Catherine Cornell

General Assembly

Data Science

Summer 2015

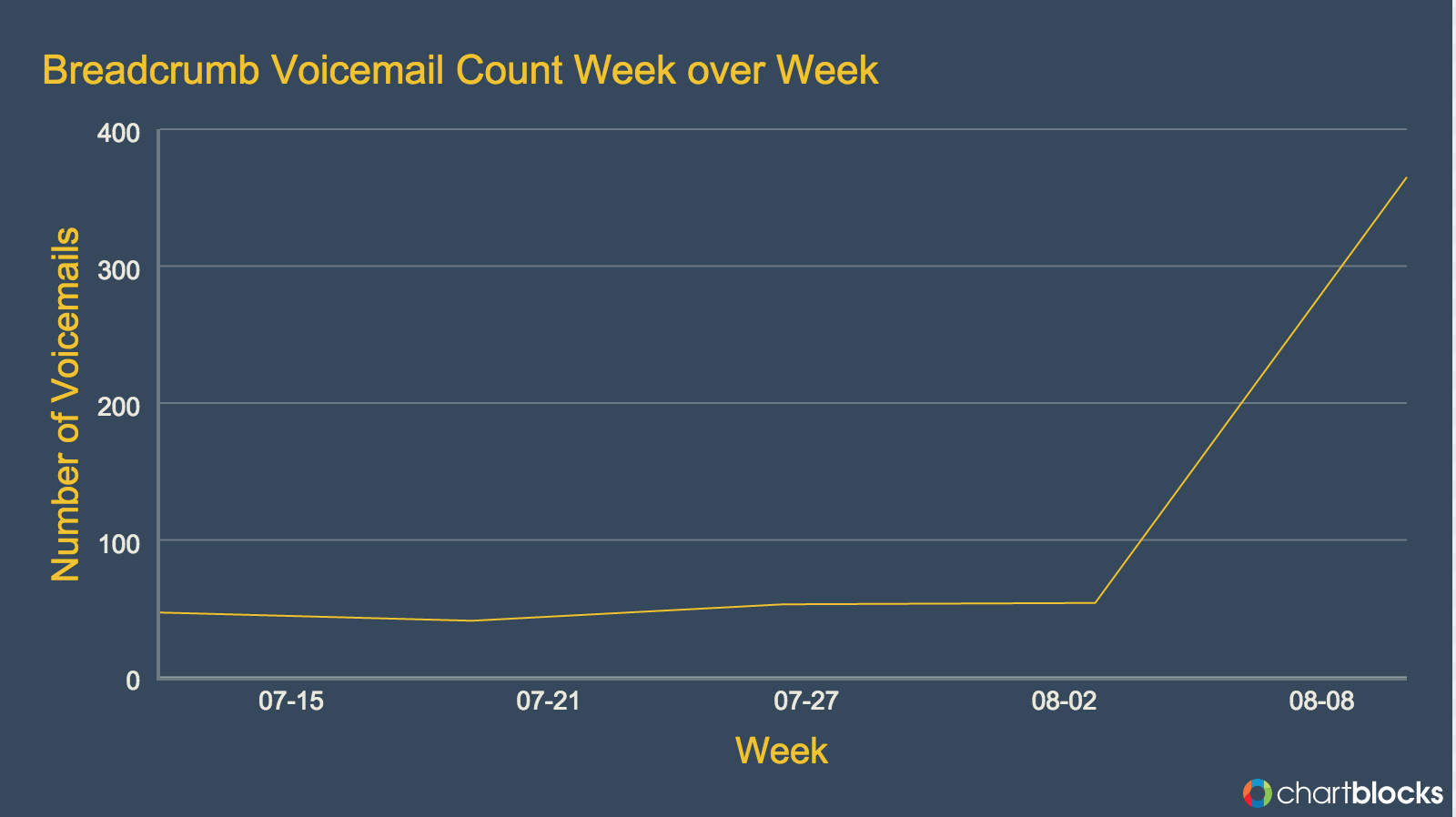
**Prioritizing Voicemails with Sentiment Analysis**

**Introduction:**

Breadcrumb Point of Sale is a mission critical software for bar, restaurants and other hospitality venues. Point of Sale (POS) systems allow for restaurants to track inventory and labor timesheets, print orders to the kitchen as well as authorize and capture payments of various tender types. Kitchens come to a screeching halt when their Point of Sale is not functioning properly. POS problems have a wide range of severity; with the worst case scenario being the venue losing out on payment.

98% of Breadcrumb POS’s merchants are one location “mom & pop” venues, meaning that in an overwhelming majority of cases the loss of a payment trickles from the restaurant down to the server, bartender, food-runner, busboy and kitchen staff. If Breadcrumb fails these individuals will usually walk away without pay for that night. The arresting impact that Breadcrumb’s uptime has on our users’ lives is why our technical support needs to be optimized to be the best it can be.

In summer of 2015 our parent company eliminated our technical call center team of 28 and replaced them with 5 Tier II agents. Unsurprisingly, with 82.2% of our answering agents gone the amount of voicemails needing attention rocketed.



I decided to perform a sentiment analysis on transcripts of the voicemails to help prioritize customer service tickets from venues experiencing urgent issues or customers who are especially frustrated and upset.

**Data Collection and Cleaning:**

Zendesk, Breadcrumb’s phone system provider, has a built in speech recognition function that transcribes and stores voicemails as text on the customer service ticket.

Zendesk offers a few different RESTful APIs for businesses to export their customer service data. Unfortunately the particular API that contained the voicemail transcripts did not feature filter parameters to only request data from tickets fulfilling specific criteria. Since we are not interested in non-voicemail tickets, or voicemail pertaining to other Groupon products it would be a waste to collect, clean, then sift through +100,000 tickets. I acquired Zendesk ticket ids to call the API by running some SQL queries in our internal ticket database: This query spit out all the Zendesk ticket ids for voicemails used in this project:

select id from tickets where created\_at > '2015-05-01 00:00:00' and subject LIKE 'Voicemail%' and ticket\_form\_id=6111;

Armed with the correct ticket ids I called the Zendesk Comment API to get the three features most useful for ranking voicemail priority: call duration, call back number, and the transcript of the voicemail.

The Zendesk Comment API returned a dictionary with three other dictionaries nested inside of it. Inside of the three nested dictionaries were varying numbers of lists with dictionaries nestled inside of them. At first I attempted to loop through each of the three nested dictionaries individually and write their contents into separate dataframes. However, this proved to be slow and resulted in poorly formatted dataframes. After a few Sypder crashes and JSON encoding errors, I switched to only mining for useful fields and writing those to a singular dataframe.

commentdata[page]=response.json()

nested\_data = commentdata[page]['comments'][0]['data']

data.append({

'comments': nested\_data['transcription\_text'],

'call\_duration': nested\_data.get('call\_duration'),

'callback\_number': nested\_data.get('from') })

After my Comments series had been created I passed it through a for loop to encode all the text to Unicode and through another for loop to ensure each transcript was a string.

During my initial exploration I noticed 9.6% of tickets had a null value for the voicemail transcript. I headed over to the Zendesk ticket to listen to the recording. I verified that null value comments are hang-ups, and filled the null values with the string “hang-up”. Our operations team has not yet decided how to rank hang-up tickets; I did not train my model to give then a positive or negative rating.

**Methods and Metrics:**

I choose to work with Text Blob over Python NLTK solely because Text Blob’s documentation for customizing the text classifiers were readily available and clear.

First step was to find the subjectivity, polarity, and sentiment of all the voicemails with the out of the box Text Blob classifiers. The goal of this model is to cherry pick the most upset venues. With this goal in mind combined my domain knowledge with the results of a few SQL queries to find the most painful common customer complaints. Venues managers, owners and staff run the gamut in terms of technical savvy. There no uniform language for reporting bugs and pain points for them. My domain knowledge was needed to help aggregate their varying terminology.

When training my model with custom phrases I ended up only feeding it negative train and test examples. This proved problematic. Only feeding the Text Blob Classifier negative training data gave the model an very pessimistic view: it went on to label all the voicemails as negative. Adding positive test and train examples resolved this.

Once my text blob classifier was corrected I charted the recalibrated subjectivity, polarity and sentiment. The subjectivity score of a voicemail measure how objective a statement is. A subjectivity score of 1 would indicate a sentence is likely an opinion (“This salad is seasoned perfectly”), while a score of 0 is an object statement (“All of the ingredients in this salad are certified organic”). Polarity scores live on a scale from -1 to 1. A negative score indicates unfavorable sentiment, 0 indicates neutrality, and positive scores indicate favorable sentiment. The sentiment column is a tuple of the polarity and subjectivity.

# Each word in the lexicon has scores for:

# 1) polarity: negative vs. positive (-1.0 => +1.0)

# 2) subjectivity: objective vs. subjective (+0.0 => +1.0)

# 3) intensity: modifies next word? (x0.5 => x2.0)[[1]](#footnote-1)

I then created features that found the probability distribution of the transcript being positive as well as the probability distribution of the transcript being negative. The probability distribution of negative calculates the probability of each word or phrase in the transcript. [[2]](#footnote-2)

**Feature Selection**:

Comparing the transcript with the smallest polarity and the transcript with the highest probability distribution of negative clearly revealed the latter was the more reliable metric.

Min Polarity:

Sentiment(polarity=-0.5, subjectivity=1.0)

Prob\_Negative: 0.6961684

"Hi, my name is Joan up we got Somebody Tavern. It's xxx-xxx-xxxx. I need some help with the printer and it's off-line and I need to set up some sort of button for coupons that I wanna start taking. Could you call me today would be great.”

Max Prob\_Neg:

Sentiment(polarity=0.113333, subjectivity=0.545)

Prob\_Negative: .999764

"Hi, my name is Kelly, I'm calling for Lisa Richard, this is absolutely [expletive] after 9:00 I been on hold for 15 minutes and I cannot get a hold of anyone. I need someone to help my printers are not functioning and this is ridiculous I pay for your service. Somebody should be able to answer my phone call. Please call back. Thanks."

**Presentation Preparation and Next Steps:**

Sorting the dataframe by probability distribution of negative gave me the ranked list of voicemails. I then wrote the dataframe to a csv and uploaded it as a Google doc for our 5 agents to work off of.

While this processing will save the team from having to listen to all the voicemails and likely reduce our Account Management compliant tickets, its still a manual process. I would like to find a way to combine API calls between the Zendesk Main Ticket API and the Zendesk Comment API to eliminate the manual step of querying the ticket database for unsolved voicemails tickets. The other bookend of needing to export the data then upload the csv into a Google doc is also manual. I would like to have a internal dashboard where our five tier II agents can view priority ticket numbers without me needing to export a csv frequently.

1. **TextBlob Sentiment: Calculating Polarity and Subjectivity**

   http://planspace.org/20150607-textblob\_sentiment/ [↑](#footnote-ref-1)
2. **Stat Trek** http://stattrek.com/probability-distributions/probability-distribution.aspx [↑](#footnote-ref-2)