data to create greater value. It is understood that Big Data can create new capabilities that can benefit society and create competitive advantage, yet, the subject of Big Data as a whole is not well understood. There are several definitions of Big Data, and there is no official definition, but the most common definition includes the “three Vs”, being volume, variety, velocity. The three V’s are still not agreed upon since there is no official measure of high and low volumes, variety and/or velocity, and some vendors include other V’s such as variability or veracity. A more practical definition is any data storage and analytical system that requires new approaches and technologies, beyond traditional structured query language (SQL) databases and file systems in order to handle the high rate of data accumulation, large amount of data, typically measured in terabytes or petabytes, and lack of consistent data structures or schemas. Traditional database systems, which are the cornerstone of online transactional processing (OLTP) are best suited to read-write operations, and are based on pre-defined table schemas. They also operate serially; in general, one compute job must be completed before the next one starts, and if a job is too large for the system’s hardware configuration, then it can take too long for a process to complete, and also increase the risk failing when performing a large analytical task. The benefit of Big Data systems is that they are able to break up the storage and compute tasks across multiple nodes within a computing cluster using a paradigm called MapReduce.

MapReduce, as implied by the name, maps out the storage and compute task across the nodes redundantly into smaller jobs, then combines (or reduces) the outputs. With this redundancy, risk of failure is reduced significantly, since it assumes that nodes will fail occasionally, yet will not cause the entire job to fail. Also, this reduces the hardware costs since it can be deployed on commodity hardware, as compared to the exponential increase in cost for mainframe computers as computational requirements increase. The Apache project is comprised of the four foundational components:
The Apache Project also includes a number of other Hadoop-related projects. The following are some popular modules that increase the efficiency and effectiveness of data analytics, mining and querying.

- Pig: a higher-level language for parallel computation
- Hive: a data warehouse infrastructure built on top of Hadoop for querying and analyzing data using structured query language (SQL).
- Spark: a fast computation engine for Hadoop data which supports a range of functions such as extract, transform and load (ETL), streaming data and machine learning.
- Mahout: a machine learning platform leveraging MapReduce and/or Spark.  

Commercial Vendors also provide their own proprietary solutions to enhance the capabilities of their distributions. For example, Cloudera includes Impala which provides a SQL-like interface. Proprietary database vendors like Oracle and Microsoft, plus other open source-databases like MongoDB and MySQL provide capabilities with their databases to connect to Hadoop data and utilities.

Hardware is also an important consideration with Big Data and several vendors market proprietary hardware configured specifically for Big Data. Some organizations choose to install “bare metal” physical servers for storage and computation requirements, while others choose to virtualize their Hadoop cluster. Virtualization software companies are partnering with Big Data solution providers to market optimized solutions.

II. **BIG DATA USE CASE: PREDICTING AMAZON REVIEW RATINGS**

This section outlines a specific use case where Amazon product review text was evaluated for a set of similar products (wearable fitness trackers) and a model is developed to predict the number of stars for a particular review based on the words used in the title and text from customer reviews. There are various ways in which this information could be useful for an e-commerce site:

- As a value-added service, feedback could be provided back to manufacturers / marketers so that they are aware of what terms are commonly used in positive and negative reviews of their products, in order to improve products and marketing strategies.
- The site could suggest a number of stars based on their review text.
- The site could prompt the customer to reconsider their review if the text and number of stars are incongruous.
- The site could review existing reviews and find ones that have conflicting star ratings and text, and automatically “downvote” them so that users are more likely to see more consistent reviews – regardless of whether they positive or negative.

**B. Use Case Methodology**

This analysis was performed in multiple stages:

- Data acquisition
- Data preparation
- Data analysis with Pig
- Rating prediction with Mahout

In evaluating the product reviews, two different classification schemes were used, with the first being the number of stars (one through five) and the second being positive (four or five stars) or neutral/negative (one, two or three stars) as it was assumed that customers and marketers alike would prefer to have four or five star products, reflecting a very positive customer experience.

**C. Data Acquisition**

Data had been previously scraped from Amazon.com using the Python `amazon_scraper` library. The dataset included 1,439 reviews for wearable fitness trackers including Microsoft Band, Garmin Vivofit, Fitbit and Jawbone Up. The data was in four separate files, one for each vendor, and the data was in JavaScript Object Notation (JSON) format. Each review included a unique ID field, a stars value represented as a decimal between 0.2 and 1.0, a title, and review body text.
D. Data Preparation

Python was used to prepare data so that it was in a format usable by Pig and Mahout. The libraries NLTK and pandas were used to facilitate data preparation. First, to prepare data for analysis in Pig, it was to be converted into a single file, with tab-delimited values. The title and text were concatenated into one field. Then this field was replicated, excluding stop-words such as “I”, “me”, “it”, and “and”. Second, the data was prepared for analysis in Mahout. The Naive Bayes classifier requires text to be in a specific file and folder structure, with a folder for each category, then each file named with a unique identifier string, and each file containing only the plain text data to be analyzed. This was completed four different ways – with and without stop words, and for two or five categories (good vs. bad, or number of stars). This was performed according to the specifications outlined in the Twenty Newsgroups Classification Example.

The following represents pseudocode for the data preparation in Python:

```python
for file in jsonFilenames:
    reviewData = read_json(file)
    Append reviewData to allReviewData
allReviewData rating = "good" if stars > 0.7, else "bad"
allReviewData stars = stars * 5 # convert stars from decimal to integer
allReviewData titleAndText = concatenate title + text
allReviewData titleAndTextNoStops = remove_stopwords(titleAndText)
export allReviewData to tab delimited text file
# prep data for Mahout
Define function
dumpFilesForMahoutNB(categoryVariable, textVar):
    Folder names = unique categoryVariable
    For folder in folder names: create folder
    For review in review data:
        Write review textVar to file in category folder
    # dump data four different ways
dumpFilesForMahoutNB('rating', 'title_and_text')
dumpFilesForMahoutNB('rating', 'title_and_text_no_stops')
dumpFilesForMahoutNB('stars', 'title_and_text')
(end pseudocode)
```

For actual Python code, see Appendix A.

E. Data Analysis with Pig

Data was analyzed in Pig primarily to determine which words were most common in “good” reviews vs. “bad” reviews. From this, a ratio of the frequencies (good review frequency over bad review frequency) was calculated for terms that appeared in both positive and negative. The average number of stars was also calculated for each product, plus the frequencies for different ratings.

The following represents pseudocode for the data analysis in Pig:

```pig
reviews = load review.tsv;
grouped = group reviews by stars;
    -- freq dist by stars
    starcount = foreach grouped generate group as stars, count(reviews) as count;
    -- avg rating by product
    grpByProduct = group review by product;
    avgRating = foreach grpByProduct generate group and avg(review.stars);
    -- word count
    reviewWords = foreach reviews generate flatten(tokenize(title_and_text_no_stops)) as word;
    wordGrps = group reviewWords by word;
    wordCount = foreach wordGrps generate count(reviewWords) as count, group as word;
    split reviews into goodReview if rating == 'Good', badReviews if rating == 'Bad';
    -- repeat the previous word counting procedure on the split data set (good/bad)
    ...
    wordCountJoined1 = join wordCount by word left outer, goodReviewWordCount by word;
    wordCountJoined2 = join wordCountJoined1 by word left outer, badReviewWordCount by word;
    wordCountJoined3 = foreach wordCountJoined2 generate goodCount / badCount as goodBadRatio;
```

This code produced three output files: the frequency count for number of stars, the average number of stars by product, and a list of words with overall frequency and also frequency for good and bad reviews. For actual Pig code, see Appendix B.

F. Rating Prediction with Mahout

Prediction of number of stars and good/bad rating was achieved using Mahout’s Naive Bayes implementation. In brief, this algorithm uses a sparse matrix where each column is a single word, and each row represents a review, and the matrix includes the frequency of each term in each review. It is referred to as “sparse” since there are many unique terms.
(over 9500 in this data set) so each row of data in the matrix contains mostly zeros. When the data is analyzed across groups, the algorithm determines the frequency of terms with the category and compares it to the overall frequency across all documents. For example, one would expect words like “great”, “excellent”, and “love” to be more common in positive reviews, while terms like “hate”, “terrible”, and “inaccurate” would be more frequent in negative reviews.

As described in the Data Preparation section, the data was put into a specific folder structure for Mahout to analyze. This folder structure was imported to HDFS, then converted into a sequence file. The sequence file was converted into a sparse matrix, weighted using term frequency-inverse document frequency (TF-IDF) vectors.xiv Next, the data was split into training and test sets, which had 70% and 30% of the reviews, respectively. The Naïve Bayes model was trained using the training set, then tested against the test set. The prediction accuracy was then recorded. This training and testing process was completed for each of the four models (with and without stop words, and predicting number of stars or good/bad rating), and was repeated six times for each to ensure that random variation in the train/test sets did not bias the results.

Pseudocode for the Linux shell scripts is as follows, which was repeated six times, with log output saved to individual text files for later analysis:

```bash
save script to mahout_NB.log
MethodNames = "rating_title_and_text/" \ 
"rating_title_and_text_no_stops/" \ 
"stars_title_and_text/" \ 
"stars_title_and_text_no_stops/"
for Methods in MethodNames
  do
    HDFS make directory MethodName
    mahout seqdirectory MethodName
    mahout seq2sparse MethodName -weighting = TF-IDF
    mahout split MethodName -set aside 30% as training set
    mahout train naive bayes model MethodName with training set
    mahout test naive bayes model MethodName with test set
  done
exit
```

Complete shell scripts can be found in Appendix C. The six log files were reviewed afterward and prediction accuracy was recorded into a spreadsheet, and the mean and standard deviation of the test set accuracy was calculated.

G. Results from Data Analysis with Pig

Pig query results indicated that Microsoft Band had the highest average rating (3.99 stars), followed by Garmin Vivofit (3.94 stars), Fitbit (3.86 stars), then Jawbone Up (3.31 stars).

![Figure 2: Average Number of Stars by Product](image)

Across all products, the frequency was calculated for each number of stars. Of the 1439 reviews, 694 had five stars, 272 had four stars, 137 had three stars, 141 had two stars, and 195 had one star.

![Figure 3: Frequency of Number of Stars](image)

“Good” reviews (four or five stars) therefore accounted for 67% of all reviews and there were 2.04 good reviews for each bad review. This ratio was thus used to scale the frequency ratio so that the ratio would not be biased, and a ratio of 1.0 would reflect equal frequency in good and bad reviews. Analysis of word frequencies yielded interesting results. The top five most common words were “band”, “sleep”, “like”, “get”, “one”, “great”, “love”, “fitbit”, and “phone”. “Love” was 4.04 times more frequent and “great” was 1.93 times more frequent in positive reviews. Surprisingly, none of the top ten words referred to fitness or activity, while “sleep” was the second mode common word. The top words that were fitness-related included “heart” (#14), “steps” (#15), and fitness (#26).

![Figure 4: Top 10 Most Common Words in Review Title and Text](image)

<table>
<thead>
<tr>
<th>Word</th>
<th>Overall Frequency</th>
<th>Good Review Frequency</th>
<th>Bad Review Frequency</th>
<th>Good:Bad Frequency Ratio - Adjusted</th>
</tr>
</thead>
<tbody>
<tr>
<td>band</td>
<td>1370</td>
<td>851</td>
<td>519</td>
<td>0.80</td>
</tr>
<tr>
<td>sleep</td>
<td>821</td>
<td>523</td>
<td>298</td>
<td>0.86</td>
</tr>
<tr>
<td>like</td>
<td>761</td>
<td>494</td>
<td>267</td>
<td>0.91</td>
</tr>
<tr>
<td>get</td>
<td>619</td>
<td>392</td>
<td>227</td>
<td>0.85</td>
</tr>
<tr>
<td>one</td>
<td>604</td>
<td>331</td>
<td>273</td>
<td>0.59</td>
</tr>
<tr>
<td>app</td>
<td>602</td>
<td>360</td>
<td>242</td>
<td>0.73</td>
</tr>
<tr>
<td>great</td>
<td>577</td>
<td>460</td>
<td>117</td>
<td>1.93</td>
</tr>
<tr>
<td>love</td>
<td>575</td>
<td>513</td>
<td>62</td>
<td>4.05</td>
</tr>
<tr>
<td>fitbit</td>
<td>567</td>
<td>338</td>
<td>229</td>
<td>0.72</td>
</tr>
<tr>
<td>phone</td>
<td>533</td>
<td>390</td>
<td>143</td>
<td>1.34</td>
</tr>
</tbody>
</table>
Words that had the highest ratio of good to bad review frequency included "fun", "loves", "cortana", "five", "voice", "music", "guided", "texts", "glad" and "starbucks". They were between 7.3 and 14.0 times more common in positive reviews. Positive words like "fun", "loves", "five" (as in "five stars"), and "glad" were expected. "Cortana" (Microsoft’s voice controlled personal assistant), "music", "guided" and "Starbucks" were unexpected and warrant further analysis as to their relationship with these products.

Figure 5: Top 10 Most Common Words in Positive Reviews Title and Text Relative to Negative Reviews

<table>
<thead>
<tr>
<th>Word</th>
<th>Overall Frequency</th>
<th>Good Review Frequency</th>
<th>Bad Review Frequency</th>
<th>Good:Bad Frequency Ratio - Adjusted</th>
</tr>
</thead>
<tbody>
<tr>
<td>fun</td>
<td>59</td>
<td>57</td>
<td>2</td>
<td>13.95</td>
</tr>
<tr>
<td>loves</td>
<td>49</td>
<td>47</td>
<td>2</td>
<td>11.51</td>
</tr>
<tr>
<td>cortana</td>
<td>66</td>
<td>63</td>
<td>3</td>
<td>10.28</td>
</tr>
<tr>
<td>five</td>
<td>216</td>
<td>204</td>
<td>12</td>
<td>8.32</td>
</tr>
<tr>
<td>voice</td>
<td>18</td>
<td>17</td>
<td>1</td>
<td>8.32</td>
</tr>
<tr>
<td>music</td>
<td>18</td>
<td>17</td>
<td>1</td>
<td>8.32</td>
</tr>
<tr>
<td>guided</td>
<td>17</td>
<td>16</td>
<td>1</td>
<td>7.83</td>
</tr>
<tr>
<td>texts</td>
<td>32</td>
<td>30</td>
<td>2</td>
<td>7.34</td>
</tr>
<tr>
<td>glad</td>
<td>32</td>
<td>30</td>
<td>2</td>
<td>7.34</td>
</tr>
<tr>
<td>starbucks</td>
<td>16</td>
<td>15</td>
<td>1</td>
<td>7.34</td>
</tr>
</tbody>
</table>

* See Appendix D for list of 100 words most commonly associated with positive reviews

Words with the smallest ratio of positive to negative frequency included "waste", "disappointing", "returned", "terrible", "inaccurate", "returning", "randomly", "crap", "poorly" and "nap", which were between 20.4 and 59.2 times more common in bad reviews. The only word that was not intuitively negative from this group is “nap” and how the products relate to napping and a negative experience might provide useful information to wearable fitness tracker marketers and manufacturers.

Figure 6: Top 10 Most Common Words in Negative Reviews Title and Text Relative to Positive Reviews

<table>
<thead>
<tr>
<th>Word</th>
<th>Overall Frequency</th>
<th>Good Review Frequency</th>
<th>Bad Review Frequency</th>
<th>Good:Bad Frequency Ratio - Adjusted</th>
</tr>
</thead>
<tbody>
<tr>
<td>waste</td>
<td>30</td>
<td>1</td>
<td>29</td>
<td>0.02</td>
</tr>
<tr>
<td>disappointing</td>
<td>20</td>
<td>1</td>
<td>19</td>
<td>0.03</td>
</tr>
<tr>
<td>returned</td>
<td>43</td>
<td>3</td>
<td>40</td>
<td>0.04</td>
</tr>
<tr>
<td>terrible</td>
<td>26</td>
<td>2</td>
<td>24</td>
<td>0.04</td>
</tr>
<tr>
<td>inaccurate</td>
<td>23</td>
<td>2</td>
<td>21</td>
<td>0.05</td>
</tr>
<tr>
<td>returning</td>
<td>23</td>
<td>2</td>
<td>21</td>
<td>0.05</td>
</tr>
<tr>
<td>randomly</td>
<td>22</td>
<td>2</td>
<td>20</td>
<td>0.05</td>
</tr>
<tr>
<td>crap</td>
<td>11</td>
<td>1</td>
<td>10</td>
<td>0.05</td>
</tr>
<tr>
<td>poorly</td>
<td>11</td>
<td>1</td>
<td>10</td>
<td>0.05</td>
</tr>
<tr>
<td>nap</td>
<td>11</td>
<td>1</td>
<td>10</td>
<td>0.05</td>
</tr>
</tbody>
</table>

* See Appendix E for list of 100 words most commonly associated with negative reviews and Appendix F for top 10 words occurring only in negative reviews.

H. Results from Mahout Naïve Bayes Predictive Model

Predictions using Mahout’s Naïve Bayes algorithm were more successful than anticipated. Predictive accuracy for good/bad rating was around 82.7% on average. For prediction of number of stars, as expected, the accuracy was lower, averaging 56.3% which was still impressive.

The Mahout testnb step produced output as follows for the good/bad classification:

```text
15/12/01 19:47:05 INFO test.TestNaiveBayesDriver: Complementary Results:
```

**Summary**

- Correctly Classified Instances : 369 85.6148%
- Incorrectly Classified Instances : 62 14.3852%
- Total Classified Instances : 431

**Confusion Matrix**

```
a   b  c
111 25 138 a = bad
37 256 293 b = good
```

**Statistics**

- Kappa: 0.6669
- Accuracy: 85.6148%
- Reliability: 56.4187%
- Reliability (standard deviation): 0.4894

15/12/01 19:47:05 INFO driver.MahoutDriver: Program took 12530 ms (Minutes: 0.2088)

The Mahout testnb step produced output as follows for the number of stars classification:

```text
15/12/01 19:47:36 INFO test.TestNaiveBayesDriver: Complementary Results:
```

**Summary**

- Correctly Classified Instances : 237 52.3179%
- Incorrectly Classified Instances : 216 47.6821%
- Total Classified Instances : 453

**Confusion Matrix**

```
a   b   c   d   e
35   8   2   6   6   | 57 a = 1
19 11  6   4   5   | 45 b = 2
11  8  12  9   | 48 c = 3
9 11 14 26 28   | 88 d = 4
8 12 10 28 157 215 e = 5
```

The Mahout testnb step produced output as follows for the number of stars classification:
Naive Bayes using Mahout has been shown to be effective in I. desirable. Yield more consistent predictive performance, which is suggests that this approach (exclusion of stop words) may accuracy was lower when stop words were excluded. This Interestingly, however, the standard deviation of prediction without stop words as indicated by the t-test p values. There was no significant difference in accuracy with or without stop words as indicated by the t-test p values. Interestingly, however, the standard deviation of prediction accuracy was lower when stop words were excluded. This suggests that this approach (exclusion of stop words) may yield more consistent predictable performance, which is desirable.

I. Next Steps in Analysis

Naive Bayes using Mahout has been shown to be effective in analyzing and classifying text at scale. However, this automated classification exercise could potentially be improved and built upon to further refine the modelling approach. The following are some other options to consider:

- Incorrectly categorized reviews could be reviewed in detail to gain insight into why the text and rating do not align which might inform improvements to the model.
- The analysis could be performed on a significantly larger (i.e. 100 or 1,000 times) data set to test scalability and refinement of the model.
- The analysis could be performed on a wider range of products, training and testing other product categories, like books, movies, clothing, health products and different types of electronic goods.

- A simple bag-of-words approach was used. A drawback of this is that negation is missed; for example, “Not great” or “didn’t like it” contain words that are positive but are negated. Different approaches to tokenizing text might be considered to improve accuracy.

- Mahout’s second implementation of Naive Bayes, known as “Transformed Weight-normalized Complement Naive Bayes”, or Cbayes for short, could be attempted to see if predictive accuracy is improved.

III. USE CASE CONCLUSION

This use case demonstrated the effectiveness of the Apache Hadoop big data stack for analysis of customer reviews. Pig is highly efficient at tokenizing text and counting words, while Mahout is able to perform popular machine learning tasks at scale, which can be integrated with production systems. These are powerful capabilities considering the large amount of unstructured data online and within corporate file and document management systems. An e-commerce site such as Amazon.com, which has millions of products, each with new reviews being provided by customers regularly, has the significant challenge of extracting insights from millions of product reviews.

This proof-of-concept has demonstrated the value and capabilities of the Apache Hadoop stack to enable big data analytics at scale. Traditional database systems are limited in their ability to analyze data at scale and in real-time since they generally do not leverage distributed computing, nor do they have machine learning algorithms implemented “out of the box”. The types of text analytics demonstrated here on a small data set could be scaled from thousands to millions of records, if hardware is scaled laterally (i.e. more nodes are added) to enable parallel computing. Hence, product reviews could be analyzed in real time to provide value-added feedback to customers, customer insights to the e-commerce vendor, plus feedback to product manufacturers and marketers. Also, with big data computing on a multi-node environment, models could be refreshed on a regular basis in order to ensure that they are kept current and predictive accuracy does not diminish.

APPENDIX

Appendix A: Python Code for Data Preparation

CIS 436 Final Project
Convert Amazon review JSON data to TSV and data folders for Pig and Mahout
Created on Tue Nov 17 21:21:10 2015
import os
import pandas as pd
import string
from nltk.corpus import stopwords

home_dir = '/home/cloudera/datasets/amazon_reviews'
json_dir = home_dir + '/json'

# load the data into a data frame
for fn in json_filenames:
    print(fn)
    d = pd.read_json(fn)
    print(d.head())
    print(len(d))

# set stars to range 1-5 instead of 0.2-1.0
all_reviews['stars'] = [int(s * 5) for s in all_reviews.stars]

# set binary good/bad variable
all_reviews['rating'] = ["good" if r > 0.7 else "bad" for r in all_reviews.stars]

# combine review title and text

# reset the index so values are unique
all_reviews.index = range(0, len(all_reviews))

# keep only the desired data columns
all_reviews = all_reviews[['product', 'id', 'rating', 'stars', 'title_and_text', 'title', 'text', 'stars']]

# save data to TSV for PIG analysis
all_reviews.to_csv('allReviews.tsv', index=None, header=None, sep='\t', encoding='utf-8')

# make a folder for the data subfolders (for Mahout)
dumpFolder = 'data_dump'

if dumpFolder not in os.listdir(os.getcwd()):
    os.mkdir(dumpFolder)

    os.chdir(dumpFolder)

    def dumpFilesForMahoutNB(categoryVariable, textVariable):
        folderName = str(categoryVariable) + '_' + str(textVariable)
        # make the parent data folder
        os.chdir(folderName)
if folderName not in os.listdir(os.getcwd()):
    os.mkdir(folderName)
    os.chdir(folderName)

# make sub-folders for each category
categories = set(all_reviews[categoryVariable])
category_list = [str(c) for c in categories]
for cat in category_list:
    if cat not in os.listdir(os.getcwd()):
        os.mkdir(cat)

# put review text files into individual folders
for i in all_reviews.index:
    this_review = all_reviews.ix[i]
    thisFileName = str(this_review[categoryVariable]) + '/' +
    this_review['id']
    global f
    f = open(thisFileName, 'w')
    f.write(this_review[textVariable])
    f.close()
    os.chdir('..')

# dump data for each categorization scheme (good/bad or # of stars) and
# for text with and without stopwords
dumpFilesForMahoutNB('rating', 'title_and_text')
dumpFilesForMahoutNB('rating', 'title_and_text_no_stops')
dumpFilesForMahoutNB('stars', 'title_and_text')
dumpFilesForMahoutNB('stars', 'title_and_text_no_stops')

Appendix B: Pig Code for Text Analysis
-- Analysis of Amazon Review - automated scoring

reviews = LOAD './amazon/allReviews.tsv' AS (  
    product: chararray,
    id: chararray,
    rating: chararray,
    stars: float,
    title: chararray,
    text: chararray,
    title_and_text: chararray,
    title_and_text_no_stops: chararray
);
goodReviewWordCount = FOREACH goodReviewWordGroups
   GENERATE
      COUNT(goodReviewWords2) AS goodCt,
      group AS goodWord;

goodReviewOrderedWordCount = ORDER
goodReviewWordCount BY goodCt DESC;
STORE goodReviewOrderedWordCount INTO './amazon/results/good_review_word_ct';
DESCRIBE goodReviewOrderedWordCount;

badReviewWords = FOREACH badReviews GENERATE
   FLATTEN(TOKENIZE(title_and_text_no_stops)) AS word;
badReviewWords2 = FILTER badReviewWords BY word
   MATCHES '\w+';
badReviewWordGroups = GROUP badReviewWords2 BY word;
badReviewWordCount = FOREACH
   badReviewWordGroups GENERATE
      COUNT(badReviewWords2) AS badCt,
      group AS badWord;
badReviewOrderedWordCount = ORDER
   badReviewWordCount BY badCt DESC;
STORE badReviewOrderedWordCount INTO './amazon/results/bad_review_word_ct';
DESCRIBE badReviewOrderedWordCount;

wordCountJoined4 = FILTER wordCountJoined3 BY
   (goodReviewFreq > 0) AND
   (badReviewFreq > 0);
wordCountDescRatio = ORDER wordCountJoined4 BY
goodBadRatio DESC;
STORE wordCountDescRatio into './amazon/results/highly_positive_words_ratio';
STORE wordCountAscRatio into './amazon/results/highly_negative_words_ratio';

wordCountJoined = JOIN word_count by allWord
   LEFT OUTER,
   goodReviewWordCount by goodWord;
wordCountJoined2 = JOIN wordCountJoined by
   allWord LEFT OUTER,
   badReviewWordCount by badWord;
wordCountJoined3 = FOREACH wordCountJoined2
   GENERATE allWord as word,
      allCt as overallFreq,
      goodCt as goodReviewFreq,
      badCt as badReviewFreq,
      (float)goodCt/(float)allCt as goodProp,
      (float)badCt/(float)allCt as badProp,
      (float)goodCt/(float)badCt as goodBadRatio;

wordCountMostCommon = ORDER wordCountJoined3 BY
   overallFreq DESC;
wordCountMostCommonGood = ORDER wordCountJoined3
   BY goodReviewFreq DESC;
wordCountMostCommonBad = ORDER wordCountJoined3 BY
   badReviewFreq DESC;
STORE wordCountMostCommonGood into './amazon/results/highly_positive_words_freq';
STORE wordCountMostCommonBad into './amazon/results/highly_negative_words_freq';

-- shell command to get files from HDFS:
-- hdfs dfs -get /user/cloudera/amazon/results/~/Downloads
Appendix C: Linux shell script for Mahout Naïve Bayes algorithm

# CIS 436 Final Project
# Naïve Bayes classifier for Amazon reviews
# Andrew Pacey

# record the output
script mahout_NB1.log

# create the folders
hdfs dfs -rm -r ./amazon/
hdfs dfs -mkdir ./amazon/

SOURCE_DIR_ROOT=/home/cloudera/datasets/amazon_reviews/data_dump/
WORK_DIR_ROOT=/user/cloudera/amazon/
SUBFOLDERS=("rating_title_and_text/" \
    "rating_title_and_text_no_stops/" \
    "stars_title_and_text/" \
    "stars_title_and_text_no_stops/"
)

# put data into HDFS
for DIR in ${SUBFOLDERS[*]}
do
    echo "============================================================
    echo $DIR
    SOURCE_DIR=$SOURCE_DIR_ROOT$DIR
    WORK_DIR=$WORK_DIR_ROOT$DIR
    mahout seqdirectory \
        -i $WORK_DIR/ \
        -o $WORK_DIR/seq/
    echo "============================================================
    done

# convert seq to sparse matrix
for DIR in ${SUBFOLDERS[*]}
do
    echo "============================================================
    echo $DIR
    SOURCE_DIR=$SOURCE_DIR_ROOT$DIR
    WORK_DIR=$WORK_DIR_ROOT$DIR
    mahout seq2sparse \
        -i $WORK_DIR/seq/ \
        -o $WORK_DIR/vectors/ \
        -lnorm \n        -nv \n        -wt tfidf
    echo "============================================================
    done

# split into training and testing sets (70% train, 30% test)
for DIR in ${SUBFOLDERS[*]}
do
    echo "============================================================
    echo $DIR
    SOURCE_DIR=$SOURCE_DIR_ROOT$DIR
    WORK_DIR=$WORK_DIR_ROOT$DIR
    mahout split -i $WORK_DIR/vectors/tfidf-vectors \
        --trainingOutput $WORK_DIR/train-vectors \
        --testOutput $WORK_DIR/test-vectors \
        --randomSelectionPct 30 \
        --validationOutput $WORK_DIR/validation-vectors \
        --validationSelectionPct 10 "
    echo "============================================================
    done
```bash
--overwrite \
--sequenceFiles \exit
-xm sequential
echo
"=================================================================
===========
" done

# train the NB model
for DIR in ${SUBFOLDERS[*]}
do
echo
"=================================================================
===========
" echo $DIR
SOURCE_DIR=${SOURCE_DIR_ROOT}$DIR
WORK_DIR=${WORK_DIR_ROOT}$DIR
mahout trainnb -i $WORK_DIR/train-vectors \
-elm
-o $WORK_DIR/model \
-li $WORK_DIR/labelindex \
-ow \
-c
echo
"=================================================================
===========
" done

# test model on holdout set
for DIR in ${SUBFOLDERS[*]}
do
echo
"=================================================================
===========
" echo $DIR
SOURCE_DIR=${SOURCE_DIR_ROOT}$DIR
WORK_DIR=${WORK_DIR_ROOT}$DIR
mahout testnb -i $WORK_DIR/test-vectors \
-m $WORK_DIR/model \
-l $WORK_DIR/labelindex \
-ow \
-o $WORK_DIR/testing \
-c
echo
"=================================================================
===========
" done
```
### Appendix D: Top 100 words with Highest Ratio of Good to Bad Frequency in Amazon Reviews

<table>
<thead>
<tr>
<th>Word</th>
<th>Overall Freq</th>
<th>Good Review Freq</th>
<th>Bad Review Freq</th>
<th>Good Bad Freq Ratio</th>
<th>Good Bad Freq Ratio - Adjusted</th>
</tr>
</thead>
<tbody>
<tr>
<td>fun</td>
<td>59</td>
<td>57</td>
<td>2</td>
<td>28.50</td>
<td>13.95</td>
</tr>
<tr>
<td>loves</td>
<td>49</td>
<td>47</td>
<td>2</td>
<td>23.50</td>
<td>11.51</td>
</tr>
<tr>
<td>cortana</td>
<td>66</td>
<td>63</td>
<td>3</td>
<td>21.00</td>
<td>10.28</td>
</tr>
<tr>
<td>five</td>
<td>216</td>
<td>204</td>
<td>12</td>
<td>17.00</td>
<td>8.32</td>
</tr>
<tr>
<td>voice</td>
<td>18</td>
<td>17</td>
<td>1</td>
<td>17.00</td>
<td>8.32</td>
</tr>
<tr>
<td>music</td>
<td>18</td>
<td>17</td>
<td>1</td>
<td>17.00</td>
<td>8.32</td>
</tr>
<tr>
<td>guided</td>
<td>17</td>
<td>16</td>
<td>1</td>
<td>16.00</td>
<td>7.83</td>
</tr>
<tr>
<td>texts</td>
<td>32</td>
<td>30</td>
<td>2</td>
<td>15.00</td>
<td>7.34</td>
</tr>
<tr>
<td>glad</td>
<td>32</td>
<td>30</td>
<td>2</td>
<td>15.00</td>
<td>7.34</td>
</tr>
<tr>
<td>starbucks</td>
<td>16</td>
<td>15</td>
<td>1</td>
<td>15.00</td>
<td>7.34</td>
</tr>
<tr>
<td>background</td>
<td>15</td>
<td>14</td>
<td>1</td>
<td>14.00</td>
<td>6.86</td>
</tr>
<tr>
<td>goal</td>
<td>44</td>
<td>41</td>
<td>3</td>
<td>13.67</td>
<td>6.69</td>
</tr>
<tr>
<td>text</td>
<td>73</td>
<td>68</td>
<td>5</td>
<td>13.60</td>
<td>6.66</td>
</tr>
<tr>
<td>incoming</td>
<td>14</td>
<td>13</td>
<td>1</td>
<td>13.00</td>
<td>6.37</td>
</tr>
<tr>
<td>con</td>
<td>14</td>
<td>13</td>
<td>1</td>
<td>13.00</td>
<td>6.37</td>
</tr>
<tr>
<td>sizing</td>
<td>14</td>
<td>13</td>
<td>1</td>
<td>13.00</td>
<td>6.37</td>
</tr>
<tr>
<td>ill</td>
<td>14</td>
<td>13</td>
<td>1</td>
<td>13.00</td>
<td>6.37</td>
</tr>
<tr>
<td>wonderful</td>
<td>14</td>
<td>13</td>
<td>1</td>
<td>13.00</td>
<td>6.37</td>
</tr>
<tr>
<td>everyday</td>
<td>27</td>
<td>25</td>
<td>2</td>
<td>12.50</td>
<td>6.12</td>
</tr>
<tr>
<td>customize</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>12.00</td>
<td>5.88</td>
</tr>
<tr>
<td>combination</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>12.00</td>
<td>5.88</td>
</tr>
<tr>
<td>enjoy</td>
<td>24</td>
<td>22</td>
<td>2</td>
<td>11.00</td>
<td>5.39</td>
</tr>
<tr>
<td>glance</td>
<td>12</td>
<td>11</td>
<td>1</td>
<td>11.00</td>
<td>5.39</td>
</tr>
<tr>
<td>burn</td>
<td>23</td>
<td>21</td>
<td>2</td>
<td>10.50</td>
<td>5.14</td>
</tr>
<tr>
<td>windows</td>
<td>113</td>
<td>103</td>
<td>10</td>
<td>10.30</td>
<td>5.04</td>
</tr>
<tr>
<td>level</td>
<td>33</td>
<td>30</td>
<td>3</td>
<td>10.00</td>
<td>4.90</td>
</tr>
<tr>
<td>motivator</td>
<td>11</td>
<td>10</td>
<td>1</td>
<td>10.00</td>
<td>4.90</td>
</tr>
<tr>
<td>previously</td>
<td>11</td>
<td>10</td>
<td>1</td>
<td>10.00</td>
<td>4.90</td>
</tr>
<tr>
<td>magnetic</td>
<td>11</td>
<td>10</td>
<td>1</td>
<td>10.00</td>
<td>4.90</td>
</tr>
<tr>
<td>visible</td>
<td>11</td>
<td>10</td>
<td>1</td>
<td>10.00</td>
<td>4.90</td>
</tr>
<tr>
<td>bonus</td>
<td>11</td>
<td>10</td>
<td>1</td>
<td>10.00</td>
<td>4.90</td>
</tr>
<tr>
<td>vivosmart</td>
<td>11</td>
<td>10</td>
<td>1</td>
<td>10.00</td>
<td>4.90</td>
</tr>
<tr>
<td>surprised</td>
<td>21</td>
<td>19</td>
<td>2</td>
<td>9.50</td>
<td>4.65</td>
</tr>
<tr>
<td>reminder</td>
<td>21</td>
<td>19</td>
<td>2</td>
<td>9.50</td>
<td>4.65</td>
</tr>
<tr>
<td>charges</td>
<td>21</td>
<td>19</td>
<td>2</td>
<td>9.50</td>
<td>4.65</td>
</tr>
<tr>
<td>pleased</td>
<td>30</td>
<td>27</td>
<td>3</td>
<td>9.00</td>
<td>4.41</td>
</tr>
<tr>
<td>honestly</td>
<td>10</td>
<td>9</td>
<td>1</td>
<td>9.00</td>
<td>4.41</td>
</tr>
<tr>
<td>l</td>
<td>10</td>
<td>9</td>
<td>1</td>
<td>9.00</td>
<td>4.41</td>
</tr>
<tr>
<td>lumia</td>
<td>10</td>
<td>9</td>
<td>1</td>
<td>9.00</td>
<td>4.41</td>
</tr>
<tr>
<td>ways</td>
<td>10</td>
<td>9</td>
<td>1</td>
<td>9.00</td>
<td>4.41</td>
</tr>
<tr>
<td>cell</td>
<td>10</td>
<td>9</td>
<td>1</td>
<td>9.00</td>
<td>4.41</td>
</tr>
<tr>
<td>motivated</td>
<td>29</td>
<td>26</td>
<td>3</td>
<td>8.67</td>
<td>4.24</td>
</tr>
<tr>
<td>uses</td>
<td>29</td>
<td>26</td>
<td>3</td>
<td>8.67</td>
<td>4.24</td>
</tr>
<tr>
<td>reminders</td>
<td>29</td>
<td>26</td>
<td>3</td>
<td>8.67</td>
<td>4.24</td>
</tr>
<tr>
<td>tool</td>
<td>19</td>
<td>17</td>
<td>2</td>
<td>8.50</td>
<td>4.16</td>
</tr>
<tr>
<td>messages</td>
<td>47</td>
<td>42</td>
<td>5</td>
<td>8.40</td>
<td>4.11</td>
</tr>
<tr>
<td>fantastic</td>
<td>28</td>
<td>25</td>
<td>3</td>
<td>8.33</td>
<td>4.08</td>
</tr>
<tr>
<td>love</td>
<td>575</td>
<td>513</td>
<td>62</td>
<td>8.27</td>
<td>4.05</td>
</tr>
<tr>
<td>amazing</td>
<td>46</td>
<td>41</td>
<td>5</td>
<td>8.20</td>
<td>4.02</td>
</tr>
<tr>
<td>platform</td>
<td>27</td>
<td>24</td>
<td>3</td>
<td>8.00</td>
<td>3.92</td>
</tr>
<tr>
<td>lifestyle</td>
<td>18</td>
<td>16</td>
<td>2</td>
<td>8.00</td>
<td>3.92</td>
</tr>
<tr>
<td>across</td>
<td>18</td>
<td>16</td>
<td>2</td>
<td>8.00</td>
<td>3.92</td>
</tr>
<tr>
<td>sports</td>
<td>9</td>
<td>8</td>
<td>1</td>
<td>8.00</td>
<td>3.92</td>
</tr>
<tr>
<td>myfitness</td>
<td>9</td>
<td>8</td>
<td>1</td>
<td>8.00</td>
<td>3.92</td>
</tr>
<tr>
<td>mainly</td>
<td>9</td>
<td>8</td>
<td>1</td>
<td>8.00</td>
<td>3.92</td>
</tr>
</tbody>
</table>
## Appendix E: Top 100 words with Lowest Ratio of Good to Bad Frequency in Amazon Reviews

<table>
<thead>
<tr>
<th>Word</th>
<th>Overall Freq</th>
<th>Good Review Freq</th>
<th>Bad Review Freq</th>
<th>Good Bad Freq Ratio</th>
<th>Good Bad Freq Ratio - Adjusted</th>
</tr>
</thead>
<tbody>
<tr>
<td>waste</td>
<td>30</td>
<td>1</td>
<td>29</td>
<td>0.03</td>
<td>0.02</td>
</tr>
<tr>
<td>disappointing</td>
<td>20</td>
<td>1</td>
<td>19</td>
<td>0.05</td>
<td>0.03</td>
</tr>
<tr>
<td>returned</td>
<td>43</td>
<td>3</td>
<td>40</td>
<td>0.08</td>
<td>0.04</td>
</tr>
<tr>
<td>terrible</td>
<td>26</td>
<td>2</td>
<td>24</td>
<td>0.08</td>
<td>0.04</td>
</tr>
<tr>
<td>inaccurate</td>
<td>23</td>
<td>2</td>
<td>21</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>returning</td>
<td>23</td>
<td>2</td>
<td>21</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>randomly</td>
<td>22</td>
<td>2</td>
<td>20</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>crap</td>
<td>11</td>
<td>1</td>
<td>10</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>poorly</td>
<td>11</td>
<td>1</td>
<td>10</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>nap</td>
<td>11</td>
<td>1</td>
<td>10</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>180</td>
<td>11</td>
<td>1</td>
<td>10</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>unable</td>
<td>11</td>
<td>1</td>
<td>10</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>disappointed</td>
<td>80</td>
<td>8</td>
<td>72</td>
<td>0.11</td>
<td>0.05</td>
</tr>
<tr>
<td>ant</td>
<td>10</td>
<td>1</td>
<td>9</td>
<td>0.11</td>
<td>0.05</td>
</tr>
<tr>
<td>failed</td>
<td>10</td>
<td>1</td>
<td>9</td>
<td>0.11</td>
<td>0.05</td>
</tr>
<tr>
<td>quit</td>
<td>9</td>
<td>1</td>
<td>8</td>
<td>0.13</td>
<td>0.06</td>
</tr>
<tr>
<td>beware</td>
<td>9</td>
<td>1</td>
<td>8</td>
<td>0.13</td>
<td>0.06</td>
</tr>
<tr>
<td>entered</td>
<td>9</td>
<td>1</td>
<td>8</td>
<td>0.13</td>
<td>0.06</td>
</tr>
<tr>
<td>broke</td>
<td>25</td>
<td>3</td>
<td>22</td>
<td>0.14</td>
<td>0.07</td>
</tr>
<tr>
<td>useless</td>
<td>32</td>
<td>4</td>
<td>28</td>
<td>0.14</td>
<td>0.07</td>
</tr>
<tr>
<td>reset</td>
<td>24</td>
<td>3</td>
<td>21</td>
<td>0.14</td>
<td>0.07</td>
</tr>
<tr>
<td>disappointment</td>
<td>16</td>
<td>2</td>
<td>14</td>
<td>0.14</td>
<td>0.07</td>
</tr>
<tr>
<td>d</td>
<td>8</td>
<td>1</td>
<td>7</td>
<td>0.14</td>
<td>0.07</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>1</td>
<td>7</td>
<td>0.14</td>
<td>0.07</td>
</tr>
<tr>
<td>troubleshooting</td>
<td>8</td>
<td>1</td>
<td>7</td>
<td>0.14</td>
<td>0.07</td>
</tr>
<tr>
<td>switches</td>
<td>8</td>
<td>1</td>
<td>7</td>
<td>0.14</td>
<td>0.07</td>
</tr>
<tr>
<td>book</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>0.17</td>
<td>0.08</td>
</tr>
<tr>
<td>story</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>0.17</td>
<td>0.08</td>
</tr>
<tr>
<td>promises</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>0.17</td>
<td>0.08</td>
</tr>
<tr>
<td>hence</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>0.17</td>
<td>0.08</td>
</tr>
<tr>
<td>additionally</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>0.17</td>
<td>0.08</td>
</tr>
<tr>
<td>mechanical</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>0.17</td>
<td>0.08</td>
</tr>
<tr>
<td>watched</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>0.17</td>
<td>0.08</td>
</tr>
<tr>
<td>frustration</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>0.17</td>
<td>0.08</td>
</tr>
<tr>
<td>stride</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>0.17</td>
<td>0.08</td>
</tr>
<tr>
<td>falls</td>
<td>19</td>
<td>3</td>
<td>16</td>
<td>0.19</td>
<td>0.09</td>
</tr>
<tr>
<td>warranty</td>
<td>25</td>
<td>4</td>
<td>21</td>
<td>0.19</td>
<td>0.09</td>
</tr>
<tr>
<td>stopped</td>
<td>54</td>
<td>9</td>
<td>45</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>touching</td>
<td>12</td>
<td>2</td>
<td>10</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>awful</td>
<td>12</td>
<td>2</td>
<td>10</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>reinstall</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>july</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>april</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>dring</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>recent</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>failure</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>continually</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>primary</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>unreliable</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>deleting</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>delivery</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>internet</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>suddenly</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>marketing</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>0.20</td>
<td>0.10</td>
</tr>
<tr>
<td>tapping</td>
<td>28</td>
<td>5</td>
<td>23</td>
<td>0.22</td>
<td>0.11</td>
</tr>
<tr>
<td>sucks</td>
<td>11</td>
<td>2</td>
<td>9</td>
<td>0.22</td>
<td>0.11</td>
</tr>
<tr>
<td>60</td>
<td>11</td>
<td>2</td>
<td>9</td>
<td>0.22</td>
<td>0.11</td>
</tr>
<tr>
<td>dead</td>
<td>27</td>
<td>5</td>
<td>22</td>
<td>0.23</td>
<td>0.11</td>
</tr>
<tr>
<td>died</td>
<td>36</td>
<td>7</td>
<td>29</td>
<td>0.24</td>
<td>0.12</td>
</tr>
<tr>
<td>fell</td>
<td>46</td>
<td>9</td>
<td>37</td>
<td>0.24</td>
<td>0.12</td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>2</td>
<td>8</td>
<td>0.25</td>
<td>0.12</td>
</tr>
<tr>
<td>heartbeat</td>
<td>10</td>
<td>2</td>
<td>8</td>
<td>0.25</td>
<td>0.12</td>
</tr>
<tr>
<td>gentle</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>0.25</td>
<td>0.12</td>
</tr>
</tbody>
</table>
Appendix F: Top 10 words Occurring only in Negative Reviews

<table>
<thead>
<tr>
<th>Word</th>
<th>Overall Freq</th>
<th>Bad Review Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>horrible</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>worst</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>refund</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>mi</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>prime</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>claimed</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>square</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>loosening</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>fiddle</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>breaker</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

ACKNOWLEDGMENT
The author would like to acknowledge the expert guidance of Professor Sunil Kakade provided in completing this project.

REFERENCES AND FOOTNOTES

This paper was submitted by Andrew Pacey on December 6, 2015 as a term project for Northwestern University’s CIS-436 course in big data technologies, completed as part of the Masters of Science in Predictive Analytics.


vi ETL is the process of extracting, transforming and loading data from a source system, such as the ERP system, into a target system, usually the enterprise data warehouse.


ix https://pypi.python.org/pypi/amazon_scraper/0.1.2


xi http://nltk.org/book

xii http://pandas.pydata.org/

xiii https://mahout.apache.org/users/classification/twenty-newsgroups.html

xiv For complete information on the Mahout implementation of Naïve Bayes, please refer to https://mahout.apache.org/users/classification/bayesian.html


xvi The words “horrible”, “worst” and “refund” occurred 29, 18 and 15 times in negative reviews, respectively, yet did not appear in any positive reviews to the ratio was effectively zero.

xvii https://mahout.apache.org/users/classification/bayesian.html