Beth Hilbert

Dualcore's Product Sentiment Analysis

IS8034 Big Data Integration

Product Sentiment Analysis

Business Question: What changes should be made to Dualcore's product line after considering customer product reviews? I will count the words from customer comments that match the positive and negative word libraries to create a score for sentiment and positivity. My goal is to improve Dualcore's customer experience through identifying positively and negatively reviewed products and brands.

1. Understanding Sentiment Analysis

Product rating comments are a useful source of information for both customers and retailers. In fact, the majority of corporate data is in semi or unstructured form. Tapping into these semi-structured information sources is needed to stay competitive. Analyzing text comments is referred to as sentiment analysis or, alternately, opinion mining. Processing text involves turning the words into numbers so powerful algorithms can be applied. These comments can be processed using a "bag of words" approach which counts words or through a "natural language processing" approach which takes into account the grammatical structure of our language.

I chose to quantify these opinions by comparing the words to a library of positive and negative words. The two measures I selected are Sentiment and Positivity. Sentiment is defined as #positives minus #negatives divided by the total of positive plus negative counts. Sentiment is scaled from -1 to 1. Positivity is the percentage of total matched words that are positive and ranges from 0 to 100%. Therefore, Sentiment scores higher than 0 and Positivity scores higher than 50% indicate the words in the specific product review tend to be more positive. I analyzed these scores for each product, product line, and brand.

2. Data Analysis Tools

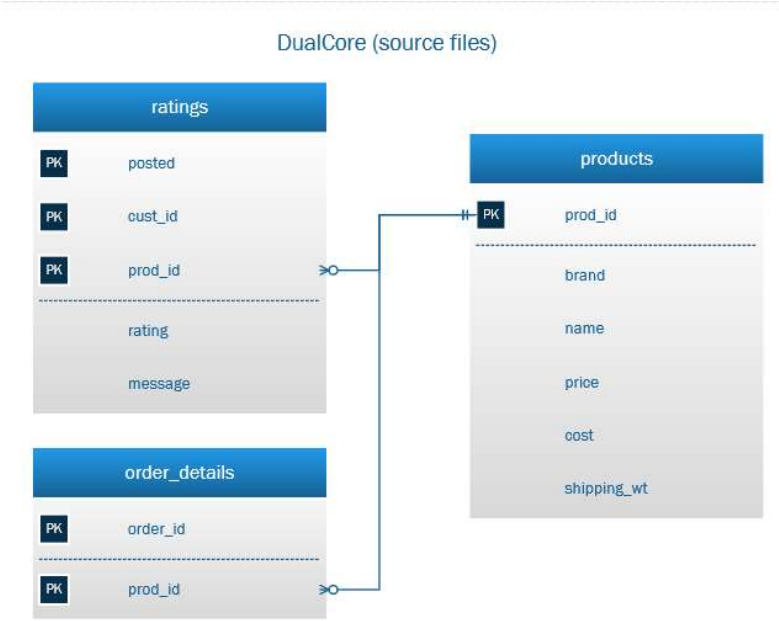
I utilized Hadoop as the system for processing these text comments. Text processing lends itself well to key-value formats. The algorithm for text processing isn't difficult but, until we had the ability to process datasets in parallel, it wasn't possible. By turning text into numbers, we can use algorithms to process large document data sets in parallel. A word count program maps (extracts) words from an input source and then reduces (summarizes) the results with a count of each word. In this analysis, instead of counting individual words, I counted the occurrence of whether the words were positive or negative for each product review.

The Hadoop ecosystem was used to store and process the large data set in parallel running on a virtual machine cluster provided by Cloudera. Scoop was used to ingest the data by chunks into the HDFS and the relationships were stored in Hive's metastore, where they could be accessed through key-part. This key-value model allows highly flexible data-storage and fast retrieval. It excels at complex math across one dimension, which is ideal for word count processing. MapReduce, Hive, and Impala were used to process the data.

After processing, the results were exported to a dimensional table for customized reporting. I used Microsoft Access but Oracle or Excel could have been used instead. This dimensional model allows complex search queries on multiple dimensions and offers flexibility in granularity (such as summation and aggregation).

3. Data Sources

Dualcore is a web-based company which sells technology products. Cloudera provided a dataset with Dualcore's sales and product reviews dated between 2008 and 2013. I combined this information with a sentiment analysis library to create a measurable opinion score. The data was at one time stored as transactions and ratings in a relational database management system. By the time the data was downstream, it was a set of tables, some connected. The relationships between the relevant Dualcore tables I used as source files are shown by the diagram below.



The word library I used was based on the specific reviews recorded by Dualcore, a sentiment training library from Cloudera, and Stanford's Natural Language Processing Library. I divided the words into positive and negative lists. I took into account that positive words like "good" are sometimes preceded by negative words like "not", giving them a negative sentiment ("not good"). Since the purpose of this analysis is to identify outliers (loved or disliked products), words that implied average ratings were not taken into consideration in these counts.

Positive Words:
good|happy|fantastic|great|perfect|enjoy|love|excellent|recommend|high|well|best|excellent|better|highly|satisfied|bargain|enjoy|awesome|pleased|enjoys|wellbuilt|decent|fine|great|bargain|easy|trust|perfectly|ideal|reliable|reasonable|worth|fine|flawless|sound|excelling|adept|supurb|impeccable|excelling|matchless|faultless|choicest|firstrate|incomparable|terrific|outstanding|unparalleled|highest|unequaled|ecstatic|excited
AND NOT not|don't|didn't

Negative Words:
absurd|angry|annoy|atrocious|averse|awful|aweful|bad|badly|blah|broke|broken|cheap|costly|crappy|crummy|damage|damaged|defective|deficient|dissatisfy|dissatisfied|fail|failure|failed|flawed|faulty|garbage|hated|hate|horrible|inadequate|inferior|junk|loath|lousy|mediocre|meh|negative|overpriced|poorly|poor|rubbish|shoddy|substandard|terrible|trash|unacceptable|unhappy|waste|worse|worst

4. Data Integration Process

Schema Alignment

Schema alignment includes determining the grain, attributes, and data sources. Since this business question results in recommending or removing individual products, the grain of the schema is one product (one brand/name combination). Attributes are reflected in the rows. Sources are reflected in the columns. The sources files are products, order_details, and ratings from Dualcore plus two sentiment dictionaries I created. Attributes are mapped from sources. Some attributes didn't exist (prod_line). Some attributes needed to be calculated (profit, sentiment, positivity) or aggregated (num_orders). Below is the final schema map at the final grain (one product). There were also intermediate tables created which are discussed in the record linkage section.

	products	**order_details**	**ratings**	**dictionary**	**Example**	**Comments**
prod_id	prod_id	prod_id	prod_id		1273641	Joins tables
prod_line	name CREATE FILTER				Tablet	Laptop, Server, Tablet
brand	brand				Byteweasel	
name	name				Tablet PC (10 in. display, 16GB)	Filter to just include Tablet PC
positive_count			messages AGGREGATED DEFAULT 0	Positive words	6	Counted against dictionary. Aggregated by product
negative_count			Messages AGGREGATED DEFAULT 0	Negative words	2	Counted against dictionary. Aggregated by product
sentiment			word count CALCULATED DEFAULT 0		.7	(Pos-neg) / (pos+neg)
positivity			word count CALCULATED DEFAULT 50		80	(Pos / (pos+neg)) * 100
price	price FORMATTED				495.79	Convert to dollars
cost	cost FORMATTED				397.43	Convert to dollars
profit	price-cost CALCULATED				98.36	Calculate. Convert to dollars
num_orders	prod_id	prod_id CALCULATED DEFAULT 0			4	Join tables on prod_id then COUNT

Issues encountered in this step:

- Determining the grain was the first issue. I wanted the final schema to reflect the magnitude of positive and negative reviews with a scale indicating the extent of reviews that were negative instead of just a yes/no response. Therefore, the initial design was at the grain of one row equaling one review. However, that resulted in a schema too complicated to analyze buried in repeated irrelevant information. I resolved this by recording a total count of positive/negative by product.
- The next issue I encountered was the finding a dictionary of words to compare with. Stanford has an excellent sentiment dictionary with a 10 point scale of positive and negative phrases. But this dictionary had been trained for movie reviews so some of the predictions were irrelevant to products. I next tried a 10,000 word dictionary from Cloudera. With this dictionary I encountered systems issue since the virtual machine uses an older version of MapReduce and could not recognize newer commands (such as add.cache) so the pointers didn't work correctly. The final dictionary solution was to combine information from both these sources with an analysis of the Dualcore product comments to generate a new dictionary specific to the domain of product reviews. This dictionary can be easily extended and trained.

Record Linkage

Record linkage includes data preprocessing and identity resolution by looking at rows. This involves formatting the source attributes to match the mediated schema and transitioning these rows to the final grain. The goal of record linkage is data in a consistent format and grain so that entity instances can be matched in the next phase.

The first step was to create a filtered table to identify and label products for three product lines. By creating this initial table, the product lines can easily be modified to add or rename product lines. All other queries are joined using this table as a filter. I called this new table product_line_filter and created an attribute named prod_line using the code `WHERE name LIKE "%Server%"`. The result is 67 products in 3 categories (8 laptop, 24 servers, and 36 tablets).

I determined the only information on orders I needed was a count of orders by product. A second table was created by joining order_details to product_line_filter, through matching the prod_id attribute. This creation also involved aggregating the individual orders to the grain of product. The result was 3,333,244 orders aggregated into a summary count of number of orders for each of the 67 products.

The next steps in record linkage involved sorting through the rating messages. I created a third table by joining the ratings table to the product_line_filter, through matching the prod_id attribute. The result was a table called ratings_filtered which was used as the basis for sentiment analysis. The grain of ratings had not changed at this

point, and included an instance for every rating. But the product filter reduced the size from 21,997 ratings to 1,810 rating rows. In this step, I also formatted the messages to remove case sensitivity, thus “Great” and “great” are counted as the same word.

The final part of record linkage involved creating tables with counts for positive_products and negative_products using the intermediate table ratings_filtered. Messages are separated into individual words (called tokens) and compared with the dictionaries. The matches were counted and aggregated by product. The positive_product included a count of positive words for the product and the negative_product included a count of negative words. These two tables combined are needed to give a full picture of the positive and negative comment counts. The ending grain matches the schema map grain of “product”.

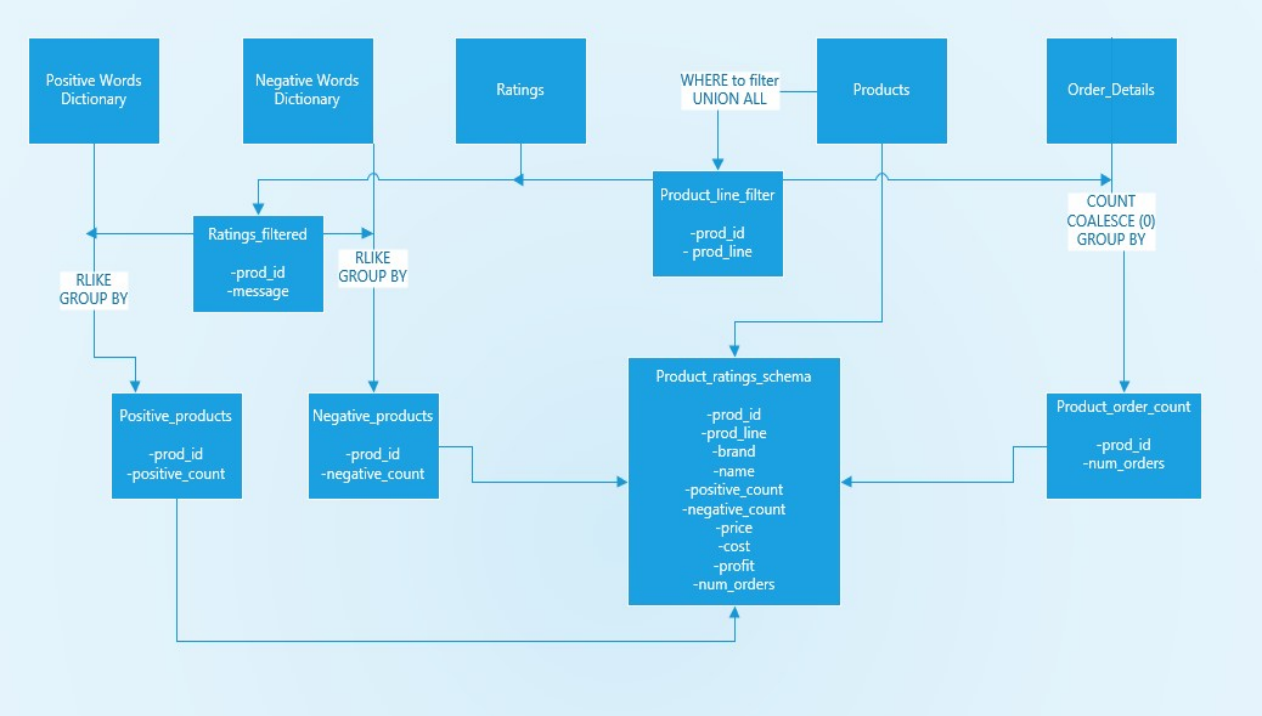
The end result of record linkage was all relevant tables matched the business question grain aggregating counts of orders, positive words, and negative words for each of the 67 products.

Issues encountered in this step:

- The first issue I encountered was the scope of the records I planned to analyze. Initially I only selected tablets, but eventually expanded the scope to also include not only the tablets themselves but all products related to tablets. I also expanded this filter to include product lines for laptops and servers. I decided to allow extendibility by creating an initial table for filtering records that could be modified to select different subsections of the data. The only fields in this filter table are prod_id and prod_line. This allows flexibility, as all other product attributes remain in the product table.
- The next issue I encountered in this step was implementing the dictionary. In addition to the positive_count file, I used another file which listed the positive and negative messages so I could verify the algorithms were working as expected and found that only 15 rows (instead of several hundred) were being flagged as negative. In attempting to fix this I tried eliminating some words and alphabetizing. I finally discovered the issue was caused by spaces beside the words so the dictionary was only matching words that had beginning and ending spaces. The word “poor” was not matched because it was by itself in message but if the word was in the middle of a sentence it was matched. Reworking the dictionary fixed this issue. A similar issue was found with contractions and the single quote. When I surrounded the dictionary with double quotes it correctly interpreted the apostrophe in contractions.
- The initial schema had more attributes included which greatly increased the complexity. I used regular expressions to extract attributes such as display size and concatenation to connect product brand and name. I determined those attributes were unnecessary to evaluate the business question.

- A major issue I encountered was using joins to change the grain did not always give the results I expected. Although it looks like SQL, HiveQL runs with limitations of HDFS and Java (MapReduce's language). I could not find an easy way to count by product directly into the final schema table without first creating intermediate tables.

Diagram of Intermediate Tables in Hadoop



Data Fusion

The final step of data integration is combining records which represent the same instance. This involves considering possibly incongruencies and identifying true value when sources provide conflicting values.

Data fusion involved creating a table within HiveQL (product_ratings_schema), then exporting the table to Microsoft Access for further refinement. The final HiveQL table merged the intermediate tables by joining on prod_id attribute. All these intermediate tables had already been converted to the final grain of "product" before this step of integration. Some fields were transferred unchanged (prod_Id, prod_line, brand, name, and num_orders). Others are formatted; price and cost are converted to currency format of two decimal places. Profit is calculated as price-cost ad then formatted to currency. Within Access, sentiment and positivity measures are calculated.

Issues encountered in this step:

- Handling nulls was an issue in this step. For nulls in counts, such as number orders and number of words, I set the value to 0. Zero makes sense for counts of these numbers because it's not an issue of failure to collect information that is available, but a result of there not being orders for a product or presence of specific negative/positive words in the message. For the measures, I set the scores to neutral opinions (0 for sentiment and 50 for positivity). Since the focus was to identify extremes, setting these measures to neutral removed them from influencing the model. Implementing these default rules was difficult because of how HiveQL handles missing values. While HiveQL looks like SQL, it behaves like Java. I experimented with COALESCE and IF statements in applying this logic. However, when I set count defaults to 0, it used that 0 in calculations which skewed the measure results. What I found worked was setting the counts within HiveQL, but calculating the measures within Access where there was more flexibility in IF statement implementation.
- The other issue I encountered in this step was that HiveQL was not the ideal solution for complex queries which grouped products along varying dimensions. I had initially planned for a key-value model since word counts naturally lend themselves to that model, but at this point I realized the business question required more joins than HiveQL supports. Access worked well to answer questions by slicing and dicing data along different dimensions.

5. Business Insights

One advantage of a dimensional model is data can be portrayed in many ways with minimal processing. I reviewed the results along three dimensions: product lines, brands, and overall positive/negative counts.

Analyzing Product Lines

Product Line Ratings

Product Line	sentiment	positivity	#ratings	brand	name	# orders	profit
Laptop	-1	0	1	XYZ	Laptop Cooling Pad	33	$26.16
	-0.1	44	9	ElCheapo	Laptop Cooling Pad	3653	$3.08
	0	50	0	Duff	Laptop Cooling Pad	0	$11.73
	0	50	0	TPS	1GB DDR2 667 Laptop R	226	$12.09
	0.1	56	9	Sparky	2GB DDR2 800MHz Lapt	3484	$15.27
	0.2	58	12	SuperGam	1GB DDR2 667 Laptop R	5148	$10.76
	0.5	75	57	ACME	Laptop Cooling Pad	4789	$0.45
	1	100	5	Overtop	1GB DDR2 667 Laptop R	3639	$8.18
Server	-1	0	6	Megachan	4 TB NAS Server	2391	$108.96
	-1	0	2	Orion	1 TB NAS Server	0	$46.52
	-0.3	33	6	Lemmon	8 TB NAS Server	1556	$51.82
	-0.3	36	11	Orion	Server Motherboard	4932	$13.95
	-0.1	46	24	TPS	Server Motherboard	6635	$20.82
	0	50	0	BuckLogix	Server (1U rackmount, I	0	$465.36
	0	50	0	BuckLogix	Server (1U rackmount, I	0	$885.60
	0	50	0	Dorx	1 TB NAS Server	0	$20.80
	0	50	0	Dualcore	Home Media Server	0	$8.84
	0	50	0	Dualcore	Server (1U rackmount, I	0	$589.32
	0	50	0	Gigabux	Server (2U rackmount,	0	$583.76

The first product line I reviewed was the laptop line, which is not a strength of Dualcore. Only several products are offered and they aren't rated highly. Based on sentiments, I would recommend dropping the laptop cooling pads other than the ACME. The highest selling laptop is only rated positive by half the people. This may be a line to investigate other manufacturers / products.

The second product line I reviewed was the server line, which offers the most products but generates the least overall sales. The Olde-Grey Home Media Server is very popular, with 5,800 sales and a general positivity of 76% (50% is neutral) from 17 reviews that matched the sentiment dictionary. Megachamp and Lemmon Servers, despite a high number of sales, were rated poorly and should be discontinued. The motherboards are also rated poorly, and other sources for this product should be reviewed. Many of the products in this line do not sell and further evaluation is warranted. Dualcore's reputation may be improved by dropping underperforming products and marketing the more well-liked server.

The tablet line, the most frequently sold product line, was the final line I reviewed. There are a few products that should be dropped from this line. The Orion 7 and 10 inch as well as the United Digital 7 and 10 inch all rated lower than 33% on positivity. The most frequently selling tablet is the Byteweasel, with 119,000 sales and 62% positivity. However, each sale results in a $1 loss so this needs to be investigated as to whether it's popular because of its value or if it's truly a well-liked product. I would recommend marketing the BuckLogix 7inch as it is generally well liked (88% with 9 reviews that matched the sentiment dictionary) and profits $50 with each sale.

Analyzing Brands

brand	Avg Of sentiment	Avg Of positivity	Avg Of total_pos_neg	Sum Of num	CountOfproc
Bigdeal	-0.60	20	6	2520	2
United Digistuff	-0.37	32	23	6162	3
XYZ	-0.33	33	1	69	3
Lemmon	-0.15	42	8	2156	2
Orion	-0.15	43	61	21015	10
TPS	-0.05	48	24	6861	2
ElCheapo	-0.05	47	15	4416	2
Megachango	-0.02	49	69	11615	6
Dorx	0.00	50	0	0	1
Dualcore	0.00	50	0	0	2
Duff	0.00	50	0	0	1
Gigabux	0.00	50	0	0	1
Krustybitz	0.00	50	0	0	3
Yoyodyne	0.00	50	0	0	2
Homertech	0.15	57	13	4584	2
ACME	0.17	58	57	4827	3
SuperGamer	0.20	58	12	5148	1
Byteweasel	0.20	60	215	124245	2
BuckLogix	0.27	63	8	1939	3
Sparky	0.28	64	17	7115	4
Tyrell	0.28	64	12	4252	5
Olde-Gray	0.50	75	18	7743	3
Electrosaurus	1.00	100	4	954	2

Dualcore sells 26 brands in these product lines. Five of these brands elicit strong negative opinions. Dualcore carries only 2 products in the Bigdeal brand, but their sentiment is -.60 (with 0 being neutral). United Digistuff accounted for 6000 sales but the sentiment was only -.37. I see these two brands as areas of concern if customer loyalty is valued. XYZ, Lemmon, and Orion should also be reviewed as their average sentiment is negative.

Analyzing Overall Positive and Negative Counts

total_p	positivity	sentime	positi	negativ	brand	name	prod_	num_orders
203	62	0.2	126	77	Byteweasel	Tablet PC (10 in. display, 64 G	Tablet	119801
57	75	0.5	43	14	ACME	Laptop Cooling Pad	Laptop	4789
44	75	0.5	33	11	Megachango	Sleeve for Tablet - Blue	Tablet	6714
24	46	-0.1	11	13	TPS	Server Motherboard	Server	6635
19	68	0.4	13	6	Megachango	Sleeve for Mini Tablet - Black	Tablet	2454
17	71	0.4	12	5	Orion	Tablet PC (10 in. display, 32 G	Tablet	5642
17	76	0.5	13	4	Olde-Gray	Home Media Server	Server	5814
15	33	-0.3	5	10	United Digistuff	Tablet PC (10 in. display, 64 G	Tablet	3246
12	58	0.2	7	5	SuperGamer	1GB DDR2 667 Laptop RAM	Laptop	5148
12	58	0.2	7	5	Byteweasel	Tablet PC (10 in. display, 16 G	Tablet	4444
11	36	-0.3	4	7	Orion	Server Motherboard	Server	4932
11	64	0.3	7	4	Homertech	Tablet PC (7 in. display, 16 GE	Tablet	3156
9	22	-0.6	2	7	Orion	Tablet PC (10 in. display, 64 G	Tablet	3848
9	44	-0.1	4	5	ElCheapo	Laptop Cooling Pad	Laptop	3653
9	56	0.1	5	4	Sparky	2GB DDR2 800MHz Laptop RA	Laptop	3484
8	12	-0.8	1	7	United Digistuff	Tablet PC (7 in. display, 16 GE	Tablet	2916
8	88	0.8	7	1	BuckLogix	Tablet PC (7 in. display, 8 GB)	Tablet	1939
7	57	0.1	4	3	Orion	Tablet PC (7 in. display, 32 GE	Tablet	1024
7	86	0.7	6	1	Orion	Tablet PC (10 in. display, 32 G	Tablet	3141
6	0	-1	0	6	Megachango	4 TB NAS Server	Server	2391
6	33	-0.3	2	4	Lemmon	8 TB NAS Server	Server	1556
6	50	0	3	3	ElCheapo	Mobile Bluetooth Keyboard a	Tablet	763

I further analyzed the dimensional data to inspect the correlation between customers who expressed strong enough opinions to write reviews and the general sentiment of their reviews. Reviews indicating the product was average are not included in this count. As expected, the more products sold, the more reviews were recorded (Byteweasel Tablet). However, I was surprised to see the number of people who took the time to write reviews used more positive than negative words. In the top 7 products measured by positive/negative word count, all but one had better than average reviews. If find this interesting because generally people tend to take time to voice their dissatisfaction more than their praise. This may reveal either a loyal base of customers who value product reviews so want to contribute or that those products have a loyal following so should be promoted to generate more sales. In addition, several products stand out with larger than average number of reviews but negative sentiments (two of which were motherboards).

Analyzing NGRAMS

I used the HiveQL's NGRAMS function to view words that frequently occur together. The format for this code is EXPLODE(NGRAMS(SENTENCES(LOWER(message)),3,10)). In messages with words marked negative, the most repeated words are mediocre and not great (words that occur more than 30 times) but the strongest feelings (horrible, bad, poor) were only a repeated several times. In messages with words marked positive, the most repeated words are decent and happy. While it's helpful to know words that occur together, analyzing the words using the dictionaries was more helpful in portraying the customer's sentiments on products and brands.

Messages marked as having negative words:

{"ngram":["not","great","but","not"],"estfrequency":35.0}
{"ngram":["great","but","not","bad"],"estfrequency":35.0}
{"ngram":["i","think","it","is"],"estfrequency":32.0}
{"ngram":["think","it","is","mediocre"],"estfrequency":32.0
{"ngram":["we","hate","this","item"],"estfrequency":2.0}
{"ngram":["item","was","a","bad"],"estfrequency":2.0}
{"ngram":["was","a","bad","value"],"estfrequency":2.0}
{"ngram":["this","item","was","the"],"estfrequency":2.0}
{"ngram":["this","product","was","horrible"],"estfrequency
{"ngram":["item","was","a","poor"],"estfrequency":2.0}

{"ngram":["not"],"estfrequency":70.0}
{"ngram":["mediocre"],"estfrequency":65.0}
{"ngram":["it"],"estfrequency":44.0}
{"ngram":["quality"],"estfrequency":42.0}
{"ngram":["bad"],"estfrequency":42.0}
{"ngram":["shoddy"],"estfrequency":41.0}
{"ngram":["great"],"estfrequency":35.0}
{"ngram":["but"],"estfrequency":35.0}
{"ngram":["is"],"estfrequency":33.0}
{"ngram":["think"],"estfrequency":32.0}

Messages marked as having positive words:

{"ngram":["the","item"],"estfrequency":44.0}
{"ngram":["was","good"],"estfrequency":42.0}
{"ngram":["this","is"],"estfrequency":40.0}
{"ngram":["item","was"],"estfrequency":36.0}
{"ngram":["the","product"],"estfrequency":32.0}
{"ngram":["good","value"],"estfrequency":30.0}
{"ngram":["product","was"],"estfrequency":26.0}
{"ngram":["i","feel"],"estfrequency":25.0}
{"ngram":["is","decent"],"estfrequency":25.0}
{"ngram":["it","is"],"estfrequency":25.0}

{"ngram":["i","feel","it","is","decent"],"estfrequency":25.0}
{"ngram":["this","is","a","decent","product"],"estfrequency":19.0}
{"ngram":["i","am","very","happy","with"],"estfrequency":9.0}
{"ngram":["this","is","better","than","the"],"estfrequency":8.0}
{"ngram":["better","than","the","previous","model"],"estfrequency":6.0}
{"ngram":["is","better","than","the","previous"],"estfrequency":6.0}
{"ngram":["very","happy","with","the","product"],"estfrequency":5.0}
{"ngram":["am","very","happy","with","the"],"estfrequency":5.0}
{"ngram":["i","absolutely","recommend","this","product"],"estfrequency":4.0}
{"ngram":["very","happy","with","this","item"],"estfrequency":4.0}

6. Future Steps

Sentiment analysis is challenging because of the complexity in how we combine words ("not good" means negative but when evaluated as two individual words it results in positive plus negative). It is also challenging because not all words have the same sentiment across domains ("cold beverages" and "cold room" have opposite meanings). Therefore, one way this model could be improved is calculating the sentiment of phrases, not just words in isolation. Another way to improve the model is to further train the dictionary for the product review domain by including intensity levels (such as negative/strong or negative/weak).

End Notes

Sources:

Cloudera's Documentation on Hadoop Custom Counters:
https://www.cloudera.com/documentation/other/tutorial/CDH5/topics/ht_example_4_sentiment_analysis.html

Github on EuroSentiment Analysis created by Mario Muñoz, June 2014:
https://github.com/EuroSentiment/eurosentiment-tutorial-java/wiki/Creating-your-first-sentiment-analysis-application.

Sentiment analysis library from Stanford Natural Language Processing Lab:
https://nlp.stanford.edu/sentiment

Big Data Integration by Xin Luna Dong and Divesh Srivastava, Morgan Claypool Publishing, 2015

Managing Data in Motion by April Reeve, Elsevier Publishing, 2013

Programming Hive by Dean Wampler, Jason Rutherglen, Edward Capriolo, O"Reily Media, 2012

University Cincinnati Professors Andrew Harrison and Zhe Shan. Professor Harrison taught this Big Data Integration class and shared his sentiment analysis wisdom assisting me in developing these algorithms. Professor Shan help with my MapReduce code and included text mining resources I referenced as part of his BI Data Mining course

HiveQL Code:

```
CREATE TABLE product_line_filter AS
SELECT uniontables.prod_id, uniontables.prod_line
FROM (SELECT prod_id, "Tablet" AS prod_line
FROM products
WHERE name LIKE "%Tablet%"
UNION ALL
SELECT prod_id, "Laptop" AS prod_line
FROM products
WHERE name LIKE "%Laptop%"
UNION ALL
SELECT prod_id, "Server" AS prod_line
FROM products
WHERE name LIKE "%Server%") uniontables

CREATE TABLE ratings_filtered AS
SELECT r.prod_id, lower(message) AS message, rating
FROM ratings r
JOIN table_prod_line p
ON r.prod_id = p.prod_id

CREATE TABLE positive_products AS
SELECT prod_id, count(prod_id) AS positive_count
FROM ratings_filtered
WHERE message RLIKE
'good|happy|fantastic|great|perfect|enjoy|love|excellent|recommend|high|well|best|excellent|better|highly|satisfied|bargain|enjoy|awesome|pleased|enjoys|well-
built|decent|fine|great|bargain|easy|trust|perfectly|ideal|reliable|reasonable|worth|fine|flawless|sound|excelling|adept|superb|impeccable|excelling|matchless|fault-less|choicest|first-
rate|incomparable|terrific|outstanding|unparalleled|highest|unequaled|ecstatic|excited'
AND message NOT RLIKE "not|don't|didn't"
GROUP BY prod_id
```

```
CREATE TABLE negative_products AS
SELECT prod_id, count(prod_id) AS negative_count
FROM ratings_filtered
WHERE message RLIKE
"absurd|angry|annoy|atrocious|averse|awful|aweful|bad|badly|blah|broke|broken|cheap|costly|crappy|crummy|damage|d
amaged|defective|deficient|dissatisfy|dissatisfied|fail|failure|failed|flawed|faulty|garbage|hated|hate|horrible|inadequate|i
nferior|junk|loath|lousy|mediocre|meh|negative|overpriced|poorly|poor|rubbish|shoddy|substandard|terrible|trash|unacce
ptable|unhappy|waste|worse|worst"
GROUP BY prod_id

CREATE TABLE product_order_count AS
SELECT p.prod_id,
 COALESCE(COUNT(d.prod_id),0) AS num_orders
FROM product_line_filter p
LEFT OUTER JOIN order_details d
ON p.prod_id = d.prod_id
GROUP BY p.prod_id

CREATE TABLE product_ratings_schema AS
SELECT t.prod_id,
 prod_line,
 brand,
 name,
 IF(positive_count !=0, positive_count, 0) AS positive_count,
 IF(negative_count !=0, negative_count, 0) AS negative_count,
 ROUND(price/100,2) AS price,
 ROUND(cost/100,2) AS cost,
 ROUND((price-cost)/100,2) AS profit,
 num_orders
FROM product_line_filter t
LEFT OUTER JOIN products p
ON t.prod_id = p.prod_id
LEFT OUTER JOIN positive_products pc
ON t.prod_id = pc.prod_id
LEFT OUTER JOIN negative_products nc
ON t.prod_id = nc.prod_id
LEFT OUTER JOIN product_order_count o
ON t.prod_id = o.prod_id

SELECT EXPLODE(NGRAMS(SENTENCES(LOWER(message)),3,10)) AS bigrams
FROM negative_messages
```

SQL Code in Access:

```
SELECT product_ratings_import.prod_id,
product_ratings_import.prod_line,
product_ratings_import.brand,
product_ratings_import.name,
product_ratings_import.positive_count,
product_ratings_import.negative_count,
[positive_count]+[negative_count] AS total_pos_neg,
Round(IIf([total_pos_neg]=0,0,(([positive_count]-[negative_count])/[total_pos_neg])),1) AS sentiment,
Round(IIf([total_pos_neg]=0,50,([positive_count]/[total_pos_neg])*100),0) AS positivity,
product_ratings_import.price,
product_ratings_import.cost,
product_ratings_import.profit,
product_ratings_import.num_orders
FROM product_ratings_import;
```