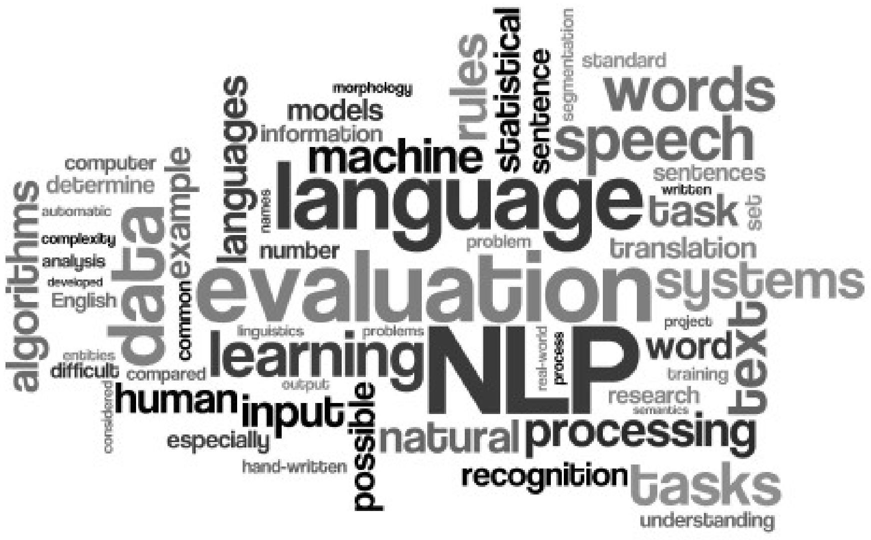
Collocations — identifying phrases that act like single words in Natural Language Processing

Nicha Ruchirawat Mar 16, 2018 · 6 min read



What is a collocation? It is a phrase consisting of more than one word but these words more commonly co-occur in a given context than its individual word parts. For example, in a set of hospital related documents, the phrase ‘CT scan’ is more likely to co-occur than do ‘CT’ and ‘scan’ individually. ‘CT scan’ is also a meaningful phrase.

The two most common types of collocation are bigrams and trigrams. Bigrams are two adjacent words, such as 'CT scan', 'machine learning', or 'social media'. Trigrams are three adjacent words, such as 'out of business', or 'Proctor and Gamble'.

If we choose any adjacent words as our bigram or trigrams, we will not get meaningful phrases. For example, the sentence 'He uses social media' contains bigrams: 'He uses', 'uses social', 'social media'. 'He uses' and 'uses social' do not mean anything, while 'social media' is a meaningful bigram. How do we make good selections for collocations? Co-occurences may not be sufficient as phrases such as 'of the' may co-occur frequently, but are not meaningful. We will explore several methods to filter out the most meaningful collocations: ***frequency counting***, ***Pointwise Mutual Information (PMI)***, and ***hypothesis testing (t-test and chi-square)***.

Some uses for collocation identification are:
a) Keyword extraction: identifying the most relevant keywords in documents to assess what aspects are most talked about
b) Bigrams/Trigrams can be concatenated (e.g. social media -> social_media) and counted as one word to improve insights analysis, topic modeling, and create more meaningful features for predictive models in NLP problems

We will use hotels reviews data that can be downloaded here.

Before applying different methods to choose the best bigrams/trigrams, we need to preprocess the reviews text. Get the code to clean the text here.

We will then use NLTK's tools to generate all possible bigrams and trigrams:

```
import nltk

bigrams = nltk.collocations.BigramAssocMeasures()
trigrams = nltk.collocations.TrigramAssocMeasures()

bigramFinder =
nltk.collocations.BigramCollocationFinder.from_words(tokens)
trigramFinder =
nltk.collocations.TrigramCollocationFinder.from_words(tokens)
```

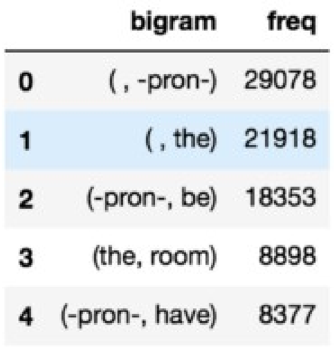
Methods to Rank Collocations

1. **Counting frequencies of adjacent words with part of speech filters:**

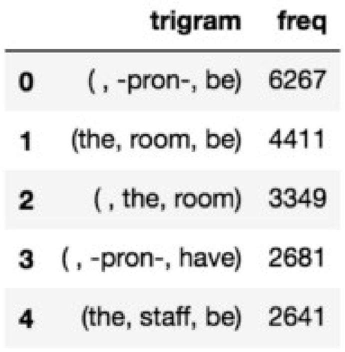
The simplest method is to rank the most frequent bigrams or trigrams:

```
#bigrams
bigram_freq = bigramFinder.ngram_fd.items()
bigramFreqTable = pd.DataFrame(list(bigram_freq), columns=
['bigram','freq']).sort_values(by='freq', ascending=False)

#trigrams
trigram_freq = trigramFinder.ngram_fd.items()
trigramFreqTable = pd.DataFrame(list(trigram_freq), columns=
['trigram','freq']).sort_values(by='freq', ascending=False)
```

However, a common issue with this is adjacent spaces, stop words, articles, prepositions or pronouns are common and are not meaningful:



	bigram	freq
0	(, -pron-)	29078
1	(, the)	21918
2	(-pron-, be)	18353
3	(the, room)	8898
4	(-pron-, have)	8377



	trigram	freq
0	(, -pron-, be)	6267
1	(the, room, be)	4411
2	(, the, room)	3349
3	(, -pron-, have)	2681
4	(the, staff, be)	2641

To fix this, we filter out for collocations not containing stop words and filter for only the following structures:

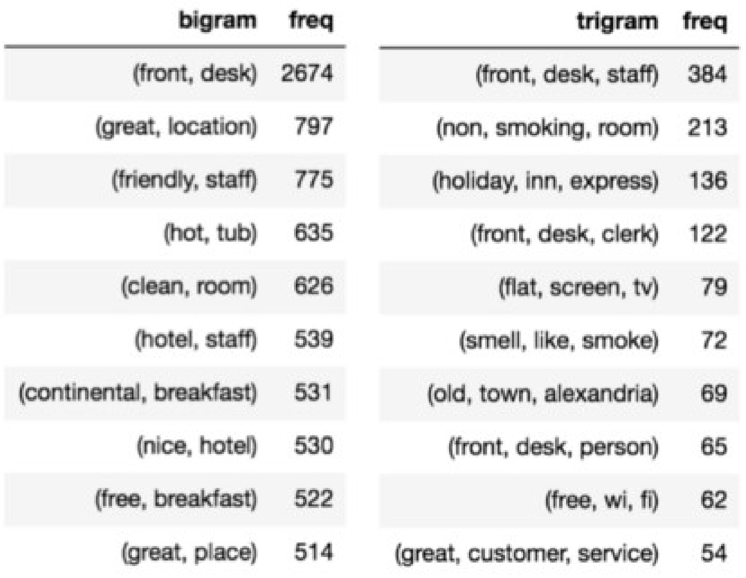
1. **Bigrams:** (Noun, Noun), (Adjective, Noun)
2. **Trigrams:** (Adjective/Noun, Anything, Adjective/Noun)

This is a common structure used in literature and generally works well.

```
#get english stopwords
en_stopwords = set(stopwords.words('english'))


#function to filter for ADJ/NN bigrams
def rightTypes(ngram):
    if '-pron-' in ngram or 't' in ngram:
        return False
    for word in ngram:
        if word in en_stopwords or word.isspace():
            return False
    acceptable_types = ('JJ', 'JJR', 'JJS', 'NN', 'NNS', 'NNP', 
'NNPS')
    second_type = ('NN', 'NNS', 'NNP', 'NNPS')
    tags = nltk.pos_tag(ngram)
    if tags[0][1] in acceptable_types and tags[1][1] in second_type:
        return True
    else:
        return False


#filter bigrams
filtered_bi = bigramFreqTable[bigramFreqTable.bigram.map(lambda x:
```

```
rightTypes(x))]

#function to filter for trigrams
def rightTypesTri(ngram):
    if '-pron-' in ngram or 't' in ngram:
        return False
    for word in ngram:
        if word in en_stopwords or word.isspace():
            return False
    first_type = ('JJ', 'JJR', 'JJS', 'NN', 'NNS', 'NNP', 'NNPS')
    third_type = ('JJ', 'JJR', 'JJS', 'NN', 'NNS', 'NNP', 'NNPS')
    tags = nltk.pos_tag(ngram)
    if tags[0][1] in first_type and tags[2][1] in third_type:
        return True
    else:
        return False

#filter trigrams
filtered_tri = trigramFreqTable[trigramFreqTable.trigram.map(lambda
x: rightTypesTri(x))]
```

Results after filtering:



bigram	freq
(front, desk)	2674
(great, location)	797
(friendly, staff)	775
(hot, tub)	635
(clean, room)	626
(hotel, staff)	539
(continental, breakfast)	531
(nice, hotel)	530
(free, breakfast)	522
(great, place)	514

trigram	freq
(front, desk, staff)	384
(non, smoking, room)	213
(holiday, inn, express)	136
(front, desk, clerk)	122
(flat, screen, tv)	79
(smell, like, smoke)	72
(old, town, alexandria)	69
(front, desk, person)	65
(free, wi, fi)	62
(great, customer, service)	54

Much better!

2. Pointwise Mutual Information

The Pointwise Mutual Information (PMI) score for bigrams is:

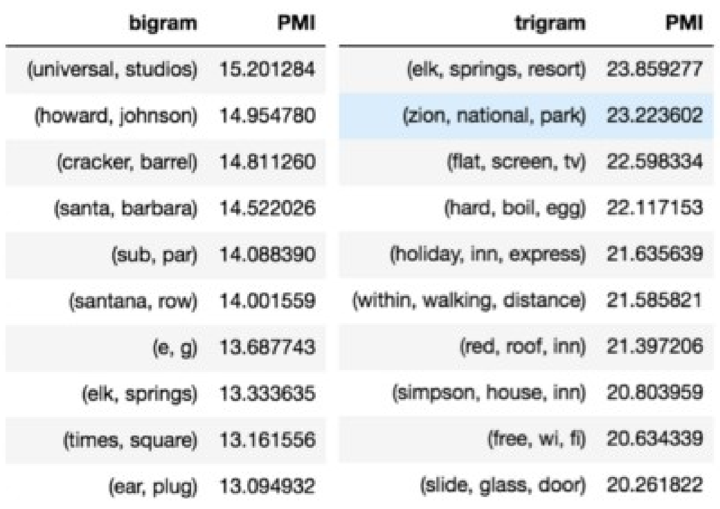
$$PMI(w^1, w^2) = log_2 \frac{P(w^1, w^2)}{P(w^1)P(w^2)}$$

For trigrams:

$$PMI(w^1, w^2, w^3) = log_2 \frac{P(w^1, w^2, w^3)}{P(w^1)P(w^2)P(w^3)}$$

The main intuition is that it measures how much more likely the words co-occur than if they were independent. However, it is very sensitive to rare combination of words. For example, if a random bigram 'abc xyz' appears, and neither 'abc' nor 'xyz' appeared anywhere else in the text, 'abc xyz' will be identified as highly significant bigram when it could just be a random misspelling or a phrase too rare to generalize as a bigram. Therefore, this method is often used with a frequency filter.

```
#filter for only those with more than 20 occurences
bigramFinder.apply_freq_filter(20)
trigramFinder.apply_freq_filter(20)

bigramPMITable =
pd.DataFrame(list(bigramFinder.score_ngrams(bigrams.pmi)), columns=
['bigram','PMI']).sort_values(by='PMI', ascending=False)

trigramPMITable =
pd.DataFrame(list(trigramFinder.score_ngrams(trigrams.pmi)),
columns=['trigram','PMI']).sort_values(by='PMI', ascending=False)
```



bigram	PMI	trigram	PMI
(universal, studios)	15.201284	(elk, springs, resort)	23.859277
(howard, johnson)	14.954780	(zion, national, park)	23.223602
(cracker, barrel)	14.811260	(flat, screen, tv)	22.598334
(santa, barbara)	14.522026	(hard, boil, egg)	22.117153
(sub, par)	14.088390	(holiday, inn, express)	21.635639
(santana, row)	14.001559	(within, walking, distance)	21.585821
(e, g)	13.687743	(red, roof, inn)	21.397206
(elk, springs)	13.333635	(simpson, house, inn)	20.803959
(times, square)	13.161556	(free, wi, fi)	20.634339
(ear, plug)	13.094932	(slide, glass, door)	20.261822

We can see that PMI picks up bigrams and trigrams that consist of words that should co-occur together.

3. Hypothesis Testing

a. t-test:

Consider if we have a corpus with N words, and social and media have word counts C(social) and C(media) respectively. Assuming null hypothesis with social and media being independent:

$$\begin{gathered}H_0 : \text{'social media' occurs with probability :}\\ \mu = P(social)P(media) = \frac{C(social)}{N} * \frac{C(media)}{N}\\ H_a : \text{'social media' does not occur with expected probability} \mu\end{gathered}$$

The test statistic is:

$$t = \frac{\bar{x} - \mu}{\sqrt{\frac{s^2}{N}}}$$

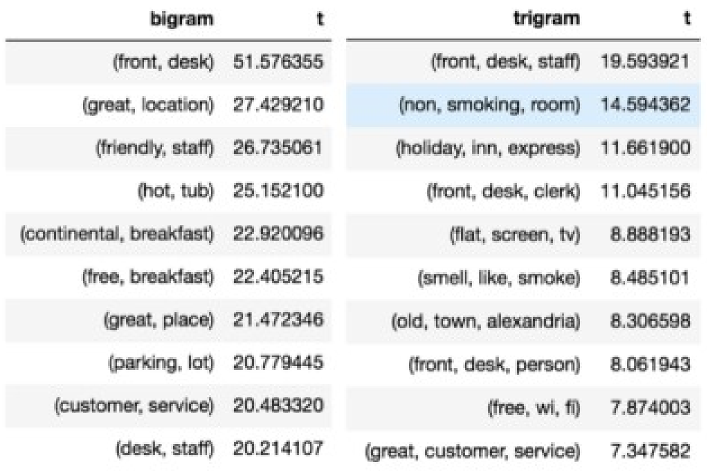
$$\bar{x} = \frac{C(\text{social media})}{N}$$

$$\text{for Bernoulli trial: } s^2 = p(1-p) \approx p \approx \bar{x}$$

However, the same problem occurs where pairs with prepositions, pronouns, articles etc. come up as most significant. Therefore, we need to apply the same filters from 1.

```
bigramTtable =
pd.DataFrame(list(bigramFinder.score_ngrams(bigrams.student_t)),
columns=['bigram','t']).sort_values(by='t', ascending=False)
trigramTtable =
pd.DataFrame(list(trigramFinder.score_ngrams(trigrams.student_t)),
columns=['trigram','t']).sort_values(by='t', ascending=False)

#filters
filteredT_bi = bigramTtable[bigramTtable.bigram.map(lambda x:
rightTypes(x))]
filteredT_tri = trigramTtable[trigramTtable.trigram.map(lambda x:
rightTypesTri(x))]
```

Results are similar to the frequency count technique from 1.:

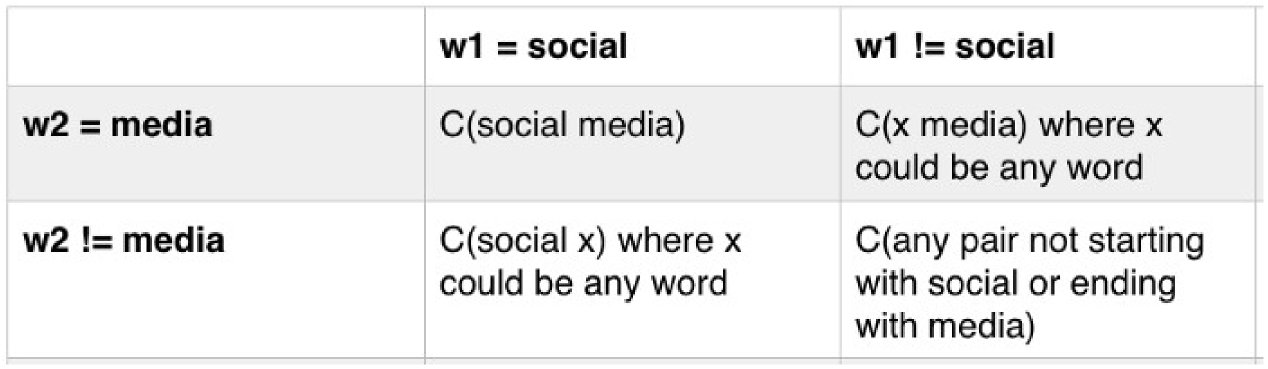


bigram	t	trigram	t
(front, desk)	51.576355	(front, desk, staff)	19.593921
(great, location)	27.429210	(non, smoking, room)	14.594362
(friendly, staff)	26.735061	(holiday, inn, express)	11.661900
(hot, tub)	25.152100	(front, desk, clerk)	11.045156
(continental, breakfast)	22.920096	(flat, screen, tv)	8.888193
(free, breakfast)	22.405215	(smell, like, smoke)	8.485101
(great, place)	21.472346	(old, town, alexandria)	8.306598
(parking, lot)	20.779445	(front, desk, person)	8.061943
(customer, service)	20.483320	(free, wi, fi)	7.874003
(desk, staff)	20.214107	(great, customer, service)	7.347582

T-test has been criticized as it assumes normal distribution. Therefore, we will also look into the chi-square test.

b. chi-square test

First, we compute a table like below for each word pair:



	w1 = social	w1 != social
w2 = media	C(social media)	C(x media) where x could be any word
w2 != media	C(social x) where x could be any word	C(any pair not starting with social or ending with media)

The chi-square test assumes in the null hypothesis that words are independent, just like in t-test. The chi-square test statistic is computed as:

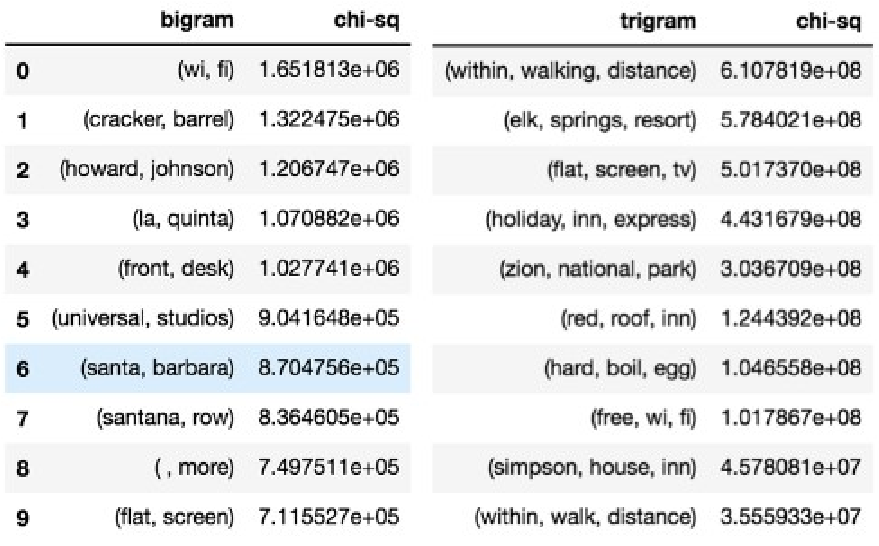
$$\chi^2 = \sum_{i,j} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

i and j are over all rows and columns of the table

O_{ij} is the observed value in the table for row i, column j

E_{ij} is the expected value if the words are independent, eg:
$E(\text{social media}) = \frac{C(\text{social})}{N} * \frac{C(\text{media})}{N} * N$
where N is total number of words in corpus.

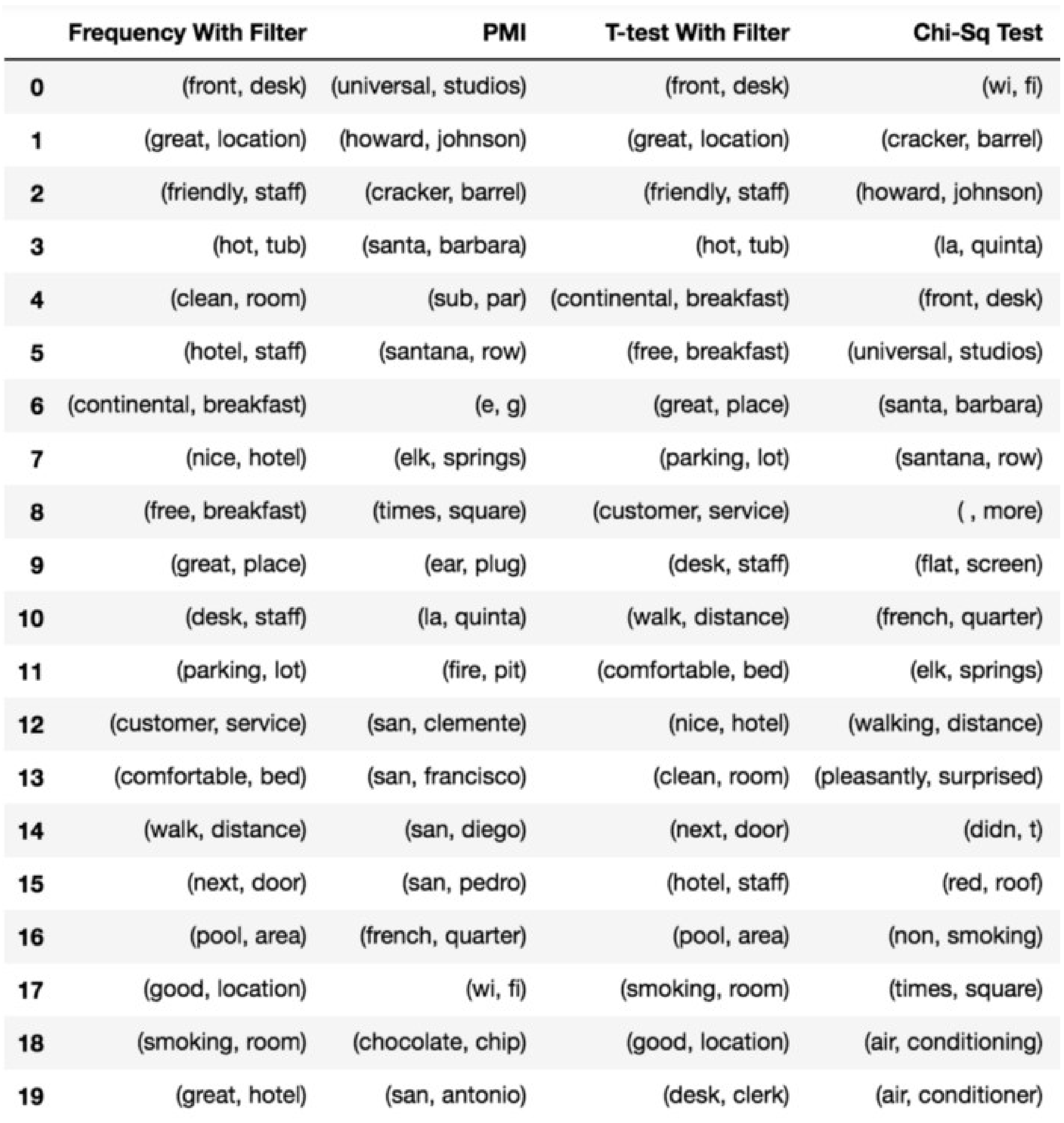
Results are as follows:



	bigram	chi-sq	trigram	chi-sq
0	(wi, fi)	1.651813e+06	(within, walking, distance)	6.107819e+08
1	(cracker, barrel)	1.322475e+06	(elk, springs, resort)	5.784021e+08
2	(howard, johnson)	1.206747e+06	(flat, screen, tv)	5.017370e+08
3	(la, quinta)	1.070882e+06	(holiday, inn, express)	4.431679e+08
4	(front, desk)	1.027741e+06	(zion, national, park)	3.036709e+08
5	(universal, studios)	9.041648e+05	(red, roof, inn)	1.244392e+08
6	(santa, barbara)	8.704756e+05	(hard, boil, egg)	1.046558e+08
7	(santana, row)	8.364605e+05	(free, wi, fi)	1.017867e+08
8	(, more)	7.497511e+05	(simpson, house, inn)	4.578081e+07
9	(flat, screen)	7.115527e+05	(within, walk, distance)	3.555933e+07

Comparing top 20 results of all methods:

Bigrams:



	Frequency With Filter	PMI	T-test With Filter	Chi-Sq Test
0	(front, desk)	(universal, studios)	(front, desk)	(wi, fi)
1	(great, location)	(howard, johnson)	(great, location)	(cracker, barrel)
2	(friendly, staff)	(cracker, barrel)	(friendly, staff)	(howard, johnson)
3	(hot, tub)	(santa, barbara)	(hot, tub)	(la, quinta)
4	(clean, room)	(sub, par)	(continental, breakfast)	(front, desk)
5	(hotel, staff)	(santana, row)	(free, breakfast)	(universal, studios)
6	(continental, breakfast)	(e, g)	(great, place)	(santa, barbara)
7	(nice, hotel)	(elk, springs)	(parking, lot)	(santana, row)
8	(free, breakfast)	(times, square)	(customer, service)	(, more)
9	(great, place)	(ear, plug)	(desk, staff)	(flat, screen)
10	(desk, staff)	(la, quinta)	(walk, distance)	(french, quarter)
11	(parking, lot)	(fire, pit)	(comfortable, bed)	(elk, springs)
12	(customer, service)	(san, clemente)	(nice, hotel)	(walking, distance)
13	(comfortable, bed)	(san, francisco)	(clean, room)	(pleasantly, surprised)
14	(walk, distance)	(san, diego)	(next, door)	(didn, t)
15	(next, door)	(san, pedro)	(hotel, staff)	(red, roof)
16	(pool, area)	(french, quarter)	(pool, area)	(non, smoking)
17	(good, location)	(wi, fi)	(smoking, room)	(times, square)
18	(smoking, room)	(chocolate, chip)	(good, location)	(air, conditioning)
19	(great, hotel)	(san, antonio)	(desk, clerk)	(air, conditioner)

Machine Learning Naturallanguageprocessing Towards Data Science Data Science Data

Trigrams: