Yelp Text To Data

**Objective:**

Measure word count frequencies under certain variables (which include geographical location, business type)

**NOTE:** This write up will exclude in-depth details regarding Python code. Check out my github if interested (I documented the code). Also, I tried to be quick to the point for my description; just ask me and I’ll try to explain further.

**Steps:**

**Step 1: Obtain the data.**

We expect Dan Silver to provide a large amount of web scraped data for Yelp soon, but we don’t have that right now.

So for now, I use my own custom web scraper to gather pieces of data just to get started.

**NOTE:** Right now, I am not worrying about sampling biases and other statistical issues. I want to build a flexible pipeline that achieves my objective. After I finish, I will make adjustments.

Substeps:

1. Go to [Yelp.com](https://www.yelp.com/), and on the search tab, search “cafe” in the “Find” tab and “ZIPCODE” in the “near” tab, where ZIPCODE is replaced the zip code that we want to analyze.
   1. **NOTE:** the businesses that we get from the search result are not necessarily in the Zip Code, but may just be near the ZIPCODE.
   2. **SOLUTIONS:** the future scraped data should have each business be labeled by its zip code, and we can do some necessary filtering.
2. Scrape the data.
3. Download results as “ZIPCODE\_NAMEOFBUSINESS.csv”

See:

[github](https://github.com/JinLi711/FAUI-Data-Scraping/tree/master/Yelp/NLP%20for%20Yelp) for an example of the content.

**Step 2: Extract relevant data from raw data.**

The scraper picks up a lot of data that we do not need at this point, but we may need for later. Right now, we are only concerned with the reviews.

Substeps:

1. Extract only the reviews from the scraped data from each csv file.
2. Combine the reviews based on zip codes.
   1. For example, if we have 60603\_FAIRGROUNDS.csv, 60603\_GODDESS.csv, and 60626\_CHARMERS.csv, we combine the reviews of 60603\_FAIRGROUNDS.csv with 60603\_GODDESS.csv.

**Step 3:Preprocess the text.**

Take the data, and normalize the words to make them suitable for comparison. Also count the word frequencies

Example:

NOTE: [“X”, “Y”, “Z”] indicates a list, “XYZ” indicates a string.

“This food is amazing! Why did it only cost me 5 dollars?”

Substeps:

1. Lowercase all the words.
   1. “this food is amazing! why did it only cost me 5 dollars?”
2. Replace numbers with [NUM].
   1. “this food is amazing! why did it only cost me [NUM] dollars?”
3. Tokenize into sentences.
   1. [ [“this”, “food”, “is”, “amazing!”], [“why”, “did”, “it”, “only”, “cost”, “me”, “[NUM]”, “dollars?”] ]
4. Replace punctuations and other non-letter characters: ()!@%^&-+\$.,?\*"#
   1. [ [“this”, “food”, “is”, “amazing”], [“why”, “did”, “it”, “only”, “cost”, “me”, “[NUM]”, “dollars”] ]
5. Lemmatize the words.
   1. [ [“this”, “food”, “is”, “amazing”], [“why”, “did”, “it”, “only”, “cost”, “me”, “[NUM]”, “dollar”] ]
   2. The only change here is from dollars to dollar. Lemmatization just refers to removing tenses. For example, “cars” becomes “car” and “flew” becomes “fly”.
6. Count word frequencies.
   1. In this case, the frequency for each word would be one.

**NOTE:** Text is very unstructured data, and we are losing a lot of information by processing it.

Examples:

1. Say we want to search for keyword “art” and its frequency
   1. Sentence: “This art is amazing. It is very inspiring.”
   2. Note that the word “it” references “art”. But our current method will only result in a word count of “art” as 1, instead of 2.
2. Misspelled words

**Step 4: Create an Embedding.**

This is probably the trickiest step, as it is going to be the hardest to understand.

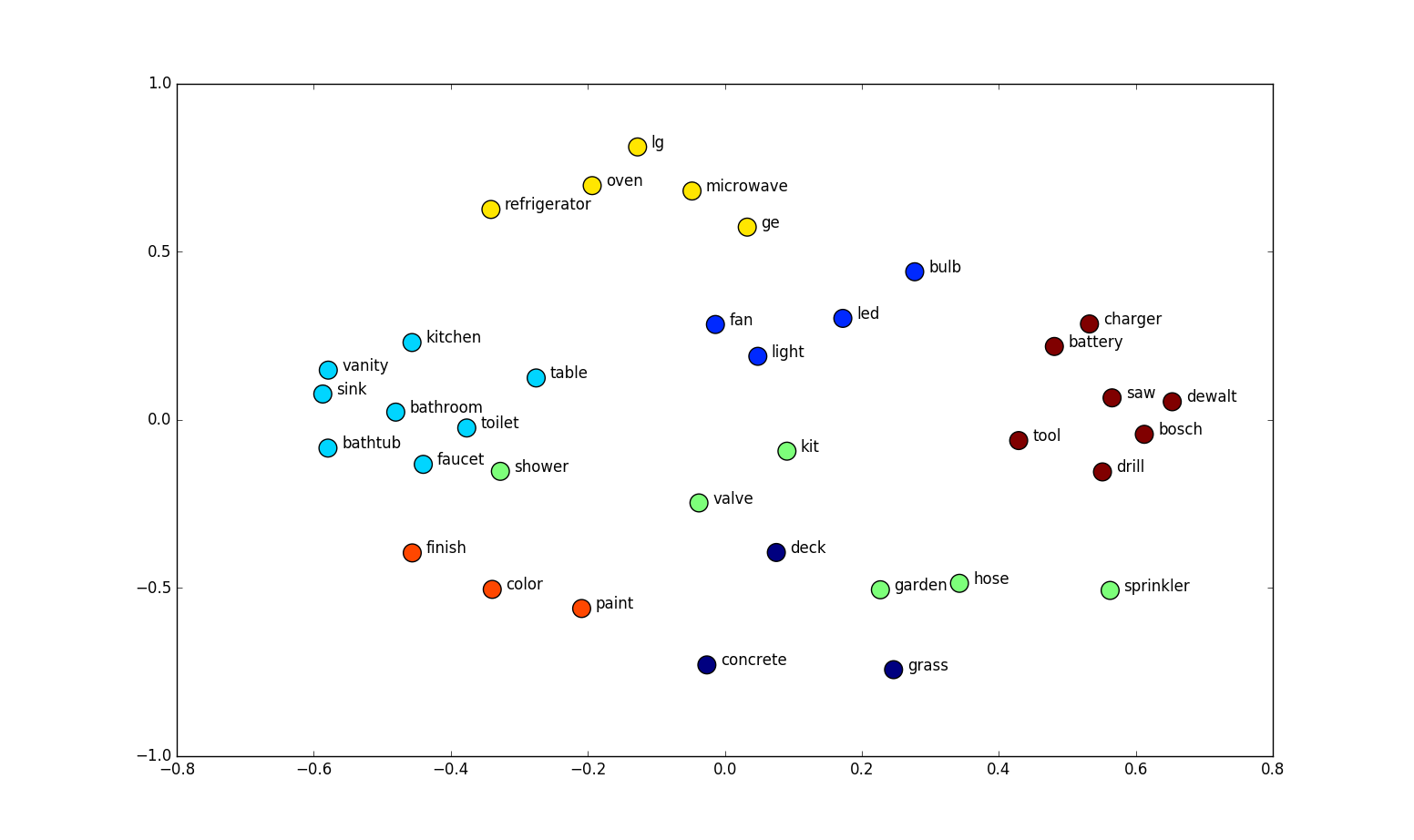
What is word embedding?:

It is essentially a way to map words to vectors in some high dimensional space. The vectors are not static; they can move around in space, but the word tied to that vector remains tied.

Why am I creating an embedding?:

Word embedding is a way to quantify words in the sense of relatedness. If done right, words that have similar meaning will have vectors that are close together. For example, the vector representation of “cheese” will be closer to the vector representation of “butter” than it would to the vector representation of “car”. Using this process, we can plot words on a scatter plot, and viewers can more easily see clustering of words.

Example:



Understanding this plot is not particularly difficult, but people do seem to have a hard time understanding that the x and y are the coordinates of the word’s vector representation. In terms of presentation, the x and y coordinate means nothing, what we want to focus on is how the words are grouped together. As is the picture, “charger”, “battery”, “tool” are clustered together.

Why not a bar graph?:

* Bar graphs are ugly to look at and hard to make any meaning out of when the x column (the categories) becomes large. We have about 30 words that are interesting to compare, and possibly even more later on. You’ll just end up with a huge bar graph that can barely fit in a page.
* Also, I dislike the fact that there is no intuition behind the ordering of the x-axis, and ordering of anything becomes more important the more there is to order.

**NOTE:** I used the [GENSIM](https://radimrehurek.com/gensim/models/word2vec.html) Word2Vec model for embedding words. I choose this because it is fast, easy to use, and reliable.

How It Works:

1. Start with assigning each word to a vector. This can be done randomly, or we can use a pre-trained model that already assigned each word to a vector.
2. Move the vector around.
   1. This is done by machine learning/ deep learning.
   2. Example sentence: “I love to pet my dog.”
      1. Take one word at a time, say the word “pet” (we’ll call this the center word).
      2. The nearby words, aka context words, would be the words around “pet”. They would be “to” and “my”. The window size, in this case, is 1, but we can change this to 2, and the new context words would be “love”, “to”, “my”, and “dog”
      3. Calculate the probability of a context word given a centre word or visa versa.
      4. Based on this probability, adjust the word vectors to maximize the probability.

Two models:

* Continuous Bag-of-Words model (CBOW)
  + predict the center word based on context words
* Skip-Gram model.
  + Predict context words based on the center word.

**NOTE:** the choice of model doesn’t really matter that much right now. I included this because it may end mattering when we have a lot more data.

My parameters for word embedding:

* CBOW
* Minimum word count of 5 (so ignore all words less than the word count)

**Step 5: Dimension Reduction.**

Word embedding usually maps words to very large dimensions (I did 100 dimensions). More dimensions make computations slower, but produces better word embedding results. But we can’t visualize 100 dimensions, so we have to reduce the dimensions to 3 or 2.

**NOTE:** Dimension reduction obviously decreases information. We want to reduce the dimension while preserving as much variance as possible (as much infomation as possible)

Algorithms for dimension reduction:

* Principled Component Analysis (PCA)
  + This is the most common, and it runs pretty quickly.
  + It basically identifies the hyperplane that lies closest to the data, and then it projects the data onto it.
* Locally Linear Embedding (LLE)
  + Measures how each point linearly relates to the closest neighbors and looks for a low-dimension representation of the points where these local relationships are best preserved.
* t-Distributed Stochastic Neighbor Embedding (t-SNE)
  + Best for visual representation, but runs slowly
  + Reduces dimensions by trying to keep similar points close and not similar points far from each other.

I chose PCA for now, but plans may change, so I included other options.

**Step 6: Filtering Words:**

We can now visually represent the words, but we have thousands of words to choose from (and that number is going to rise to tens of thousands when we have more data).

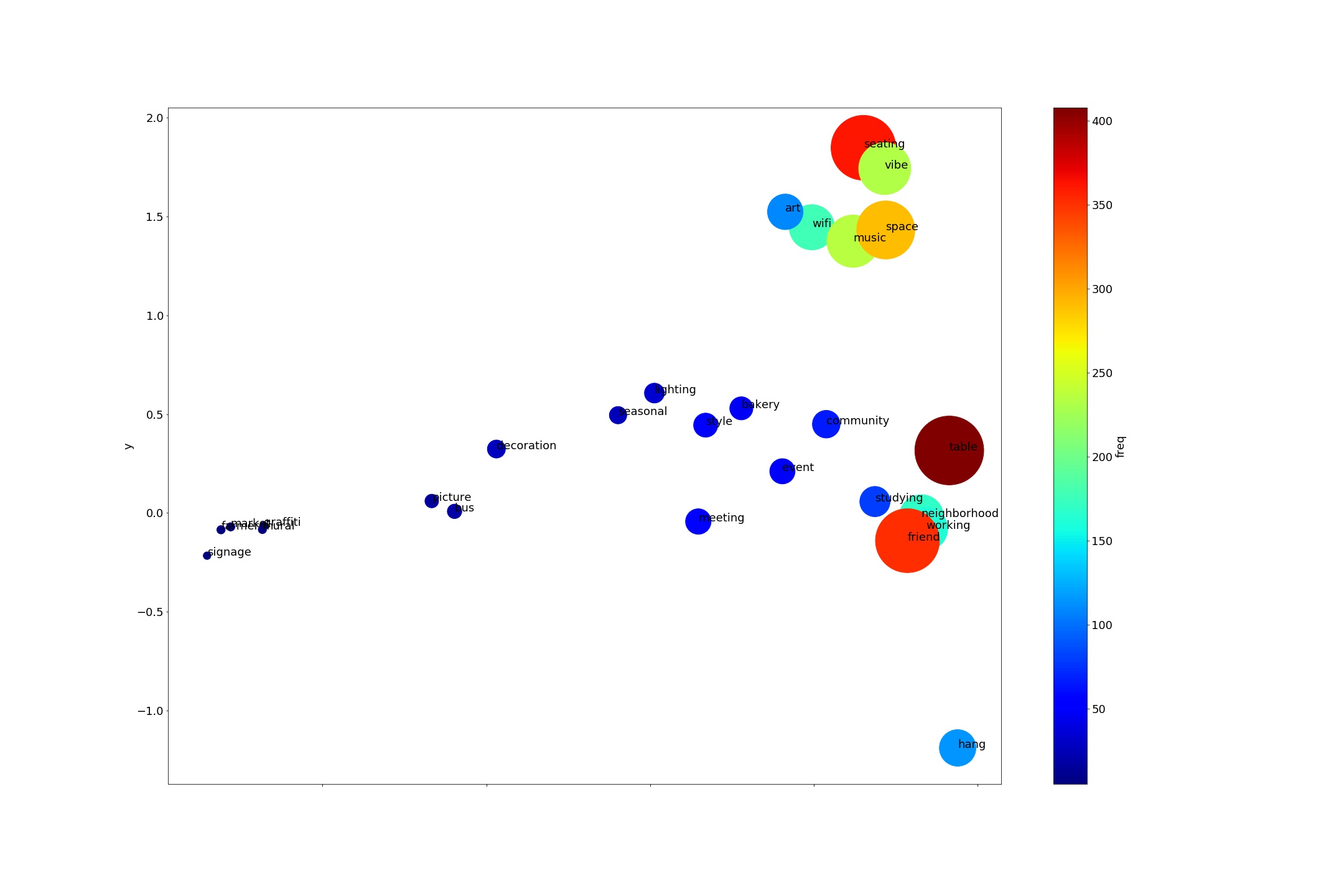
So, lets create some csv files for filtering (send me an email for the csv files, I’m not going to place them here because I’m continuously updating them).

Substeps:

1. Create a list of words to keep (immunity from filtering).
   1. bakery, decoration, style, seating, picture, lighting, community, quietness, music, space, table, outlets, wifi, vibe, event, mural, art, signage, metra, l station, bus, subway, graffiti, neighborhood, community, festival, event, farmer, market, flea  
      Seasonal, friend, studying, meeting, children, hang, dating, working
   2. This is just the list Hyesun made.
2. Use Google’s 20,000 most common words.
   1. I know we may be losing a lot of information by using this as a filter, but I don’t know any other option except hand filtering (which would take a long time)
   2. But if someone comes up with a better way, I’ll take this filter off.
3. Create my own filter list to account for words that are not interesting to consider and are not in the google list.
   1. This includes food items.

**Step 7: Create a scatter plot:**

Self explanatory. I’ll just put an example:



**Things to Work On/ Consider:**

Consider word density instead of absolute word count.

* Some places will just have a lot less words in the reviews.

Download Stanfrod GloVe

Combine the zip codes by type of neighborhood

* Should different zip codes be weighed differently?