Worst & Best Noun and Name

In [1]:
```
# Library Packages
import regex as re
import itertools as it
import spacy

%run libraries.py
from __future__ import division

# Settings
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
seed = 7
np.random.seed(seed)

import warnings
warnings.filterwarnings('ignore')

def front(self, n):
    return self.iloc[:, :n]

def back(self, n):
    return self.iloc[:, -n:]

# Like normalization, standardization can be useful, and even require
# machine learning algorithms when your time series data has input va

from sklearn.preprocessing import StandardScaler

def Standardisation(df):
    listed = list(df)
    scaler = StandardScaler()
    scaled = scaler.fit_transform(df)
    df = pd.DataFrame(scaled)
    df.columns = listed
    return df



np.set_printoptions(threshold=np.nan)

!free -h
```

```
/bin/sh: free: command not found
```

In [2]:
```
yelp = pd.read_csv("bjs-restaurant-sanbruno.csv")

yelp["date"] = yelp["date"].apply(lambda x: x[:10])
yelp["date"] = yelp["date"].apply(lambda x: x[:-1] if x[-1]=="\\" els
yelp["date"] = yelp["date"].apply(lambda x: x[:-2] if x[-1]=="n" else

from datetime import datetime
from dateutil.parser import parse

yelp['date'] = yelp['date'].apply(lambda x: parse(x))
```

In [5]:
```
yelp.head()
```

Out[5]:

	Username	location	friend_count	review_count	photo_count	date	rating	
0	Finau F.	Millbrae CA	152	38	50.0	2017-09-15	3.0	This i defini place can r i...
1	Tanya F.	Fairfield CA	82	20	33.0	2017-09-10	1.0	Serio worst restau ever:S servi.
2	Michelle S.	San Francisco CA	200	143	453.0	2017-09-09	2.0	Yikes venue is coo decor
3	Danielle S.	Pacifica CA	246	33	50.0	2017-08-31	4.0	Did y know BJ's l ton o prom
4	Teena N.	San Francisco CA	35	235	1076.0	2017-08-26	3.0	Seco here time in. +S

In [10]:
```
from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA

yelp["positive"] = 0
yelp["compound"] =  0.0
yelp["negative"] = 0
yelp["neutral"] =  0

analyzer = SIA()
for sentence, row in zip(yelp["review"], list(range(yelp.shape[0]))):
    vs = analyzer.polarity_scores(sentence)
    yelp["compound"][row] = float(vs["compound"])
    if vs["compound"] <-0.5:
        yelp["negative"][row] = 1
    elif vs["compound"] >0.5:
        yelp["positive"][row] = 1
    else:
        yelp["neutral"][row] = 1
    #print("{:-<65} {}".format(sentence, str(vs)))
```

In [17]:
```
worst = yelp[(yelp["rating"]==1) & (yelp["compound"]<-.95) ]
worst = worst.sort_values("date", ascending=False).head(10).reset_inc

best = yelp[(yelp["rating"]==5) & (yelp["compound"]>.95) ]
best = best.sort_values("date", ascending=False).head(10).reset_index
```

In [387]:

```
# Entity Extraction From Review
import itertools as it
import spacy

nlp = spacy.load('en')

sample_review = ""
for i in best["review"]:
    sample_review = sample_review + str(i)

#print(sample_review)

len(sample_review)

sample_review = sample_review.replace("\\", "")

parsed_review = nlp(sample_review)

#print(parsed_review)

token_text = [token.orth_ for token in parsed_review]
token_pos = [token.pos_ for token in parsed_review]

df = pd.DataFrame({'token_text':token_text, 'part_of_speech':token_pc

# Unigrams
import nltk
from nltk import word_tokenize
from nltk.util import ngrams
from collections import Counter

token = nltk.word_tokenize(str(parsed_review))
grams = ngrams(token,1)

dra = Counter(grams)

t = pd.DataFrame()
f = pd.DataFrame(list(dra.keys()))

f = f[0]

t["name"] = f
t["count"] = list(dra.values())


df = df.drop_duplicates()
r = pd.merge(t, df, left_on=["name"], right_on=["token_text"], how="l
r = r.drop("token_text",axis=1)
r.columns = ["name","count","pos"]
```

In [388]:
```
r[r["pos"]=="NOUN"].sort_values("count",ascending=False)
```

Out[388]:

	name	count	pos
131	food	12	NOUN
159	experience	8	NOUN
180	visit	5	NOUN
141	place	5	NOUN
73	service	5	NOUN
155	night	4	NOUN
319	rib	4	NOUN
147	manager	4	NOUN
260	group	3	NOUN
371	love	3	NOUN

In [389]:
```
r[r["pos"]=="NOUN"].sort_values("count",ascending=False)
```

Out[389]:

	name	count	pos
131	food	12	NOUN
159	experience	8	NOUN
180	visit	5	NOUN
141	place	5	NOUN
73	service	5	NOUN
155	night	4	NOUN
319	rib	4	NOUN
147	manager	4	NOUN
260	group	3	NOUN
371	love	3	NOUN

In [391]:
```
for num, sentence in enumerate(parsed_review.sents):
    print('Sentence {}:'.format(num + 1))
    print(sentence)
    print("")



ent = []
lab = []
```

```
    []


for num, entity in enumerate(parsed_review.ents):
    ent.append(entity[0])
    lab.append(entity.label_)

ent_df = pd.DataFrame()
ent_df["entity"] = ent
ent_df["label"] = lab
rab = ent_df
# for num, entity in enumerate(parsed_review.ents):
#     ent_df["entity"][num] = entity
#     ent_df["label"][num] = entity.label_


ent_df["entity"] = ent_df["entity"].astype(str)

ent_df = pd.merge(ent_df.groupby("entity").count().reset_index(), ent

ent_df.columns = ["entity","count","type"]

from difflib import SequenceMatcher

def similar(a, b):
    return SequenceMatcher(None, a, b).ratio()


vent = ent_df[ent_df["type"].isin(["GPE","PERSON","ORG"])]["entity"]

import jellyfish
from fuzzywuzzy import fuzz
from fuzzywuzzy import process


dar = []
sar = []
kar = []
jar = []
lev = []


for i in vent:
    for r in vent:
        dar.append(i)
        sar.append(r)
        jar.append(jellyfish.jaro_distance(i,r))
        kar.append(similar(i,r))
        lev.append(jellyfish.levenshtein_distance(i,r))

sos = pd.DataFrame()
sos["original"] = dar
sos["match"] = sar
sos["percentage"] = kar
sos["distance"] = jar
```

```
sos["leven"] = lev
sos["together"] = (sos["percentage"]  + (sos["distance"])/2)*(1/sos["
                # Including leven is important because it also counts
                # of characters, maybe change below, 0.2 to 0.3 if fu

sos = sos[(sos["together"]<1.0)&(sos["together"]>0.4)].reset_index()

sos["count_original"] = 0
sos["count_contender"] = 0
for i, c, r in zip(sos["original"], sos["match"], list(range(sos.shap
    da = np.where(ent_df["entity"]==i, ent_df["count"],np.nan )
    x = da[~np.isnan(da)]
    sos["count_original"][r] = x

    da = np.where(ent_df["entity"]==c, ent_df["count"],np.nan )
    x = da[~np.isnan(da)]
    sos["count_contender"][r] = x


sos

dar = np.where(sos["count_original"]>=sos["count_contender"],sos["ori
sos["final"] = dar

cas = sos[["match","final"]]


for match, final in zip(cas["match"],cas["final"]):
    print(match)
    ent_df['entity'] = ent_df.entity.replace([str(match)],[str(final)

res = pd.DataFrame()
res["start"] = sos["original"]

ent_df = pd.merge(ent_df.groupby("entity").sum().reset_index(), ent_d

ent_df["count"] = ent_df["count_x"]

ent_df = ent_df[["entity","count","type"]].sort_values("count",ascend
```

```
Sentence 1:
This is a long overdue review.

Sentence 2:
First and foremost  out of all the BJ's restaurants that I've been
to  this branch is my favorite.

Sentence 3:
The hosts are very friendly here and always have a smile on their
faces when you walk in.

Sentence 4:
I've also never had a problem with any of the servers here.
```

```
Sentence 5:
My favorite server is Katrina B.!

Sentence 6:
If you go to this branch  and have her as your server  you will ge
t above and beyond service
```

In [394]:
```
ent_df[ent_df["type"].isin(["ORG","PERSON","GPE"])]
# If a person uses the word twice, there is probably a good reasons,
```

Out[394]:

	entity	count	type
6	BJ	3	PERSON
8	Bjs	2	PERSON
11	Efraim	2	PERSON
12	Eugene	2	PERSON
10	Debbie	2	PERSON
0		1	ORG
14	Katrina	1	PERSON
16	Rochelle	1	GPE
15	Norm	1	PERSON
9	Cheers	1	ORG
7	Billy	1	ORG

In [34]:
```
token_text = [token.orth_ for token in parsed_review]
token_pos = [token.pos_ for token in parsed_review]
```

In [42]:
```
df = pd.DataFrame({'token_text':token_text, 'part_of_speech':token_pc
```

In [43]:
```
df
```

Out[43]:

	part_of_speech	token_text
0	DET	This
1	VERB	is
2	ADV	definitely
3	DET	a
4	NOUN	place
5	PRON	you
6	VERB	can
7	VERB	resort
8	ADP	to
9	ADP	if

In [46]:
```
# Zip is different for Python 3 It is an itterator in three so have t

token_lemma = [token.lemma_ for token in parsed_review]
token_shape = [token.shape_ for token in parsed_review]

pd.DataFrame(list(zip(token_text, token_lemma, token_shape))[:],
             columns=['token_text', 'token_lemma', 'token_shape'])
```

Out[46]:

	token_text	token_lemma	token_shape
0	This	this	Xxxx
1	is	be	xx
2	definitely	definitely	xxxx
3	a	a	x
4	place	place	xxxx
5	you	-PRON-	xxx
6	can	can	xxx
7	resort	resort	xxxx
8	to	to	xx
9	if	if	xx

In [49]:

```
token_entity_type = [token.ent_type_ for token in parsed_review]
token_entity_iob = [token.ent_iob_ for token in parsed_review]

pd.DataFrame(list(zip(token_text, token_entity_type, token_entity_iob
             columns=['token_text', 'entity_type', 'inside_outside_be
```

Out[49]:

	token_text	entity_type	inside_outside_begin
0	This		O
1	is		O
2	definitely		O
3	a		O
4	place		O
5	you		O
6	can		O
7	resort		O
8	to		O
9	if		O

In [50]:
```
token_attributes = [(token.orth_,
                     token.prob,
                     token.is_stop,
                     token.is_punct,
                     token.is_space,
                     token.like_num,
                     token.is_oov)
                    for token in parsed_review]

df = pd.DataFrame(token_attributes,
                  columns=['text',
                           'log_probability',
                           'stop?',
                           'punctuation?',
                           'whitespace?',
                           'number?',
                           'out of vocab.?'])

df.loc[:, 'stop?':'out of vocab.?'] = (df.loc[:, 'stop?':'out of voca
                                       .applymap(lambda x: u'Yes' if

df
```

Out[50]:

	text	log_probability	stop?	punctuation?	whitespace?	number?	ou voca
0	This	-6.785319	Yes				
1	is	-4.329765	Yes				
2	definitely	-9.063265					
3	a	-3.983075	Yes				
4	place	-8.045827					
5	you	-4.547973	Yes				
6	can	-5.913871	Yes				
7	resort	-11.149202					
8	to	-3.838520	Yes				

In [52]:
```
# This part of the analysis is different to the on that I am used to.

from gensim.models import Phrases
from gensim.models.word2vec import LineSentence
```

In [53]:
```
def punct_space(token):
    """
    helper function to eliminate tokens
    that are pure punctuation or whitespace
    """

    return token.is_punct or token.is_space

def line_review(filename):
    """
    generator function to read in reviews from the file
    and un-escape the original line breaks in the text
    """

    with codecs.open(filename, encoding='utf_8') as f:
        for review in f:
            yield review.replace('\\n', '\n')

def lemmatized_sentence_corpus(filename):
    """
    generator function to use spaCy to parse reviews,
    lemmatize the text, and yield sentences
    """

    for parsed_review in nlp.pipe(line_review(filename),
                                  batch_size=10000, n_threads=4):

        for sent in parsed_review.sents:
            yield u' '.join([token.lemma_ for token in sent
                             if not punct_space(token)])
```

In [85]:
```
# Writing all the reviews to a file, each item in the list to a new l
# Import os

thefile = open('test.txt', 'w')

for item in yelp["review"].tolist():
  thefile.write("%s\n" % item)
```

In [86]:
```
#intermediate_directory = os.path.join('..', 'intermediate')

unigram_sentences_filepath = os.path.join('uni_test.txt')
```

In [87]:
```
%%time

import os
import codecs


# this is a bit time consuming - make the if statement True
# if you want to execute data prep yourself.
if 0 == 0:

    with codecs.open(unigram_sentences_filepath, 'w', encoding='utf_8
        for sentence in lemmatized_sentence_corpus("test.txt"):
            f.write(sentence + '\n')
```

```
CPU times: user 14.7 s, sys: 1.59 s, total: 16.3 s
Wall time: 16.6 s
```

In [88]:
```
unigram_sentences = LineSentence(unigram_sentences_filepath)
```

In [89]:
```
for unigram_sentence in it.islice(unigram_sentences, 10, 20):
    print(u' '.join(unigram_sentence))
    print(u'')
    # Once you have a few more companies the above, bi,tri rams will
```

```
but again yike

food come out super slow

-PRON- take so long -PRON- almost ask and -PRON- never ask

the server also put -PRON- appetizer in as entree so -PRON- apps com
e out at the same time as -PRON- entree which be pretty frustrating

also two of -PRON- dish be straight up cold

like not kinda cold like cold

like -PRON- sweet potato fry feel like -PRON- would be sit out for 3
0 minute

-PRON- send -PRON- back and the server do not seem surprised or apol
ogetic

seem kind of like the norm

the sweet potato fry be good -PRON- fried calamari be okay
```

```
# This next one is more interesting, it is topic modelling with LDA:

LDA is fully unsupervised. The topics are "discovered" automatically
from the data by trying to maximize the likelihood of observing the
documents in your corpus, given the modeling assumptions. They are
expected to capture some latent structure and organization within the
documents, and often have a meaningful human interpretation for
people familiar with the subject material.
```

In [92]:

```
from gensim.corpora import Dictionary, MmCorpus
from gensim.models.ldamulticore import LdaMulticore

import pyLDAvis
import pyLDAvis.gensim
import warnings
import cPickle as pickle
```

```
---------------------------------------------------------------------
-------
ModuleNotFoundError                       Traceback (most recent cal
l last)
<ipython-input-92-a7d8e4582c30> in <module>()
      2 from gensim.models.ldamulticore import LdaMulticore
      3
----> 4 import pyLDAvis
      5 import pyLDAvis.gensim
      6 import warnings

ModuleNotFoundError: No module named 'pyLDAvis'
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: