# # Capstone1: Feature Engineering

The purpose of this document is to explain the steps taken to in the feature engineering stage of the process. A number of sources were used to support this work and have been referenced in the final section of the report.

## ## Steps

1. Set up jupyter notebook - `feature\_engineering.ipynb`

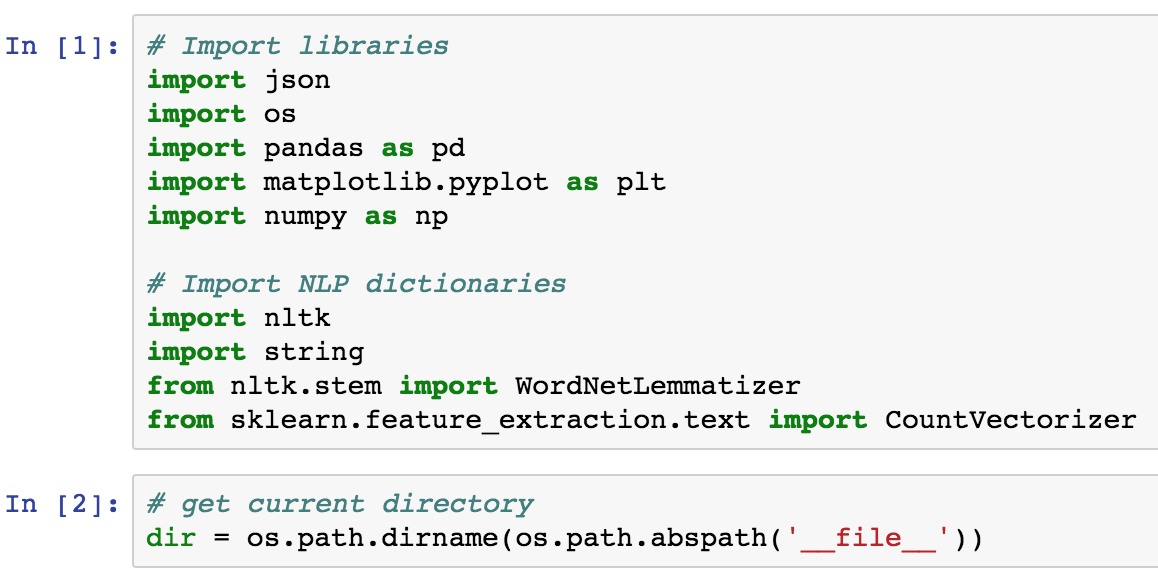
2. Load in pre-processed dataframe

3. Vectorise text

### ### 1. Set up jupyter notebook

The data wrangling stage can be found a Jupyter notebook called `data\_wrangling.ipynb`. It will contain the key steps laid out in this document. The inputs are described in step 2, and the outputs are described in step 13.

As part of the set up, the library dependencies are loaded.



In particular:

The common DataScience libraries

- json for loading in json files

- os for non-OS dependent file paths

- pandas for manipulation of DataFrames

- matplotlib.pyplot for plots and simple visualizations

- numpy to handle simple data arrays

As well as more specific NLP libraries

- nltk for stopwords, the lemmatization,

- string to get the library of punctuations

- sklearn.feature\_extraction.text for the string to word vectorizer

The working directory is also generated for future reference.

### ### 2. Load in data

The dataframe to be loaded is the pre-processed data that was obtained from the data wrangling stage. It had been checked in the data wrangling stage and can be loaded in with:

```

file\_path = os.path.join(dir, '02\_processed\_data','restaurant\_reviews.csv')

joint\_df2 = pd.read\_csv(file\_path, index\_col = False, parse\_dates= ['date'])

```

### ### 3. Vectorize text

Once the review text has been preprocessed, it is ready to be converted into a vector. First, the vectorizer needs to be defined. In our case we have set the parameters as follows:

```

n\_features = 3000

vectorizer = CountVectorizer(analyzer = ‘word’,

max\_features = n\_features,

max\_df=0.95,

min\_df=2).fit(" ".join(line) for line in joint\_df['processed\_review'])

```

max\_features: The total number of words used in the bag of words is restricted to 3000. This number is arbitrary large but can be tweaked in the future if necessary.

max\_df: This restriction is the upper threshold of the document frequency that can be expected. This has been arbitrarily set at 95% but can be lowered if commonly occurring words are not helpful.

min\_df: This is the lower threshold for the document frequency that must be met for the word to feature. If it is a float, the parameter represents a proportion of documents, if it is an integer, the parameter represents absolute counts.

For more information on the parameters of CountVectorizer, refer to reference [1] [2]

It may be observed that these parameters create a vectorizer with 3,000 features (the limit). The vectorized reviews create an array with the following properties:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Config 1 | Config 2 | Config 3 |
| n\_features | 3000 | 3000 |  |
| max\_df | 0.95 | 0.95 |  |
| min\_df | 2 | 2 |  |
| sparse matrix shape | 2876509, 3000 | 2876509, 3000 |  |
| number of non-zeros | 119826809 |  |  |
| sparsity | 1.39% |  |  |

Additional combinations can be studied to find a more optimal configuration.

The vectorizer is then applied to the review text column and stored to a new numpy array:

```

# Vectorize text and convert the result to an Numpy array

vectorized\_text\_array = vectorizer.transform(joint\_df['text']).toarray()

```

The most common words do no indicate any abnormities or outliers.

### References:

[1] (http://scikit-learn.org/stable/modules/generated/sklearn.feature\_extraction.text.CountVectorizer.html)

[2] (https://stackoverflow.com/questions/22920801/can-i-use-countvectorizer-in-scikit-learn-to-count-frequency-of-documents-that-w)