Hi Mike,

Here is the final run of small data matrix that I developed for the Helio sentiment analysis project (Small Matrix unlabeled 8d). I have also attached a detailed explanation of each column in the matrix (Helio Sentiment Analysis Matrix Detail). I have explained my basic approach for the new analyst below. I will send the three Python programs I developed to construct the small data matrix shortly.

All the Best,

Amy

Amy Gorman
Data Analyst
Alert! Analytics

General Approach

My approach is to scan the Common Crawl for documents that are relevant to the text analysis (documents that express meaningful sentiment about one of the phones) and then, for each relevant document, collect information about the sentiment toward key features of the phone (camera, display, performance, and operating system). I believe that patterns in the sentiment toward these features can help us develop a model to predict the overall sentiment toward each phone.

To gather information about the relevancy of the document and the sentiment toward phone features, I basically look for and count relevant words. To assess relevancy, I look for references to the device itself and references to words that indicate the document contains a meaningful view of the device. To gather sentiment toward each feature, I look for references to the feature itself and references to positive, negative, and neutral words. Documents are retained only if they have at least one mention of a phone or phone OS term (rows 5-12, columns B-H), and at least one of the following terms is present in the document: review; critique; looks at; in depth; analysis; evaluate; evaluation; assess.

I know this seems simple, but my mantra is to use the simplest, most efficient approach that provides meaningful results. I have found that more complex approaches to text analysis can become untenable as they are scaled up across billions of web pages. For example, I've found that more is not necessarily better when it comes to features. If you create a set of features that is too large, it seems to actually hurt the performance of classifiers because they will be exposed to too much 'noise', or irrelevant data. In principle, this could likely be mitigated with more sophisticated and non-linear machine

learning algorithms, but those algorithms will likely become very computationally expensive quickly.

To keep things manageable, I've found that it's best to use the simplest set of features possible, with the simplest classifier possible, to achieve the best results. Another hazard I've noticed (with regard to making things too complex) is that I may not always make the right interpretation for why certain results were obtained. In my experience, getting a good feature set that is both compact and useful for prediction is not always an easy goal to reach and can require extensive experimentation to achieve, but trying to keep it as simple as you can seems to be a good guiding principle.

Structure of Small Data Matrix

- Each row in the csv file is a webpage from the Common Crawl that is relevant to the analysis.
- Columns A-BF contains information I collected from each web page to assess the sentiment toward each phone.

It may be helpful to think of the small data matrix as being organized into the following five sections:

1. **Attributes that collect information about the relevancy of the webpage toward each device** (columns A-E).
    - These attributes basically answer the question: "Does this webpage express meaningful sentiment about a device we care about?"
    - I use the values in these columns to include the webpages I want to collect further information from and throw out those that aren't relevant.
1. **Attributes that collect information about the sentiment toward the operating system used on the phone**. (columns F-G)
    - These two columns record the number of mentions of terms for each operating system.
    - This column helps us determine if the operating system is discussed in the webpage, the number of positive, negative and neutral words about the operating systems are recorded in columns BA-BF.
1. **Attributes that collect information about the sentiment toward a phone's camera** (columns H-V).
    - These columns record the number of positive, negative and neutral words that are within reasonable proximity to the word camera or alternative words for camera (e.g., iSight)
1. **Attributes that collect information about the sentiment toward a phone's display** (columns W-AK)

- These columns record the number of positive, negative and neutral words that are in reasonable proximity to the word display or alternative words for display.

1. **Attributes that collect information about the sentiment toward a phone's performance** (columns AL-BF).
    - In the first set of columns (AL-AZ) I collect information about the sentiment towards the performance of a phone's hardware. These columns record the number of positive, negative and neutral words that are within reasonable proximity to the word performance or alternative words for performance.
    - In the next set of columns (BA-BF), I record the number of positive, negative and neutral words that are in reasonable proximity to words related to the operating system (e.g., iOS, Android operating system etc.)

Lessons Learned

I thought it might be helpful to the new analyst if I explained three key discoveries I made during the development of the small data matrix.

1. **Counting mentions of the device name produces too many irrelevant results.**
    - To determine if a webpage is relevant to our analysis I wrote the script to count the number of references to a given device (e.g., Samsung Galaxy) and accept a document as relevant if the number of references met a certain threshold. This approach let in a an unacceptable number of irrelevant documents, for example, web pages that contain trending search topics mention the name of a handset multiple times, but don't include any discussion of the handset itself. The method also let in lengthy and rambling forums, in which the handset is mentioned multiple times, but no coherent, meaningful sentiment is expressed.
    - This lesson is important because a key challenge in a typical large-scale sentiment analysis is to find the fractionally small number of documents that are relevant to the analysis across billions of webpages.
    - Current approach: The script now checks for both the name of the device and words that indicate that the document includes a meaningful assessment of the phone (e.g., review etc.). This method eliminates most of the irrelevant documents from the small data matrix.
2. **Looking for positive, negative, neutral words that are adjacent to a "feature" word produced too few results.**
    - I set the script to look for positive, negative, or neutral words immediately adjacent to terms that refer to camera, display, performance, and

operating system. A test returned almost no results, as very few webpages state opinion about a feature in such a literal way.
    - This lesson is important because, in most cases, hundreds of thousands of webpages need to be analyzed to get a meaningful assessment of sentiment.
    - Current approach: The script looks for positive, negative, and neutral words that are within five words of a feature word (camera, display, performance) or an alternate of a feature word (e.g., iSight).
3. **Apache Pig has limited application to text analysis.**
    - I originally explored using the Pig language to create the small data matrix. At first glance, it seemed to be the perfect way to script the function the mapper and reducer python programs currently execute. However it is likely not the best choice in our case for the following reason: The Pig language does not support regular expressions when counting word occurrences. This means that only single words or word phrases that are immediately adjacent to each other can be counted. This does not allow the flexibility to capture many different uses of language when describing sentiment toward a phone or one of the phone's features. Although this limitation can be overcome by building a 'user-defined function' (UDF) in another language (such as Python) and implementing the regular expressions in that UDF. However, I found the UDF function is not efficiently mapped across the distributed computational power of the EMR platform, creating a significant bottleneck in processing power. It seems to me that this inefficiency has the potential to increase the time required to process a batch of one billion pages from about two days to over two weeks, depending on the number of instances being used.
    - This lesson is important because using Apache Pig requires less effort and time than developing Python programs so it seems very attractive to leverage, but it will not deliver even a relatively simple text analysis in an efficient manner, which is critically important when analyzing text in scale.
    - Current approach: I wrote programs in Python to run several map and reduce jobs on Common Crawl data using EMR. These EMR jobs run multiple instances of the mapper Python script ***(Mapper.py)*** on multiple servers (nodes). Each of the mapper instances processes a subset of the overall set of Common Crawl files for the job. When a mapper completes it sends its results to a reducer process (run by the ***Reducer.py*** script). The reducer process combines the results from some number of mapper processes, then writes the combined results out to the S3 location. When a mapper or reducer completes, another process starts up in its place if

there are still files to process. The data from the job flows is in comma separated value format (.csv) and could be opened in Excel or a text editor. However, since the results are split into one folder per step and one file per reducer, I created a Python script (***ConcatenateFiles.py***) and a .csv file (***MatrixHeader.csv***) that combines everything together and adds in a header row.