# Feature Engineering

This section will cover the following types of features for the Yelp reviews:

1. Bag of Words
2. Bag of N-Grams
3. TF-IDF (term frequency over inverse document frequency)

```
import pandas as pd
import numpy as np
import re
import nltk
```

The corpus or the reviews were extracted from the Yelp review dataset using pandas

```
corpus_df = pd.read_csv('C:/Users/Peter Dell/Documents/ML1010-Yelp-Project/data/subset.csv'
corpus = corpus_df['text']
corpus.head()
```

```
0    Hallelujah! I FINALLY FOUND IT! The frozen yog...
1    I drop by BnC on a weekly basis to pick up my ...
2    My personally experience here wasn't the best,...
3    37 °C = 98.6°F\r\nKoreatown establisments disp...
4    My husband & I visited Toronto from the U.S. f...
Name: text, dtype: object
```

# Text pre-processing

As part of Text pre-processing we removed the special characters, whitespaces and numbers and, converted all the text to lower case.

```
wpt = nltk.WordPunctTokenizer()
stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
    # lower case and remove special characters\whitespaces
    doc = re.sub(r'[^a-zA-Z\s]', '', doc, re.I)
    # doc = re.sub(r'[^a-zA-Z0-9\s]', '', doc, re.I)
    doc = doc.lower()
    doc = doc.strip()
    # tokenize document
    tokens = wpt.tokenize(doc)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # re-create document from filtered tokens
    doc = ' '.join(filtered_tokens)
    doc = ''.join(i for i in doc if not i.isdigit())
    return doc
```

```
normalize_corpus = np.vectorize(normalize_document)
```

```
norm_corpus = normalize_corpus(corpus)
norm_corpus
```

# 1. Bag of Words Model

```
We created the Bag of Words model to determine the unique words in each document along with
```

```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(min_df=0., max_df=1.)
cv_matrix = cv.fit_transform(norm_corpus)
cv_matrix = cv_matrix.toarray()
cv_matrix
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

Thus you can see that our documents have been converted into numeric vectors such that each document is represented by one vector (row) in the above feature matrix. The following code will help represent this in a more easy to understand format.

```
# get all unique words in the corpus
vocab = cv.get_feature_names()
vocab
```

```
['______',
 '_______',
 '_______________',
 '____berto',
 '_accommodating',
 '_c',
 '_finally_',
 '_gyibeahdfylsszc_g',
 '_lozhqednolhvbg',
 '_reasonable',
 '_she',
 '_third_',
 '_us_',
 '_very',
 '_xhxtuykqnyphmylm',
 'aa',
 'aaa',
 'aaaaaalright',
 'aaaamazing',
 'aaammmazzing',
 'aaand',
 'aah',
 'aand',
 'aaron',
 'aarp',
 'ab',
 'aback',
 'abacus',
 'abandon',
 'abandoned',
 'abandoning',
 'abba',
 'abbaye',
 'abbey',
 'abbreviate',
 'abbreviated',
 'abbreviations',
 'abby',
 'abc',
 'abdomen',
 'abe',
 'aberration',
```

```
# show document feature vectors
pd.DataFrame(cv_matrix, columns=vocab)
```

| | ______ | _______ | _______________ | ____berto | _accommodating | _c | _finally_ | _g |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **5** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **7** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **8** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **9** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **10** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **11** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **12** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **13** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **14** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **15** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **16** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **17** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **18** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **19** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **20** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **21** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **22** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **23** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **24** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

# 2. Bag of N-Grams Model

We created the Bag of bi-grams and tri-grams to look at the 2-word and 3-word strings used

```
bv = CountVectorizer(ngram_range=(2,2))
```

```
bv_matrix = bv.fit_transform(norm_corpus)

bv_matrix = np.asarray(bv_matrix)
vocab = bv.get_feature_names()
# pd.DataFrame(bv_matrix, columns=vocab)
vocab
```

```
['______ ordered',
 '_______ oakland',
 '_______________ update',
 '____berto matter',
 '_accommodating evening',
 '_finally_ found',
 '_gyibeahdfylsszc_g adventures',
 '_lozhgednolhvbg http',
```

```
bv = CountVectorizer(ngram_range=(3,3))
bv_matrix = bv.fit_transform(norm_corpus)

bv_matrix = np.asarray(bv_matrix)
vocab = bv.get_feature_names()
vocab
```

```
['______ ordered chicken',
 '_______ oakland coliseum',
 '_______________ update first',
 '____berto matter basically',
 '_accommodating evening appointments',
 '_finally_ found place',
 '_gyibeahdfylsszc_g adventures phoenix',
 '_lozhqednolhvbg http www',
 '_reasonable amount time',
 '_she listens every',
 '_she pretty busy',
 '_third_ visit since',
 '_us_ going wonder',
 '_very friendly _accommodating',
 '_xhxtuykqnyphmylm mqg dessert',
 'aa accessories fab',
```

# 3. TF-IDF Model

```python
from sklearn.feature_extraction.text import TfidfVectorizer

tv = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
tv_matrix = tv.fit_transform(norm_corpus)
tv_matrix = tv_matrix.toarray()

vocab = tv.get_feature_names()
pd.DataFrame(np.round(tv_matrix, 2), columns=vocab)
```

| | ______ | _______ | _______________ | ____berto | _accommodating | _c | _finally_ | _ |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **1** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **2** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **3** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **4** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **5** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **6** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **7** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **8** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **9** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **10** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **11** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **12** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **13** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **14** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **15** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |

The TF-IDF based feature vectors for each of our text documents show scaled and normalized values as compared to the raw Bag of Words model values.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **18** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **19** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **20** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **21** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **22** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **23** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **24** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **25** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **26** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **27** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **28** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **29** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

| ... | ... | ... | ... | ... | ... | ... | ... |
|---|---|---|---|---|---|---|---|