**Material sourced from:** [Python for Data Science and Machine Learning Bootcamp](https://www.udemy.com/python-for-data-science-and-machine-learning-bootcamp/learn/v4/t/lecture/5800258?start=0)

**Natural language processing** (or NLP) serves numerous use cases when dealing with text or unstructured text data.

Imagine if you worked for [*Google News*](https://news.google.com/news/?ned=us&hl=en) and wanted to group news articles by topic. Or imagine if you worked at a legal firm and had to find documents relevant to a particular case. It would be very tiring and time-consuming to manually sift through thousands of articles, right? This is where NLP could come in handy.

Today, let’s build a simple text classifier using Python’s [Pandas](http://pandas.pydata.org/), [NLTK](http://www.nltk.org/) and [Scikit-learn](http://scikit-learn.org/stable/) libraries. Our goal is to build a sentiment analysis model that predicts whether a user liked a local business or not, based on their review on Yelp.

**The dataset**

Our data contains 10,000 reviews, with the following information for each one:

1. **business\_id** (ID of the business being reviewed)
2. **date** (Day the review was posted)
3. **review\_id** (ID for the posted review)
4. **stars** (1–5 rating for the business)
5. **text** (Review text)
6. **type** (Type of text)
7. **user\_id** (User’s id)
8. {**cool** / **useful** / **funny**} (Comments on the review, given by other users)

Let’s see how we can go about analysing this dataset using Pandas, NLTK, and Scikit-learn.

**Importing the dataset**

Firstly, let’s import the necessary Python libraries. NLTK is pretty much the standard library in Python library for text processing, which has many useful features. Today, we will just use NLTK for stopword removal.

import **pandas** as **pd**  
import **numpy** as **np**  
import **matplotlib**.**pyplot** as **plt**  
import **seaborn** as **sns**import **nltk**  
from **nltk**.**corpus** import **stopwords**

Next, we can import the [Yelp Reviews CSV file](https://www.dropbox.com/s/wc6rzl1a2os721d/yelp.csv?dl=0) and store it in a Pandas dataframe called yelp.

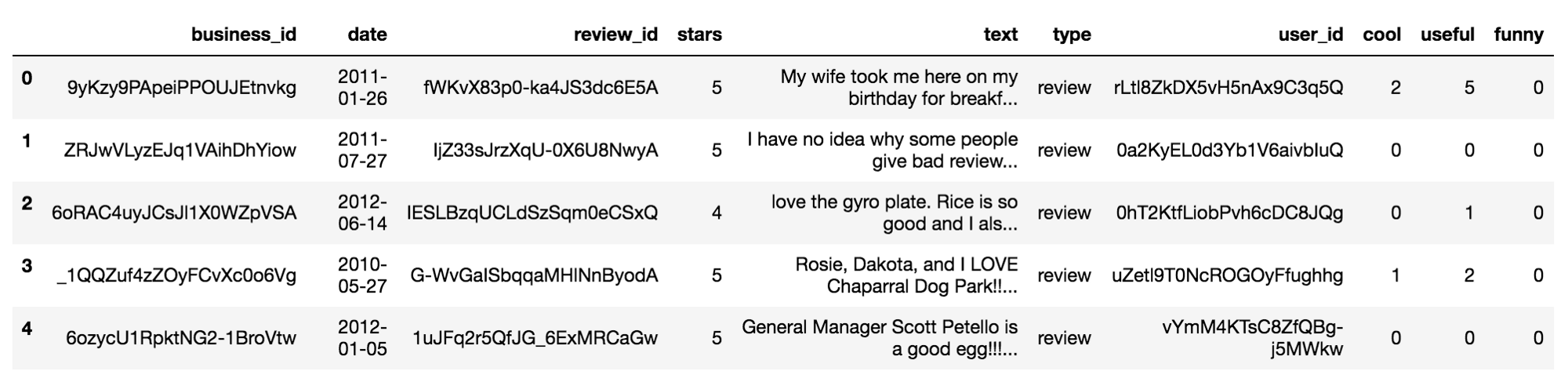
yelp = pd.**read\_csv**('yelp.csv')

Let’s get some basic information about the data. The .shape method tells us the number of rows and columns in the dataframe.

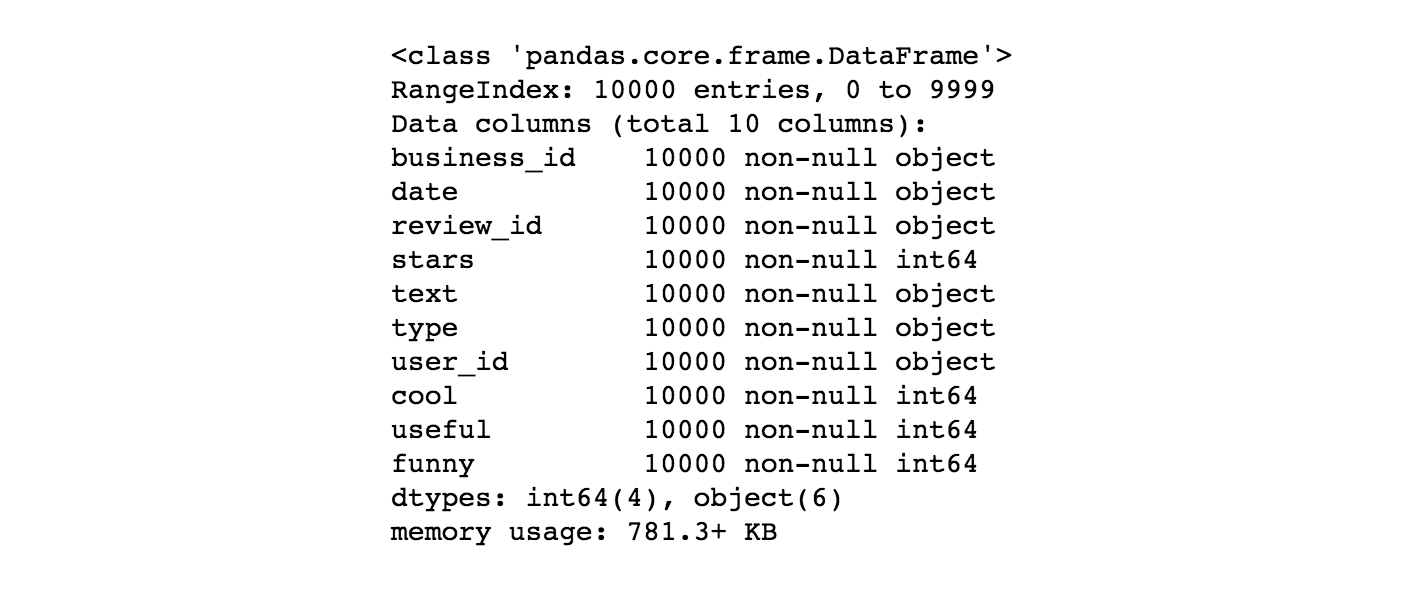
yelp.**shapeOutput:** (10000, 10)

We can learn more using .head(), .info(), and .describe().

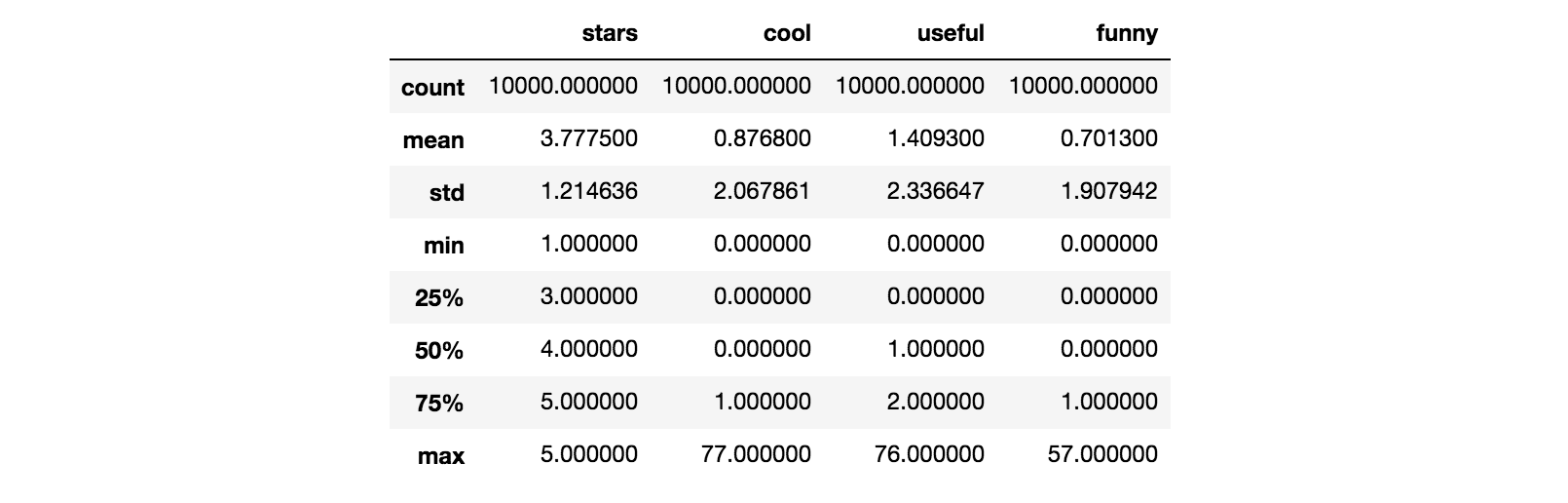
yelp.**head**()

The first 5 rows of our dataset.

yelp.**info**()

Basic information about each column in our dataset.

yelp.**describe**()



More information about the numeric columns in our dataset.

To get an insight on the length of each review, we can create a new column in yelp called text length. This column will store the number of characters in each review.

yelp[**'text length'**] = yelp['text'].**apply**(len)  
yelp.**head**()

We can now see the text length column in our dataframe using .head():

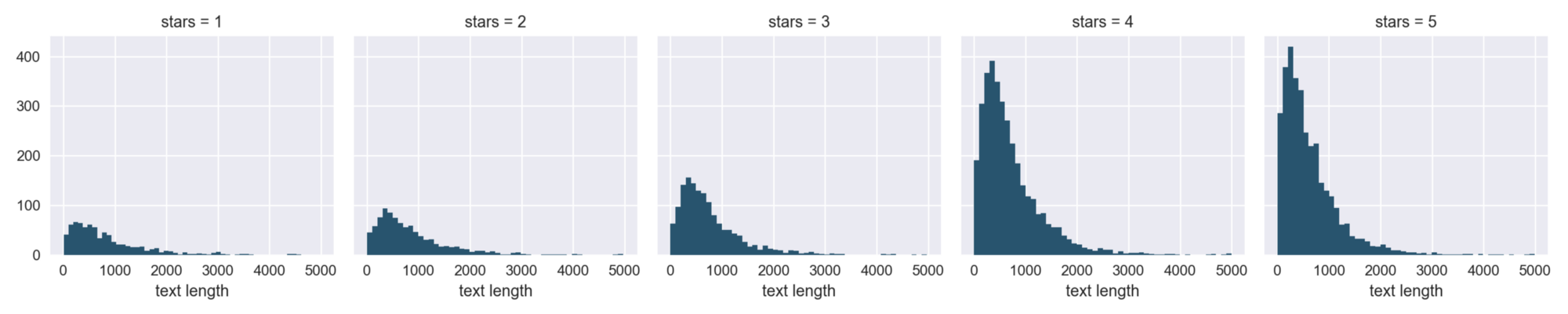
The first 5 rows of the yelp dataframe with text length feature added at the end.

**Exploring the dataset**

Let’s visualise the data a little more by plotting some graphs with the [Seaborn](http://seaborn.pydata.org/) library.

Seaborn’s [FacetGrid](http://seaborn.pydata.org/generated/seaborn.FacetGrid.html) allows us to create a grid of histograms placed side by side. We can use FacetGrid to see if there’s any relationship between our newly created text length feature and the stars rating.

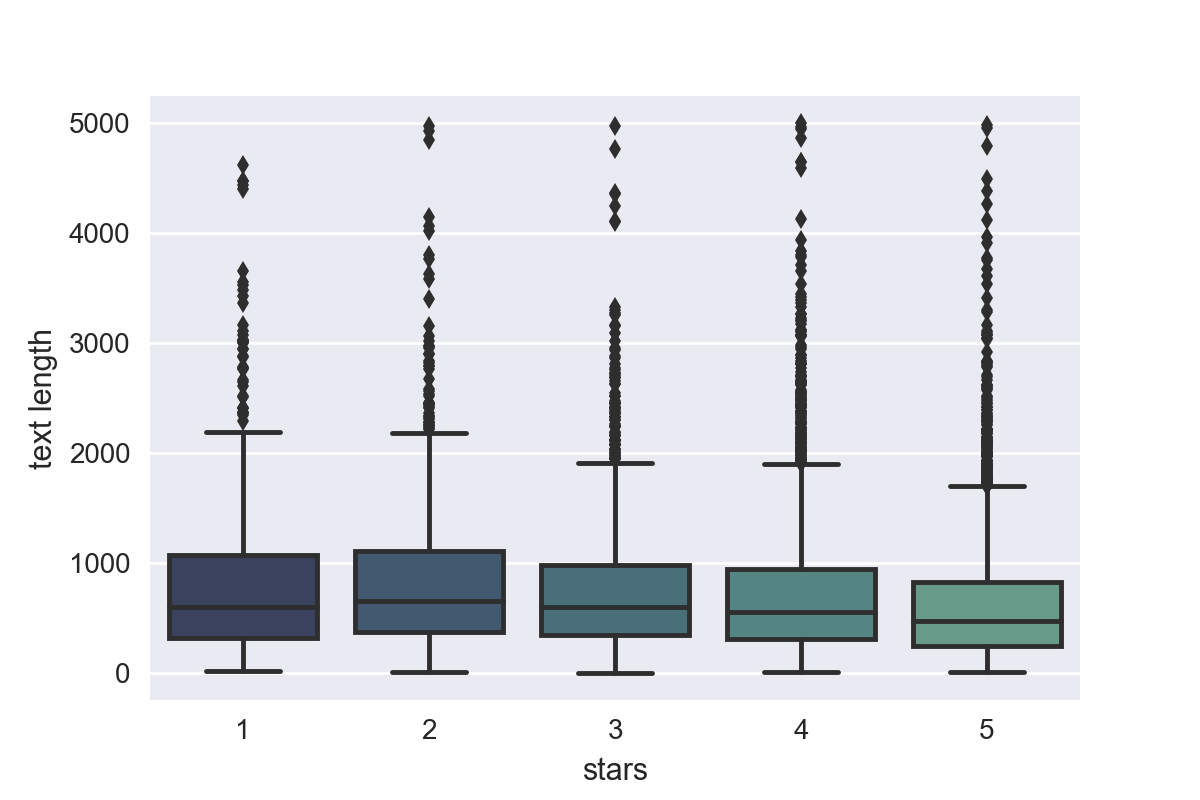
g = sns.**FacetGrid**(data=yelp, col='stars')  
g.**map**(plt.hist, 'text length', bins=50)

Histograms of text length distributions for each star rating. Notice that there is a high number of 4-star and 5-star reviews.

Seems like overall, the distribution of text length is **similar** across all five ratings. However, the number of text reviews seems to be skewed a lot higher towards the 4-star and 5-star ratings. This may cause some issues later on in the process.

Next, let’s create a [box plot](http://seaborn.pydata.org/generated/seaborn.boxplot.html) of the text length for each star rating.

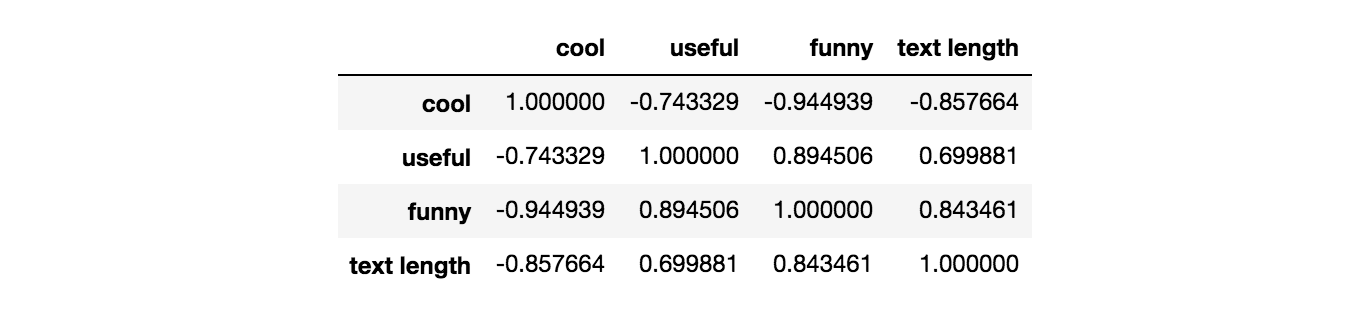
sns.**boxplot**(x='stars', y='text length', data=yelp)

Box plot of text length against star ratings.

From the plot, looks like the 1-star and 2-star ratings have much longer text, but there are many outliers (which can be seen as points above the boxes). Because of this, maybe text length won’t be such a useful feature to consider after all.

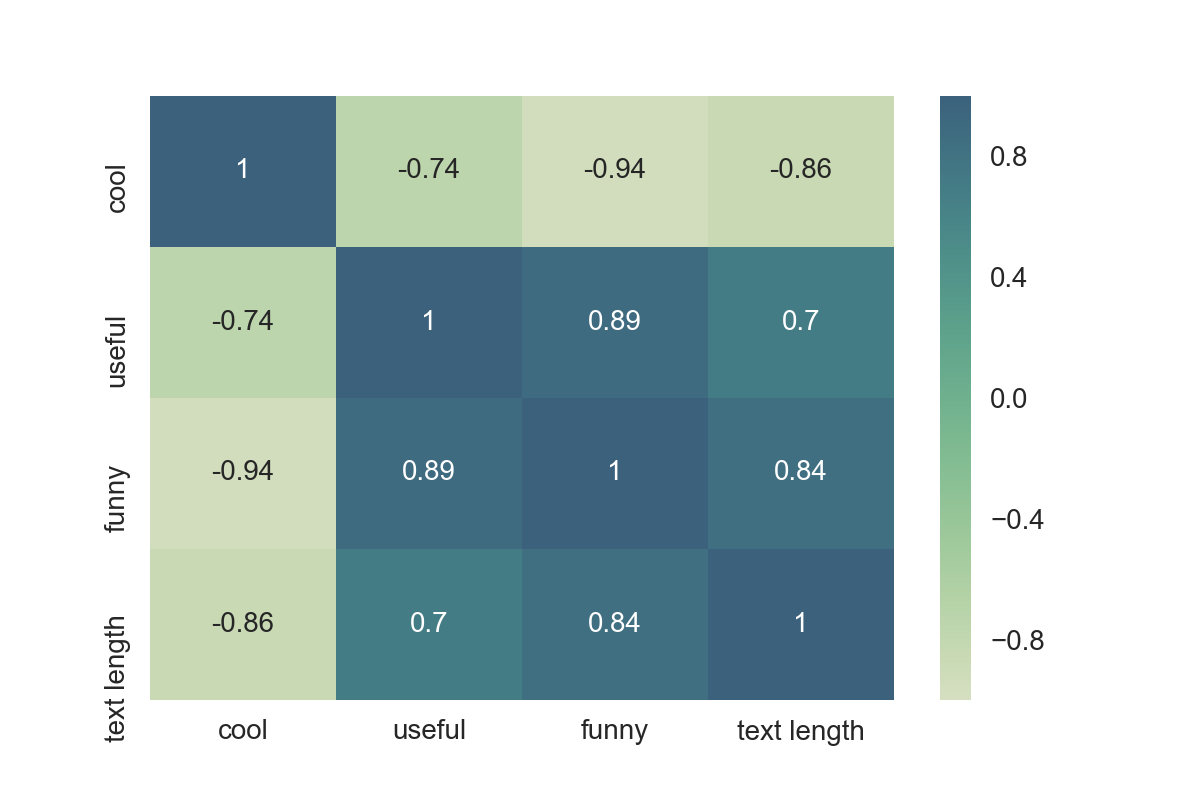
Let’s group the data by the star rating and see if we can find a correlation between features such as cool, useful, and funny. We can use the .corr()method from Pandas to find any correlations in the dataframe.

stars = yelp.**groupby**('stars').mean()  
stars.**corr**()

Correlations between cool, useful, funny, and text length.

To visualise these correlations, we can use Seaborn’s [heatmap](http://seaborn.pydata.org/generated/seaborn.heatmap.html):

sns.**heatmap**(data=stars.**corr**(), annot=True)

Heat map of correlations between cool, useful, funny, and text length.

Looking at the map, funny is strongly correlated with useful, and usefulseems strongly correlated with text length. We can also see a negative correlation between cool and the other three features.

**Independent and dependent variables**

Our task is to predict if a review is either bad or good, so let’s just grab reviews that are either 1 or 5 stars from the yelp dataframe. We can store the resulting reviews in a new dataframe called yelp\_class.

**yelp\_class** = yelp[(yelp['stars'] == 1) | (yelp['stars'] == 5)]  
yelp\_class.**shapeOutput:** (4086, 11)

We can see from .shape that yelp\_class only has 4086 reviews, compared to the 10,000 reviews in the original dataset. This is because we aren’t taking into account the reviews rated 2, 3, and 4 stars.

Next, let’s create the X and y for our classification task. X will be the text column of yelp\_class, and y will be the stars column.

X = yelp\_class['text']  
y = yelp\_class['stars']

**Text pre-processing**

The main issue with our data is that it is all in plain-text format.

X[0]**Output:** 'My wife took me here on my birthday for breakfast and it was excellent. The weather was perfect which made sitting outside overlooking their grounds an absolute pleasure. Our waitress was excellent and our food arrived quickly on the semi-busy Saturday morning. It looked like the place fills up pretty quickly so the earlier you get here the better.\n\nDo yourself a favor and get their Bloody Mary. It was phenomenal and simply the best I\'ve ever had. I\'m pretty sure they only use ingredients from their garden and blend them fresh when you order it. It was amazing.\n\nWhile EVERYTHING on the menu looks excellent, I had the white truffle scrambled eggs vegetable skillet and it was tasty and delicious. It came with 2 pieces of their griddled bread with was amazing and it absolutely made the meal complete. It was the best "toast" I\'ve ever had.\n\nAnyway, I can\'t wait to go back!'

The classification algorithm will need some sort of **feature vector** in order to perform the classification task. The simplest way to convert a corpus to a vector format is the **bag-of-words** approach, where each unique word in a text will be represented by one number.

First, let’s write a function that will split a message into its individual words, and return a list. We will also remove the very common words (such as “the”, “a”, “an”, etc.), also known as stopwords. To do this, we can take advantage of the NLTK library. The function below removes punctuation, stopwords, and returns a list of the remaining words, or tokens.

import **stringdef** text\_process(**text**): *'''  
 Takes in a string of text, then performs the following:  
 1. Remove all punctuation  
 2. Remove all stopwords  
 3. Return the cleaned text as a list of words  
 '''* **nopunc** = [char **for** char **in** text **if** char **not in** string.punctuation] nopunc = ''.**join**(nopunc)  
   
 **return** [word **for** word **in** nopunc.split() **if** word.lower() **not in** stopwords.words('english')]

To check if the function works, let’s pass in some random text and see if it gets processed correctly.

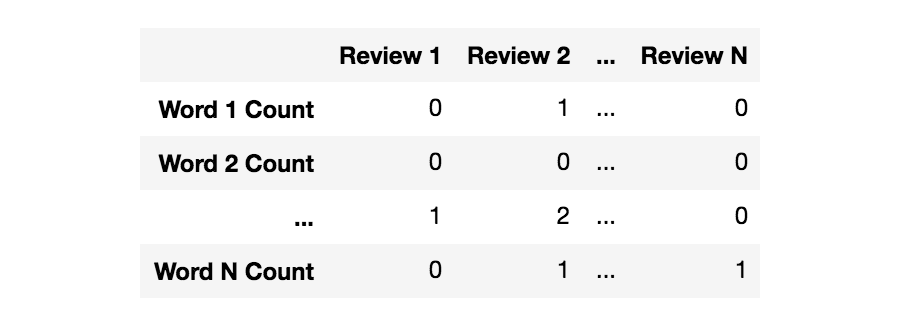
**sample\_text** = "Hey there! This is a sample review, which happens to contain punctuations."print(**text\_process**(sample\_text))**Output:** ['Hey', 'sample', 'review', 'happens', 'contain', 'punctuations']

Seems like it works! There are no punctuations or stopwords, and the remaining words are returned to us as a list of tokens.

**Vectorization**

At the moment, we have our reviews as lists of tokens (also known as [lemmas](https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html)). To enable Scikit-learn algorithms to work on our text, we need to convert each review into a vector.

We can use Scikit-learn’s [CountVectorizer](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html) to convert the text collection into a matrix of token counts. You can imagine this resulting matrix as a 2-D matrix, where each row is a unique word, and each column is a review.

A matrix of token counts, indicating how many instances of a particular word appear in a review.

Since there are many reviews, we can expect a lot of zero counts for the presence of a word in the collection. Because of this, Scikit-learn will output a [sparse matrix](https://en.wikipedia.org/wiki/Sparse_matrix).

Let’s import CountVectorizer and fit an instance to our review text (stored in X), passing in our text\_process function as the analyser.

bow\_transformer = **CountVectorizer**(analyzer=**text\_process**).**fit**(X)

Now, we can look at the size of the vocabulary stored in the vectoriser (based on X) like this:

**len**(bow\_transformer.**vocabulary\_**)**Output:** 26435

To illustrate how the vectoriser works, let’s try a random review and get its **bag-of-word counts** as a vector. Here’s the twenty-fifth review as plain-text:

**review\_25** = X[24]  
review\_25**Output:** 'I love this place! I have been coming here for ages.  
My favorites: Elsa's Chicken sandwich, any of their burgers, dragon chicken wings, china's little chicken sandwich, and the hot pepper chicken sandwich. The atmosphere is always fun and the art they display is very abstract but totally cool!'

Now let’s see our review represented as a vector:

**bow\_25** = bow\_transformer.**transform**([review\_25])  
bow\_25**Output:** (0, 2099) 1  
(0, 3006) 1  
(0, 8909) 1  
(0, 9151) 1  
(0, 9295) 1  
(0, 9616) 1  
(0, 9727) 1  
(0, 10847) 1  
**(0, 11443) 3**  
(0, 11492) 1  
(0, 11878) 1  
(0, 12221) 1  
(0, 13323) 1  
(0, 13520) 1  
(0, 14481) 1  
(0, 15165) 1  
(0, 16379) 1  
(0, 17812) 1  
(0, 17951) 1  
(0, 20044) 1  
(0, 20298) 1  
**(0, 22077) 3**  
(0, 24797) 1  
(0, 26102) 1

This means that there are 24 unique words in the review (after removing stopwords). Two of them appear thrice, and the rest appear only once. Let’s go ahead and check which ones appear thrice:

print(bow\_transformer.**get\_feature\_names**()[**11443**])  
print(bow\_transformer.**get\_feature\_names**()[**22077**])**Output:**   
chicken  
sandwich

Now that we’ve seen how the vectorisation process works, we can transform our X dataframe into a sparse matrix. To do this, let’s use the .transform()method on our bag-of-words transformed object.

X = bow\_transformer.**transform**(X)

We can check out the shape of our new X.

print('Shape of Sparse Matrix: ', X.**shape**)  
print('Amount of Non-Zero occurrences: ', X.**nnz**)# Percentage of non-zero values **density** = (100.0 \* X.nnz / (X.shape[0] \* X.shape[1]))  
print(‘Density: {}’.format((density)))**Output:**  
Shape of Sparse Matrix: (4086, 26435)  
Amount of Non-Zero occurrences: 222391  
Density: 0.2058920276658241

**Training data and test data**

As we have finished processing the review text in X, It’s time to split our Xand y into a training and a test set using train\_test\_split from Scikit-learn. We will use **30%** of the dataset for testing.

from **sklearn**.**model\_selection** import **train\_test\_split**X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X, y, test\_size=0.3, random\_state=101)

**Training our model**

[Multinomial Naive Bayes](http://www.cs.waikato.ac.nz/~eibe/pubs/kibriya_et_al_cr.pdf) is a specialised version of Naive Bayes designed more for text documents. Let’s build a Multinomial Naive Bayes model and fit it to our training set (X\_train and y\_train).

from **sklearn**.**naive\_bayes** import **MultinomialNB**nb = **MultinomialNB**()  
nb.**fit**(X\_train, y\_train)

**Testing and evaluating our model**

Our model has now been trained! It’s time to see how well it predicts the ratings of previously unseen reviews (reviews from the test set). First, let’s store the predictions as a separate dataframe called preds.

preds = nb.**predict**(X\_test)

Next, let’s evaluate our predictions against the actual ratings (stored in y\_test) using confusion\_matrix and classification\_report from Scikit-learn.

from **sklearn**.**metrics** import **confusion\_matrix**, **classification\_report**print(**confusion\_matrix**(y\_test, preds))  
print('\n')  
print(**classification\_report**(y\_test, preds))**Output:**[[157 71]  
[ 24 974]]  
  
  
 precision recall f1-score support  
  
 1 0.87 0.69 0.77 228  
 5 0.93 0.98 0.95 998  
  
avg / total **0.92** **0.92** **0.92** 1226

Looks like our model has achieved **92%** accuracy! This means that our model can predict whether a user liked a local business or not, based on what they typed!

**Data Bias**

Although our model achieved quite a high accuracy, there are some issues with bias caused by the dataset.

Let’s take some singular reviews, and see what rating our model predicts for each one.

**Predicting a singular positive review**

positive\_review = **yelp\_class**['text'][59]  
positive\_review**Output:** 'This restaurant is incredible, and has the best pasta carbonara and the best tiramisu I've had in my life. All the food is wonderful, though. The calamari is not fried. The bread served with dinner comes right out of the oven, and the tomatoes are the freshest I've tasted outside of my mom's own garden. This is great attention to detail.\n\nI can no longer eat at any other Italian restaurant without feeling slighted. This is the first place I want take out-of-town visitors I'm looking to impress.\n\nThe owner, Jon, is helpful, friendly, and really cares about providing a positive dining experience. He's spot on with his wine recommendations, and he organizes wine tasting events which you can find out about by joining the mailing list or Facebook page.'

Seems like someone had the time of their life at this place, right? We can expect our model to predict a rating of 5 for this review.

positive\_review\_transformed = bow\_transformer.**transform**([positive\_review])nb.**predict**(positive\_review\_transformed)[0]**Output:** 5

Our model thinks this review is positive, just as we expected.

**Predicting a singular negative review**

negative\_review = **yelp\_class**['text'][281]  
negative\_review**Output:** 'Still quite poor both in service and food. maybe I made a mistake and ordered Sichuan Gong Bao ji ding for what seemed like people from canton district. Unfortunately to get the good service U have to speak Mandarin/Cantonese. I do speak a smattering but try not to use it as I never feel confident about the intonation. \n\nThe dish came out with zichini and bell peppers (what!??) Where is the peanuts the dried fried red peppers and the large pieces of scallion. On pointing this out all I got was " Oh you like peanuts.. ok I will put some on" and she then proceeded to get some peanuts and sprinkle it on the chicken.\n\nWell at that point I was happy that atleast the chicken pieces were present else she would probably end up sprinkling raw chicken pieces on it like the raw peanuts she dumped on top of the food. \n\nWell then I spoke a few chinese words and the scowl turned into a smile and she then became a bit more friendlier. \n\nUnfortunately I do not condone this type of behavior. It is all in poor taste...'

This is a slightly more negative review. So, we can expect our model to rate this a 1-star.

negative\_review\_transformed = bow\_transformer.**transform**([negative\_review])nb.**predict**(negative\_review\_transformed)[0]**Output:** 1

Our model is right again!

**Where the model goes wrong…**

another\_negative\_review = **yelp\_class**['text'][140]  
another\_negative\_review**Output:** 'Other than the really great happy hour prices, its hit or miss with this place. More often a miss. :(\n\nThe food is less than average, the drinks NOT strong ( at least they are inexpensive) , but the service is truly hit or miss.\n\nI'll pass.'

Here’s another negative review. Let’s see if the model predicts this one correctly.

another\_negative\_review\_transformed = bow\_transformer.**transform**([another\_negative\_review])nb.**predict**(another\_negative\_review\_transformed)[0]**Output:** 5

Our model thinks this review is positive, and that’s incorrect.

**Why the incorrect prediction?**

One explanation as to why this may be the case is that our initial dataset had a much higher number of 5-star reviews than 1-star reviews. This means that the model is more **biased** towards positive reviews compared to negative ones.

In conclusion, although our model was a little biased towards positive reviews, it was fairly accurate with its predictions, achieving an accuracy of 92% on the test set.

**References**

Links to the primary sources I used are linked below:

1. **Dataset obtained from:** [Kaggle: Yelp Business Rating Prediction](https://www.kaggle.com/c/yelp-recsys-2013)