TheYelpElites

Sentiment analysis of yelp reviews in 10 metropolitan areas

CS 123 Project

Aanya Jhaveri| Samantha Stagg | Katarina Keating | Shyamsunder Sriram

Difficulties of sentiment analysis

We could ‘classify’ words…

Positive

fantastic
incredible
superb
phenomenal
wonderful

Neutral

okay
fine
good

Negative

terrible
awful
horrendous
disgusting
bad

Unrelated

taco
toronto
place

…but contextually such outright classification is impossible!

Double Negatives

not bad
wasn’t terrible
it was good but…

Sarcasm

This restaurant has proven to be **incredible** at disappointing people with their **wonderful** cold pizza…

Project methodology

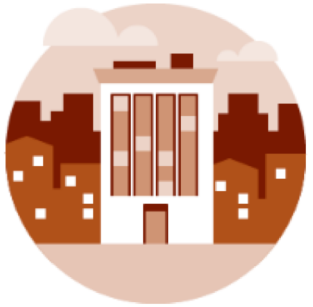
Don’t worry sentiment analysis is still possible…just needs more complex methods

Question	Approach	Hypothesis
Can we successfully bag words and emphasize words with sentiment over unrelated words?	Build Tf-idf vectors for each review	**Tf-idf vectors will emphasize sentiment words over regular words**
Can we identify positive vs. negative words, and words that are related to each other?	Create PMI word embeddings	**Sentiment words will be related to other sentiment words**
Can we make review recommendations more personalized based on text analysis?	Cosine similarity of Tf-idf vectors	**Reviews with similar words will yield similar results**

5GB size Yelp dataset

6,685,900 reviews

192,609 businesses



10 metropolitan areas

Dataset source: https://www.yelp.com/dataset/download

Approach 1: Emphasizing words with sentiment

1

Created vocabulary

Looped over dataset and counted all unique 'words' uttered and took the top 20k words uttered.

$$a_{11} \quad a_{12} \quad \cdots \quad a_{1m}$$

Each index corresponds to a word in a review. If the word in the vocabulary does not exist, the index would be zero.

2

Created Tf-idf vectors

The values of the tfidf vectors would be populated using this formula

$$Tf(w) * IDF(w)$$

where $Tf(w) = \frac{\#\ times\ word\ w\ appears\ in\ a\ review}{total\ \ of\ words\ in\ a\ review}$

and $Idf(w) = \ln \frac{Total\ \#\ of\ reviews}{Total\ \#\ of\ reviews\ that\ contain\ word\ w}$

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix}_{n \times m}$$

This would be a **sparse matrix** of tfidf

Does Tfidf it work? Good enough.

Oh no! There's a double negative!

"Azzip Pizza is literally **SO good**. I have been there so many times and each time I get a \"Big Zip\" for $8.22. The place is **amazing** and the pizza cooks in about 2 minutes. It is the build your own concept and I have **never** had a **bad** experience there. Also, I go there about 4 times a week.. Don't worry I hit the gym, sometimes.."

Results

Picks up positive sentiment

Wordcloud of words with high scores among 5 star reviews in sample dataset in South Carolina

Reliable accuracy

83% accuracy in recognizing 4+ star reviews after running logistic regression in sample dataset of 32000 reviews.

Approach 2: Word embeddings

1

Loop through each review and create word embedding pairs of n x n matrix of counts where n is size of vocab

If this is a sample review…

I think Thai 55 is trash.

then the word embedding pairs for thai within a defined window of 2 is
(I, think), (I, Thai), (Thai, 55), (Thai, is).

Key points:

- A window size of 1, is pretty uninformative, so we made ours 5.
- Removed stopwords and only found pairs among vocabulary

2

Applying PMI word embedding formula And creating matrix of word embeddings

After 2 pages of disgusting math derivations we reach this formula…

$$M_{ij} = v_{w_i}^T v_{w_j} = \log\left(\frac{N^p(w_i, w_j) \cdot N(\mathcal{S}^p)}{N^p(w_i) \cdot N^p(w_j)}\right)$$

And apply formula to **square** matrix of counts.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix}_{n \times m}$$

Take Euclidean distances of all rows, and the shortest ones correspond to related words

Approach 3: Recommendation engine

1. **Filtered out unique users of each city and the reviews they wrote**

2. **Filtered out positive reviews and took cosine similarities**

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

3. **Recommended places corresponding to indices with highest similarity scores**

$$\begin{array}{c} \\ \text{doc}_1 \\ \text{doc}_2 \\ \vdots \\ \text{doc}_m \end{array} \begin{array}{c} \begin{array}{cccc} \text{term}_1 & \text{term}_2 & \cdots & \text{term}_n \end{array} \\ \begin{bmatrix} tfidf_{11} & tfidf_{12} & \cdots & tfidf_{1n} \\ tfidf_{21} & & & tfidf_{2n} \\ \vdots & & & \vdots \\ tfidf_{m1} & tfidf_{m1} & \cdots & tfidf_{mn} \end{bmatrix} \end{array} \quad \theta \quad \begin{array}{c} \text{doc}_{\text{new}} \\ \begin{bmatrix} tfidf_1 \\ tfidf_2 \\ \vdots \\ tfidf_n \end{bmatrix} \end{array} = \begin{array}{c} \\ \text{doc}_1 \\ \text{doc}_2 \\ \vdots \\ \text{doc}_m \end{array} \begin{array}{c} \text{doc}_{\text{new}} \\ \begin{bmatrix} sim_1 \\ sim_2 \\ \vdots \\ sim_n \end{bmatrix} \end{array}$$

Tfidf matrix

Tfidf vector of liked restaurant

Indices of most similar restaurants

Challenges faced

[gcloud]:

•authentication errors
•setup
•running on mac OS and not on VM
•streaming of final files:

- Attempted to create tf/idf vectors with gcloud and mapreduce as it was a lot of counting. This created quite an issue as while the runtime was no more than a few minutes on gcloud, streaming the tf/idf vector output from google cloud took over 9 hours. This would often timeout which prompted our switch to MPI.

[MPI]:

•Connecting to google cloud

[MRJOB]:

• Creating review ID to index dictionary:

- Using runner file does not work the same way when running on a file that is stored in the cloud, had to alter approach

[file sizes]

Some of the files were too big to push to git (>100MB)

Big data approaches

[mapreduce]

Used to prepare relevant .json files for sentiment analysis and word embeddings

•Counting review frequency of words (make a set of all words used, then count each word & how many reviews it appears in)

- local runtime: > 4 hours
- gcloud runtime with 5 clusters: 10 minutes
- output of this was also used to create vocabulary in linear time

•Creating dictionaries

- Review ID to review text
- User ID to reviews written
- Review ID to index

[MPI]:

Used to create tf/idf vectors

•csv file of review text converted to numpy array (split by review)

•Numpy array scattered to instances, which calculate tf-idf vectors for each word and put them in a matrix

Results summary

…time for a demonstration