What Really Matters? Leveraging Customer Reviews to Understand the Restaurant Experience Using Aspect Based Opinion Mining

Section 1 | Team 2
Jithin Mathews George | Katie Krawczyk | Hemanth Rao Natarajan
Vineet Tanna | Hanjing Wang

MSBA 6330 | Big Data Analytics

December 5, 2017

Github Repository: https://github.com/vineettanna/Aspect-Based-Opinion-Mining-Using-Spark

Introduction

You own a restaurant, and you want good reviews, but what really matters? Answering this question that permeates the minds of restaurant owners everywhere, is exactly the goal of this project. To accomplish this goal, we will use Topic Modeling (Latent Dirichlet Allocation) and sentiment analysis from Spark MLlib on 4.7M restaurant reviews from 12 metropolitan areas provided by Yelp.com, to better understand key themes and sentiments that appear in reviews for restaurants across various cuisines. Then through a regression analysis, we aim to derive insights about what dining experience topics carry the most weight when explaining restaurant star rating reviews.

Background

Yelp reviews contain a great amount of information about businesses, but the semi structured and unstructured nature of the data can make this information difficult to extract. The Yelp.com data set is a collection of semi-structured JSON files ranging from 125 KB to 4 GB in size. In particular, the review file contains 4.7 millions rows of review text, posing a data volume challenge. Additionally, the files are nested JSONs and have a variety of formats, including strings, binary variables, and numeric values.

As described in the introduction, the objective of the use case is to extract information from restaurant reviews across numerous cuisines, to provide restaurant owners and managers with a better understanding of the experience factors that matter most for their Yelp review score. In order to do this at the scale of the data, the analysis will utilize algorithms and functions from general PySpark, SparkSQL, Python, SparkML, and Spark MLlib executed on an Amazon Web Services Elastic Cloud Cluster.

Data

The data source for this project is an open data set from Yelp.com. Within this dataset, there are five unique tables; user, review, business, tip, and checkin. From these five files, two are of particular interest for this project, review and business.
The business table is a nested JSON file formatted as follows:

```
{
   // string, 22 character unique string business id
   "business_id": "tnhfDv5Il8EaGSXZGiuQGg",
   // string, the business's name
   "name": "Garaje",
   // string, the neighborhood's name
   "neighborhood": "SoMa",
   // string, the full address of the business
   "address": "475 3rd St",
   // string, the city
   "city": "San Francisco",
   // string, 2 character state code, if applicable
   "state": "CA",
   // string, the postal code
   "postal code": "94107",
   // float, latitude
   "latitude": 37.7817529521,
   // float, longitude
   "longitude": -122.39612197,
   // float, star rating, rounded to half-stars
```

```
    "stars": 4.5,
    // integer, number of reviews
    "review_count": 1198,
    // integer, 0 or 1 for closed or open, respectively
    "is_open": 1,
    // object, business attributes to values. note: some attribute values might be objects
    "attributes": {
        "RestaurantsTakeOut": true,
        "BusinessParking": {
            "garage": false,  "street": true, "validated": false,  "lot": false,
            "valet": false
        },
    },
    // an array of strings of business categories
    "categories": [
        "Mexican",
        "Burgers",
        "Gastropubs"
    ],
    // an object of key day to value hours, hours are using a 24hr clock
    "hours": {
        "Monday": "10:00-21:00",
        "Tuesday": "10:00-21:00",
        "Friday": "10:00-21:00",
        "Wednesday": "10:00-21:00",
        "Thursday": "10:00-21:00",
        "Sunday": "11:00-18:00",
        "Saturday": "10:00-21:00"
    }
}
```

The review table is also a JSON file, formatted as follows:

```
{
    // string, 22 character unique review id
    "review_id": "zdSx_SD6obEhz9VrW9uAWA",
    // string, 22 character unique user id, maps to the user in user.json
    "user_id": "Ha3iJu77CxlrFm-vQRs_8g",
    // string, 22 character business id, maps to business in business.json
    "business_id": "tnhfDv5Il8EaGSXZGiuQGg",
    // integer, star rating
    "stars": 4,
    // string, date formatted YYYY-MM-DD
    "date": "2016-03-09",
    // string, the review itself
    "text": "Great place to hang out after work: the prices are decent, and the
ambience is fun. It's a bit loud, but very lively. The staff is friendly, and the food is
good. They have a good selection of drinks.",

    // integer, number of useful votes received
    "useful": 0,
    // integer, number of funny votes received
    "funny": 0,
    // integer, number of cool votes received
    "cool": 0
}
```

Overall, the review table has 4.7 million rows of reviews, giving it a file size of 3.7 GB and the business table contains 156,000 businesses with a file size of 125 KB.

Methodology

All data preparation for LDA and the LDA model are executed in a Jupyter notebook using PySpark. Results from each step in the execution flow (Figure 1) directly influence the next step in the process and integrate seamlessly across different libraries in the Spark environment.

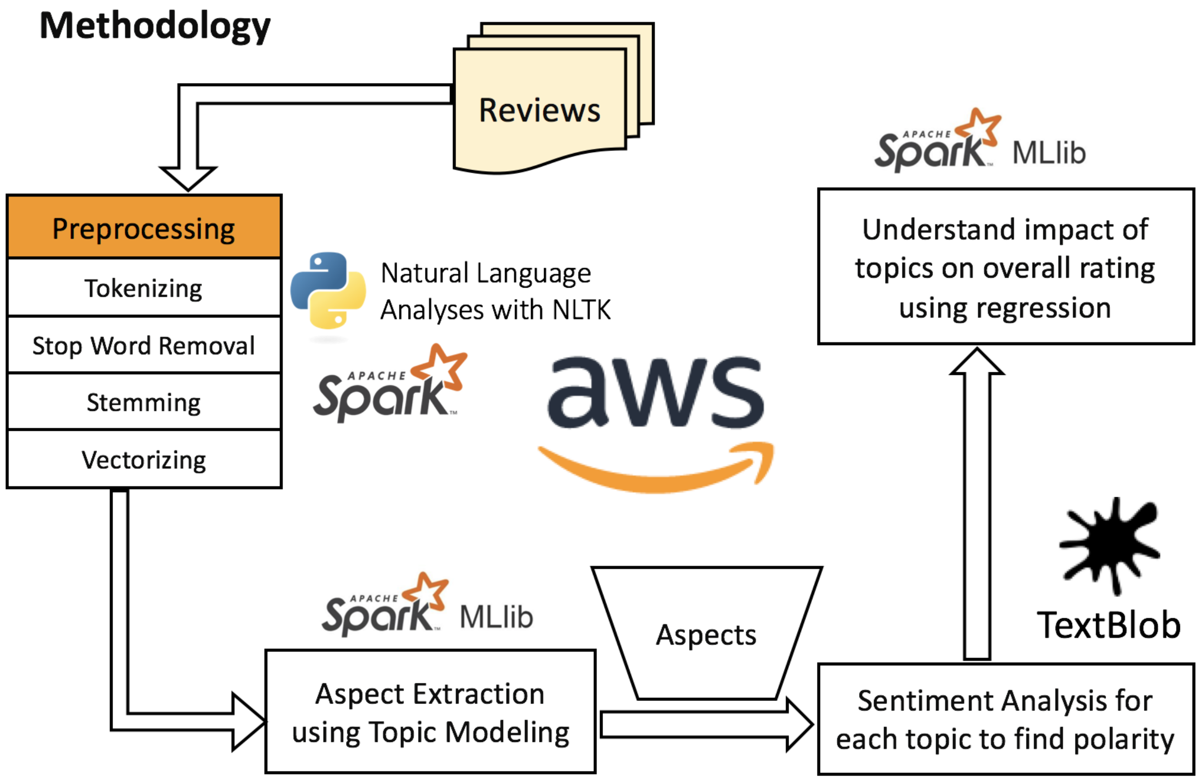
Methodology
Reviews
Preprocessing
Tokenizing
Stop Word Removal
Stemming
Vectorizing
Natural Language Analyses with NLTK
Spark
aws
Spark MLlib
Aspect Extraction using Topic Modeling
Aspects
TextBlob
Sentiment Analysis for each topic to find polarity
Spark MLlib
Understand impact of topics on overall rating using regression

Figure 1: Project Process Flow

LDA Data Preparation

In order to effectively use LDA, the data must be properly cleaned and formatted for the algorithm. This means that after reading in the files, the data must be filtered to retain only the desired restaurant categories. From this step, each row in the table needs a unique index so it will be accepted by the LDA algorithm from Spark MLlib. Once indexed, sentences will be tokenized, stop words will be removed, individual words will be stemmed so they are in their root form, and they will finally be put into dense vector format. Each of these steps ensures that the data is in its cleanest format and helps increase the potential for getting desirable results.

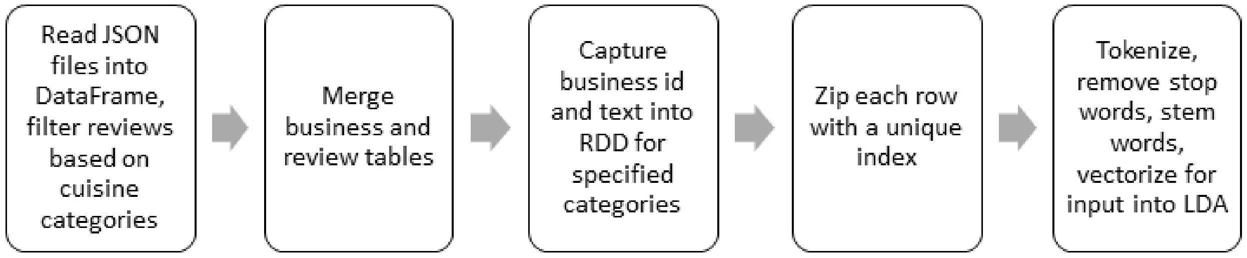
Read JSON files into DataFrame, filter reviews based on cuisine categories
Merge business and review tables
Capture business id and text into RDD for specified categories
Zip each row with a unique index
Tokenize, remove stop words, stem words, vectorize for input into LDA

Figure 2: Data Pre-Processing Flow for LDA Algorithm

Step One:
Since there is a vast amount of data in the business and review tables, but only a few columns are needed for analysis, reading the JSON files into a DataFrame will allow for quick queries and filtering of the data set.

```
mydir = ('file:/home/cloudera/Yelp/review.json')
sqlContext = SQLContext(sc)
df1 = sqlContext.read.json(mydir)

mydir2 = ('file:/home/cloudera/Yelp/business.json')
df2 = sqlContext.read.json(mydir2)
```

Step Two:
Join the business table to the review table based on business id. This is the common feature between the two tables and will allow for filtering through the reviews based on categorical key words.

Step Three:
The category column of the DataFrame contains arrays of category keywords, so in order to create a DataFrame containing only reviews from the desired cuisine, use a where condition searching for arrays that contain the desired keyword.
For example, if looking to analyze reviews for restaurants serving Mexican food, use

```
asian=df.where(array_contains(df.categories,"Chinese")
```

where df is the name of the combined review and business tables. Then, register this table as a temporary table so that it can be queried in the next step.

Step Four:
At this point, the table now has columns for all of the possible features in the business and review tables, but for the LDA, the only columns desired are the text and the business id (which is only helpful until an index is added). Use a SQL command to acquire these two columns as part of a new DataFrame.

Step Five:
Convert the text and business id DataFrame into an RDD and use regular expressions to remove non-alphanumeric characters and words that are shorter than three letters. This will start the process of cleansing the text reviews for the LDA model.

Step Six:
Convert the RDD back to a DataFrame, for the purpose of attaching an index to each row. In the DataFrame format, a unique index is determined, then the DataFrame is converted back to an RDD, zipping the unique index identifiers to each row. Then, it is converted back to a DataFrame so that the next set of preparations can be completed using ML functions.

Step Seven:
Each of the reviews need to first be separated into an array of comma separated words using the tokenizer function. Then, stop words are removed with a stop words function, using a custom combination of a common stop words list from the Spark ML library and words that are commonly used to describe food in a dining experience, such as good, bad, or great as they are not indicative of a specific experience topic, but rather only used to describe one of the topic aspects. Next, the words are tokenized using the Porter stemmer algorithm from NLTK to bring them down to their common roots, then these words are placed into vectors.

Step Eight:
Construct a corpus from the DataFrame, mapping in the index and the vector column of tokens. In this final preparation step, these two columns are converted to an RDD and are now ready to be input into the LDA algorithm.

LDA with Spark MLlib

The purpose of using LDA is to extract common aspects that can be described by a dining related topic. For this analysis, the LDA algorithm comes from Spark MLlib. This means the input must be in RDD format and each row must have a unique index attached to the features for the model.

Step One:
Using the cleaned corpus RDD, train the LDA model. Note that there are many choices of input parameters, but the choices in this analysis are k, which is the number of topics, and the seed. When selecting a number of topics, the “correct” number was determined through trial and error as there is no way to discover the true number of topics in a document. Additional parameter options can be found in the documentation.

```
lda_model = LDA.train(rdd=corpus, k=3, seed=12)
```

Step Two:
Extract the topics and aspects from the trained model.

```
topics = lda_model.describeTopics(maxTermsPerTopic=10)
vocabulary = count_vectorizer_model.vocabulary
```

Step Three:
For the final step, print each of the topics and their vocabulary words. From these topics, make inferences about what topics the aspects may be capturing. The aspects from the topics defined in this step will be used to filter reviews in the next step. Note that while each topic contained ten aspects, all ten aspects were not used in the filtering step. Only words that are not ambiguous, in that they are not likely to describe multiple topics within the dining experience, are used in the next phase, sentiment analysis.

Sentiment Analysis with TextBlob

With the aspects mined in the previous stage, the goal is now to understand the sentiment around the topics the aspects define. To accomplish this, the analysis is completed using the Python library TextBlob within the spark environment.

Step One:
Filter the reviews based on the aspects from topic modeling to extract the sentences which contain any of the aspects for each topic. This was done using NLTK, leveraging the map function and a UDF created to check for any of the aspects.

Step Two:
Aggregate the reviews for each topic for each individual restaurant in order to understand the sentiment for a specific location.

```
##Step One and Two
words = [x.lower() for x in
['noodl','dish','chicken','fri','rice','soup','sauc','beef','pork
','tast']]
def intersect(row):
     # convert each word in lowercase
     row = [x.lower() for x in row.split()]
     return True if set(row).intersection(set(words)) else False

import nltk.data
sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
summarised=asian_text.rdd.map(lambda
(business,text):(business,sent_detector.tokenize(text.strip())))\
.map(lambda (business,text):(business," ".join(filter(lambda
sent: intersect(sent),text))))\
.reduceByKey(lambda x,y:x+y).toDF()\
.withColumnRenamed('_1','BusinessId').withColumnRenamed('_2','Rev
iew')
```

Step Three:
Leverage the map function to calculate the average polarity for each topic at each individual restaurant leveraging TextBlob. To do this, use the sentiment property, which provides the polarity of each input sentence.

```
asian_sample_rdd=summarised.rdd
asian_sample_polarity = asian_sample_rdd.map(lambda
x:(x[0],TextBlob(x[1]).sentiment.polarity))
```

The sentiment analysis was performed for each topic and at a restaurant level, i.e. for each restaurant. As final output for each restaurant, a polarity was assigned, based on all sentences in the reviews containing the aspects mined in the previous phase.

Linear Regression Model with Spark MLlib

From the polarities determined for the key topics of each restaurant, to better understand the effect each of these features has on the overall restaurant rating, a linear regression model can be utilized. This model works well because the beta coefficients provide information about the relationship between each topic and the rating.

Step One:
Import the linear regression model from Spark ML library. The algorithm is coming from this library because the input from the previous step is in DataFrame format.

```
from pyspark.ml.regression import LinearRegression
```

Step Two:
Set the numeric columns to a list containing the names of all columns capturing polarity for a specific topic.

```
numericCols = ['polarity1','polarity2']
```

Step Three:
Build a vector assembler that takes the numeric columns as input and outputs these features in vector format to a new column.

```
from pyspark.ml.feature import (VectorAssembler)
assembler = VectorAssembler(inputCols=numericCols,
outputCol="features")
```

Step Four:
Construct a linear regression model that takes the feature vector from the previous step as input, and the number of stars for the review as the labelled column.
```
lr = LinearRegression(maxIter=10,
featuresCol="features",labelCol="stars")
```

Step Five:
Transform the data set use the vector assembler from step two.
```
training2=assembler.transform(training)
```

Step Six:
Train the linear regression model using the vectorized DataFrame from the previous step. Then, use this model to examine the beta coefficients for each feature to determine which carries greater weight in contributing to the overall star review.

Handling the Scale of Data with AWS

With 4GB of data, the need for a computing platform to handle the volume of data with faster processing speed than a local machine is adamant. Additionally, with the group nature of the project, the solution needed to be conducive to this environment. Based on these requirements, Amazon Web Services' Elastic Cloud Computing (EC2) offered an effective solution. The base configuration for the task is an EC2 instance with 2 virtual CPUs, 8 GB of ephemeral memory, and 30 GB of Elastic Block Storage for memory on an Ubuntu operating system.
Advantages of using this solution are:

- **Flexibility**: Due to the diversity of tasks being carried out in the project flow, the computing power and memory needs change based on the task. This solution allows for the expansion of memory and increase computing power without losing any data.
- **Collaboration:** The collaborative nature of this approach is illustrated by the ability to share an image of the instance with other members in the AWS system. So, each individual is able to operate their own instance, but the configuration of the cluster, including tools installed on the instance, can be uniform across team members.

After launching the EC2 cluster with the described configuration, additional actions to configure the cluster include:

- Download and Install Anaconda 2.5.0 (Python 2.7)
- Configure Python Path and Jupyter Notebook
- Install Java and Scala as they are prerequisites for running Spark
- Install Py4J so that the Python interpreter can dynamically access Java Objects in a Java Virtual Machine
- Install Spark 1.6.0 and Hadoop 2.6.0

The detailed steps of installation can be found at the link below.
Installing Jupyter Notebook and Spark on EC2
Once the Jupyter Notebook is running with Spark, data files can be uploaded to EBS using the Secure File Transfer Protocol (SFTP) on FileZilla. The guide to this can be found in the link below:
Transferring Files to EBS using FileZilla

Findings

This analysis was conducted on three distinct groups of restaurants, Latin American cuisines, Chinese food, and breweries. The steps in the analysis were the same for each type of restaurant, but each group yielded different aspects belonging to each topic.

Latin American Cuisine

To build the body of reviews for Latin American restaurants, the review frame was filtered for restaurants containing the aspects:

- Argentine
- Colombian
- Emanadas
- Latin American
- Mexican
- New Mexican Cuisine
- Peruvian
- Spanish
- Tacos
- Tex-Mex
- Venezuelan

From these reviews, the aspects emerging from the LDA analysis were food and service.

Food Aspects:

- Taco
- Sauce
- Flavor
- Chicken
- Taste
- Cheese
- Dish
- Try
- Fry

Service Aspects:

- Time
- Service
- Table
- Order
- Drink
- Ask
- Even
- Wait

Using these aspects, reviews were filtered to build bodies of reviews for each of these topics based on restaurant type. The resulting RDD then captured polarities for each of these topics for each unique restaurant in the Latin American category. Based on these results, the linear regression model yielded the following results for the importance of each topic as it relates to the overall review score for the restaurant.

Topic	Beta Coefficient
Food	0.7642
Service	2.1798

R-squared: 0.1977795817

Interpretation of results:
For each unit increase in food polarity, review ratings increase by .07 holding service constant. Similarly, for each unit increase in service polarity, review ratings increase by .2 holding food constant.

Based on these interpretations, service is believed to be of more importance in reviews for Latin American restaurants. A single unit increase in service polarity yields three times the result of a unit increase in food polarity. This strength indicates that within this category, restaurants looking to improve their overall Yelp ratings should consider improving their service before putting focus on improving their food.

Chinese Food

To build the body of reviews for Chinese restaurants, the review DataFrame was filtered for restaurants containing the keyword Chinese. From these reviews, the aspects emerging from the LDA analysis were once again food and service.

Food Aspects:
- Noodle
- Dish
- Chicken
- Fried
- Rice
- Soup
- Sauce
- Beef
- Pork
- Taste

Service Aspects:
- Service
- Time
- Lunch
- Price
- Friendliness

From the aspects, the sentiment analysis conducted followed the same process as that for the Latin American food, however the results indicate that the importance of each topic is different.

Topic	Beta Coefficient
Food	1.2324
Service	0.8339

R-squared: 0.1027998257

Interpretation of results:
For each unit increase in food polarity, review ratings increase by .1 holding service constant. Similarly, for each unit increase in service polarity, review ratings increase by .08 holding food constant.

These results indicate that the importance of food versus service for Chinese food is the opposite of the Latin American cuisines. An increase in food polarity has the potential to yield greater results for the overall restaurant review than service.

Breweries
To build the body of reviews for breweries, the review frame was filtered for restaurants containing the aspects:
- Beer
- Beer Bar
- Beer Garden
- Beer Gardens
- Beer Hall
- Beer Tours
- Breweries

From these reviews, the aspects emerging from the LDA analysis are atmosphere and beer in addition to food and service.

Atmosphere Aspects

- Service
- Friendliness
- Atmosphere
- Staff
- Fun

Service Aspects

- Time
- Table
- Wait
- Server
- Drink
- Minute

Beer Aspects

- Beer
- Brew
- Breweries
- Bar
- Tap
- Local
- Craft
- IPA
- Select
- Ale

Food Aspects

- Fried
- Chicken
- Order
- Burger
- Menu
- Sandwich
- Cheese
- Sauce
- Taco
- Fish

From the aspects, the sentiment analysis conducted followed the same process as the previous two, yielding the results:

Topic	Beta Coefficient(statistically significant)
Food	0.6106
Service	1.013
Beer	0.9753
Atmosphere	1.0632

R-squared: 0.2237817348

Interpretation of results:
For each unit increase in food polarity, review ratings increase by .06 holding all other features constant. Similarly, for each unit increase in service polarity, review ratings increase by .1 holding all other features constant. In addition, for each unit increase in beer polarity, review ratings increase by .01 holding all other features constant and for each unit increase in atmosphere polarity, review ratings increase by .1 holding all other features constant.

Of the four distinct topics, service, beer, and atmosphere are nearly equal in their influence on overall restaurant reviews. Food on the other hand, is of the least importance, which seems logical based on the brewery category. While many breweries have food, their primary business is beer, so maintaining the variety, flavor, and quality of the beer is key to keeping their business alive.

Conclusions

From the linear regression, it is clear that there are certain types of experience topics that matter more than others and those topics vary by cuisine. However, for each cuisine, there is now a better understanding of what customers are talking about and how they influence a restaurant’s overall Yelp review ratings.

Based on this analysis, further exploration on the topic can come in multiple forms. One way would be to repeat this process looking at reviews from other industries. Replicating the analysis on product reviews, customer surveys, or credit card feedback form would yield different topics and different results than this restaurant analysis. Another exploration would be to build a predictive model that uses polarities from customer reviews alone to predict overall restaurant ratings.

From a big data tool perspective, it is clear that Spark’s ability to integrate Python libraries like NLTK and TextBlob seamlessly enables fast computing on big data sets and ease of use for the analyst. Additionally, Spark MLlib in conjunction with the computing power of AWS empowers users to perform machine learning on large datasets that a local machine would struggle to compute efficiently.

As for the team experience, the team as a whole feels that this was a successful project that was only successful because of team collaboration. Due the the expanse of methods used for the project, dividing the work amongst each member of the team made the project feel manageable because each team member has a different set of specialty skills. Collectively, the team elicits that this was an excellent learning experience because they were able to use multiple Spark libraries integrated with multiple Python libraries to tackle a data set with 4.7 million rows and gain insights that could impact businesses in a tangible way.

Bibliography

Apache Spark. *Spark SQL, DataFrames and Datasets Guide*. spark.apache.org/docs/1.6.0/sql-programming-guide.html. Accessed 3 Dec. 2017.

Apache Spark. *Machine Learning Library (MLlib) Guide*. spark.apache.org/docs/1.6.0/mllib-guide.html. Accessed 3 Dec. 2017.

databricks. *Data preparation*. databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3923635548890252/1357850364289680/4930913221861820/latest.html. Accessed 3 Dec. 2017.

eliasah. *Preparing data for LDA training with PySpark 1.6*. 30 May 2017, stackoverflow.com/questions/44184881/preparing-data-for-lda-training-with-pyspark-1-6. Accessed 3 Dec. 2017.

ML_TN. "How to add custom stop word list to StopWordsRemover." *Python - How to add custom stop word list to StopWordsRemover - Stack Overflow*, Stack Overflow, 29 Apr. 2017, stackoverflow.com/questions/43623400/how-to-add-custom-stop-word-list-to-stopwordsremover.

Pieters, Martijn. "Remove words of length less than 4 from string." *Python - Remove words of length less than 4 from string - Stack Overflow*, Stack Overflow, 20 June 2014, stackoverflow.com/questions/24332025/remove-words-of-length-less-than-4-from-string.

Yelp. *Yelp Dataset JSON*. www.yelp.com/dataset/documentation/json. Accessed 3 Dec. 2017.