Nicolas Brooks

Professor Anandarajan

Independent Study

Summer 2017

Natural Language Processing with Retail Data

Project Github: <https://github.com/nicapotato/Jupyter/tree/master/NLP/Retail%20Project>

Code, data and written report also available through this link.

Programming Language: Python 3.5 in the Jupyter Notebook Environment

Textbook Resources Used:

Swamynathan, Manohar. *Mastering Machine Learning with Python in Six Steps: a Practical Implementation Guide to Predictive Data Analytics Using Python*. Apress, 2017.

Bird, Steven. *Natural Language Processing with Python*. O'Reilly Media, 2016.

Code Navigation:

In the code, text after hastags (#) are supportive explanations, not executed as code.

Indented line signifies code is part of larger function or loop. Not standalone. Furthermore, functions are used in order to facilitate the simplicity and exploratory process of the code.

Code: Packages Used

# General

import numpy as np

import pandas as pd

import nltk

import random

import os

from os import path

from PIL import Image

# Visualization

import matplotlib as mpl

import matplotlib.pyplot as plt

%matplotlib inline

from subprocess import check\_output

from wordcloud import WordCloud, STOPWORDS

# Pre-Processing

import string

from nltk.tokenize import RegexpTokenizer

from nltk.corpus import stopwords

import re

from nltk.stem import PorterStemmer

# Modeling  
from nltk.sentiment.vader import SentimentIntensityAnalyzer

from nltk.sentiment.util import \*

from nltk.util import ngrams

from collections import Counter

from gensim.models import word2vec

Code Explanation:

These packages are seperated in four categories: *General, Visualization, Pre-Processing, and Modeling.* The *General* category includes the basic data manipulation tools for scientific computation (numpy), dataframes (pandas), Natural Language Processing (NLTK), path directory manipulation (os), and image saving (PIL). The *Visualiation* section enables the creation of simple graphics (matplotlib, seaborn), aswell as wordclouds (wordcloud). The *Pre-Processing* section extracts more specialized modules from the NLTK package such as tokenizers and stemmers to enable the preperation of text data for mathematical analysis. The *Modeling* section includes NLTK’s sentiment analysis module, which can determine the mood of text, NLTK’s N-grams, and gensim’s word2vec.

Introduction:

This independent study is concerned with using the Python programming language and Natural Language Processing technology to explore trends in the customer reviews from a women’s clothing retail store, and extract actionable plans to improve its online e-commerce. The data is a collection of 23,486 rows with 17 featured variables. Each row includes a written comment aswell as additional customer information. This analysis will focus on using Natural Language techniques to find broad trends in the written thoughts of the customers. The total number of unique words in the dataset is 9810. In this analysis, the data will be introduced using exploratory data analysis, and will be further analyzed by employing frequency distribution, Word Clouds, Sentiment Analysis, Naive Bayes, and finally Word2Vec to extract actionable findings.

Code: Read Data

# Path

os.chdir(r"C:\Users\Nicol\Google Drive\Learning\Jupyter\NLP\Retail Project")

os.listdir()

# Read Data

data = pd.ExcelFile("retail dataset.xlsx")

df = data.parse("Sheet1", header=1)

df = df.drop(["Unnamed: 18", "Unnamed: 19", "Unnamed: 20", "Unnamed: 21","Unnamed: 22","Unnamed: 23","Unnamed: 24","Unnamed: 25","Unnamed: 26"], axis = 1)

df = df.drop(["REVIEW\_ID", "RATING\_RANGE"], axis = 1)

# Transform Depedent Variable to Binary

df["LABEL"] = 0

df.loc[df.RATING >= 3,"LABEL"] = 1

Code Explanation:

The *Path* section changes the working directory of the python program to where the data is located. The *Read Data* section uses a Pandas function to read and parse the first sheet of the excel formatted retail dataset. Then, blank columns are removed, aswell as the unused variables “REVIEW\_ID” and “RATING\_RANGE”. Finally, the *Transform Depedent Variable to Binary* section creates a new variable.

Code Interpretation: Variables and Processing

The first step is to address the the variables and dive into the pre-processing steps necessary to turn raw text into valuable output. This dataset’s notable variables include: review title and review body of clothing product, rating assigned to the product, age of customer, whether the product was recommended, and finally department and division.

In order to facilitate the use of sentiment analysis, a new boolean variable is created to categories good and bad reviews. All reviews with a rating of 3 and over, were deemed good, and reviews under 3 deemed bad. This step is especially important for the use of Naive Bayes’ supervised learning algorithm, since it requires a clear binary label to train upon.

Code: Pre-Processing

ps = PorterStemmer()

tokenizer = RegexpTokenizer(r'\w+')

stop\_words = set(stopwords.words('english'))

def **preprocessing**(data):

txt = data.str.lower().str.cat(sep=' ') #1

words = tokenizer.tokenize(txt) #2

words = [w for w in words if not w in stop\_words] #3

words = [ps.stem(w) for w in words] #4

return words

Code Explanation:

This chunk of code creates a function that takes each review and combines them into one seamless text. It then applies lowercase, tokenizer, removes stopwords and punctuation, and finally uses the PorterStemmer.

Code Interpretation:

In order to process the dataset’s centerpiece, the review body, I utilized the NLTK package to lowercase, tokenize, and remove stopwords and punctuation. Tokenizing treats each word as its own value, while the other steps gets rid of the noise and irrelevant symbols in the data, standardizing the reviews for analysis. Upon reviewing the performance of text analysis, I decided to implement the Porter Stemmer on the tokens in order to combine words with tense and plurality deviance. I contemplated exploring the use of sequential models, such as Long Short-term memory, which would benefit from stopwords, but unfortunately I could only find predictive applications of it, no insight extracting aspects.

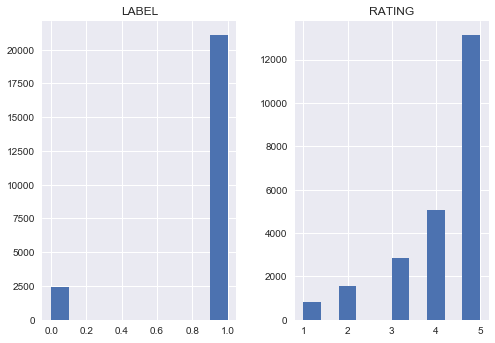
The last piece of data transformation conducted was to bin the continuous variable age into a categorical variable: age category.

Exploratory Data Analysis:

Code:

df[["LABEL","RATING"]].hist()

Code Output:



Code Explanation:

Using Matplotlib, a simple variable frequency barplot is created.

Code Interpretation:

The vast majority of reviews were highly positive, with a score of five out of five. This suggests that this retail store is performing fairly well, but comparison to competitors would determine whether it is satisfactory. Competitor reviews may be scraped and analyzed. It is important to note that these reviews are subjective, and some negative reviews may a outcome of a bad day, instead of constructive feedback. In the plot below, the Label plot is the binary classification of 1 = good, and 0= bad.

Sentiment Analysis

Code:

# Pre-Processing

SIA = SentimentIntensityAnalyzer()

data = df[["TITLE","REVIEW\_TEXT","CLASS\_NAME","AGE","RATING", "LABEL"]]

data.REVIEW\_TEXT= data.REVIEW\_TEXT.astype(str)

# Applying Model, Variable Creation

data['polarity\_score']=data.REVIEW\_TEXT.apply(lambda x:SIA.polarity\_scores(x)['compound'])

data['neutral\_score']=data.REVIEW\_TEXT.apply(lambda x:SIA.polarity\_scores(x)['neu'])

data['negative\_score']=data.REVIEW\_TEXT.apply(lambda x:SIA.polarity\_scores(x)['neg'])

data['positive\_score']=data.REVIEW\_TEXT.apply(lambda x:SIA.polarity\_scores(x)['pos'])

data['sentiment']=''

data.loc[data.polarity\_score>0,'sentiment']='POSITIVE'

data.loc[data.polarity\_score==0,'sentiment']='NEUTRAL'

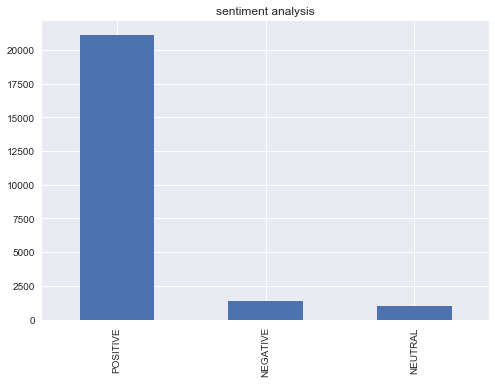
data.loc[data.polarity\_score<0,'sentiment']='NEGATIVE'

# Visualization

data.sentiment.value\_counts().plot(kind='bar',title="sentiment analysis")

plt.show()

Code Output:



Code Explanation:

*Pre-processing* chunk loads the NLTK Sentiment Intensity Analyzer module, selects desired variables, and finally applies lowercasing to the column of reviews in the dataframe. The second paragraph of code *Applying Model and Variable Creation* classifies each review in the dataset on three dimensions: Positive, Neutral, and Negative. These results are stored in three respective columns. The overall sentiment is then determined and stored in the Sentiment column. The last chunk, Visualization, plots the frequency of sentiments in a bar plot using matplotlib.

Code Interpretation:

Because the data sentiment is already outlined by the rating category, the use of sentiment analysis is redundant. Nevertheless, since this is my first time doing Natural Language Processing and programming with Python, I am eager enough to play around with the technology, and see how well it compares to customer labeling. Just like the rating, sentiment analysis classified just over 20,000 positive reviews, suggesting that sentiment analysis is a valid technology to on product reviews without labels, such as facebook or twitter reviews. The creation of a lexicon may also prove useful in analyzing competitor reviews.

Age Variable:

Now that I know the distribution of reviews, I am curious to find out the role played by customer age:

Code:

data["AGECAT"] = pd.cut(np.array(data.AGE), 5, retbins=False)

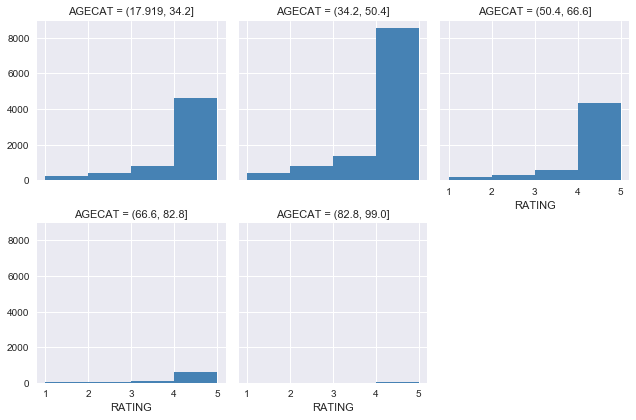
sns.set(style="darkgrid")

g = sns.FacetGrid(data, col="AGECAT",col\_wrap=3, margin\_titles=True)

bins = np.linspace(1, 5, 5)

g.map(plt.hist, "RATING", color="steelblue", bins=bins, lw=0)

Code Output:



Code Explanation:

This code is less obvious than the previous visualization since it attempts a more complex visualization using Seaborn instead of Matplotlib. Firstly, the “AGECAT” variable is created by cutting the continuous variable “AGE” into a categorical variable with 5 bins. Then, the FacetGrid module is used to create a panel of graphs, in which a graph may be created for each factor in a category, which in this case is the categorical age variable “AGECAT”. The bins for the x-axis are then specified, aswell as the variable used in the x-axis: Raiting.

Code Interpretation:

My a priori expectation was that the biggest group of reviewing customers would be young, tech savvy women between the age of 18 and 34. However, this plot would say otherwise, since it appears that not only is the 34 to 50 year old age most engage in reviewing products, they also appear to be the most positive reviewers, since they proportionately give higher more reviews of 5. Before making insight about these point, it would be wise to gather further data on the age distribution of shoppers. Nevertheless, this trend suggest that the core market segment for this clothing brand is women between 34 and 50.

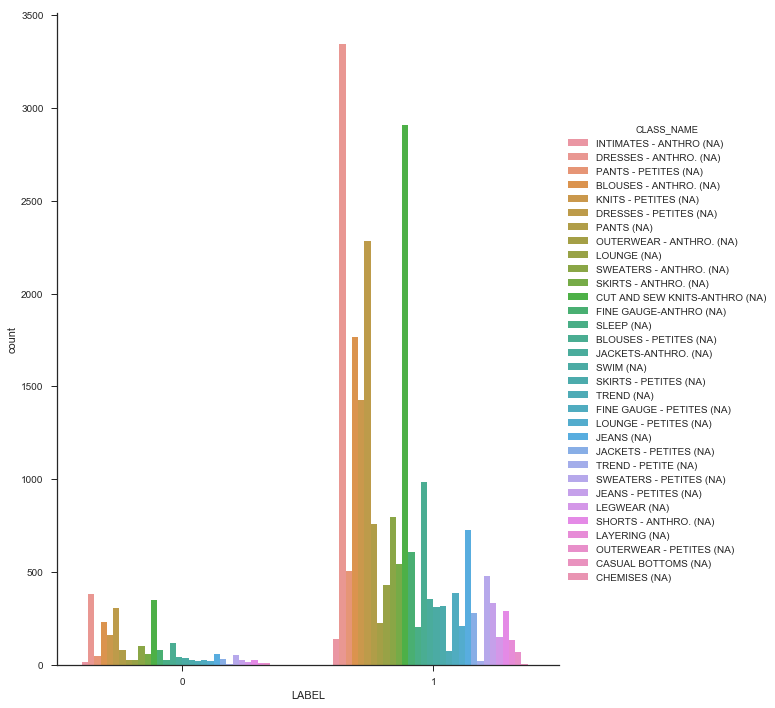
Clothing Type Variable:

Code:

sns.set(style="ticks")

g = sns.factorplot("LABEL", hue="CLASS\_NAME", data=data, kind="count", size=10, aspect=.8)

Code Output:



Code Explanation:

Once again using the seaborn package, the type of review (positive 1, negative 0) is explored in terms of the class of clothing. Since this is once again a frequency plot, only the x-axis variable “LABEL” is specified. The hue argument is what assigns separate bars and colors to each class type, and the kind argument specifies that the Label variable should be counted by frequency on the y-axis.

Code Interpretation:

Exploring the class variable suggests that the most popular clothing types are: Petite and Anthro, Dresses, Blouses, and Cut and Sew Knits. The distribution of reviews is fairly constant, suggesting that there are not negative nor positive outliers. This statement has been further verified by taking the mean of the label by class group. The results show that no class falls above .80, and the majority rest at .90. Casual bottoms and Chemises scored the highest in this criteria with a 100% positive review rate, however upon investigation this is because only 4 reviews were made in these categories.

Word Clouds:

Now that a general understanding of the bariables have been laid out, I will begin to analysis the customer reviews. Using the categories good and bad, I created two separate word clouds.

Token word frequency and WordCloud

Code:

def cloud(text):

# Setting figure parameters

mpl.rcParams['figure.figsize']=(10.0,10.0)

mpl.rcParams['font.size']=12

mpl.rcParams['savefig.dpi']=100

mpl.rcParams['figure.subplot.bottom']=.1

# Processing Text

stopwords = set(STOPWORDS) # Redundant

wordcloud = WordCloud(width=1600, height=800,

background\_color='black',

stopwords=stopwords,).generate(str(text))

print(wordcloud)

# Output Visualization

fig = plt.figure(figsize=(20,10), facecolor='k')

plt.imshow(wordcloud)

plt.axis('off')

plt.tight\_layout(pad=0)

plt.show()

Code Explanation

This code creates the *word cloud visualization function*. This function’s mathematical processes are hidden, since it does not explicitly state that it determines the frequency occurrence of each word in relation to the entire dictionary of words. Within the function, the *Setting Function Parameter* section creates the graphic structure using matplotlib. Then the text is formatted, and the word frequency is determined. Finally, the matplotlib structure is filled with words, where the larger the word size, the higher the word occurrence.

For this section, I deviated from the book and heavily relied upon the following online resources:

<https://www.kaggle.com/longdoan/word-cloud-with-python>

<https://github.com/amueller/word_cloud/issues/134>

<https://amueller.github.io/word_cloud/auto_examples/masked.html>

Execution of Functions

# Code Execution

# Highly Raited

out = preprocessing(df[df.RATING >= 3]['REVIEW\_TEXT'])

print(cloud(out))

# Low Raited

out = preprocessing(df[df.RATING < 3]['REVIEW\_TEXT'])

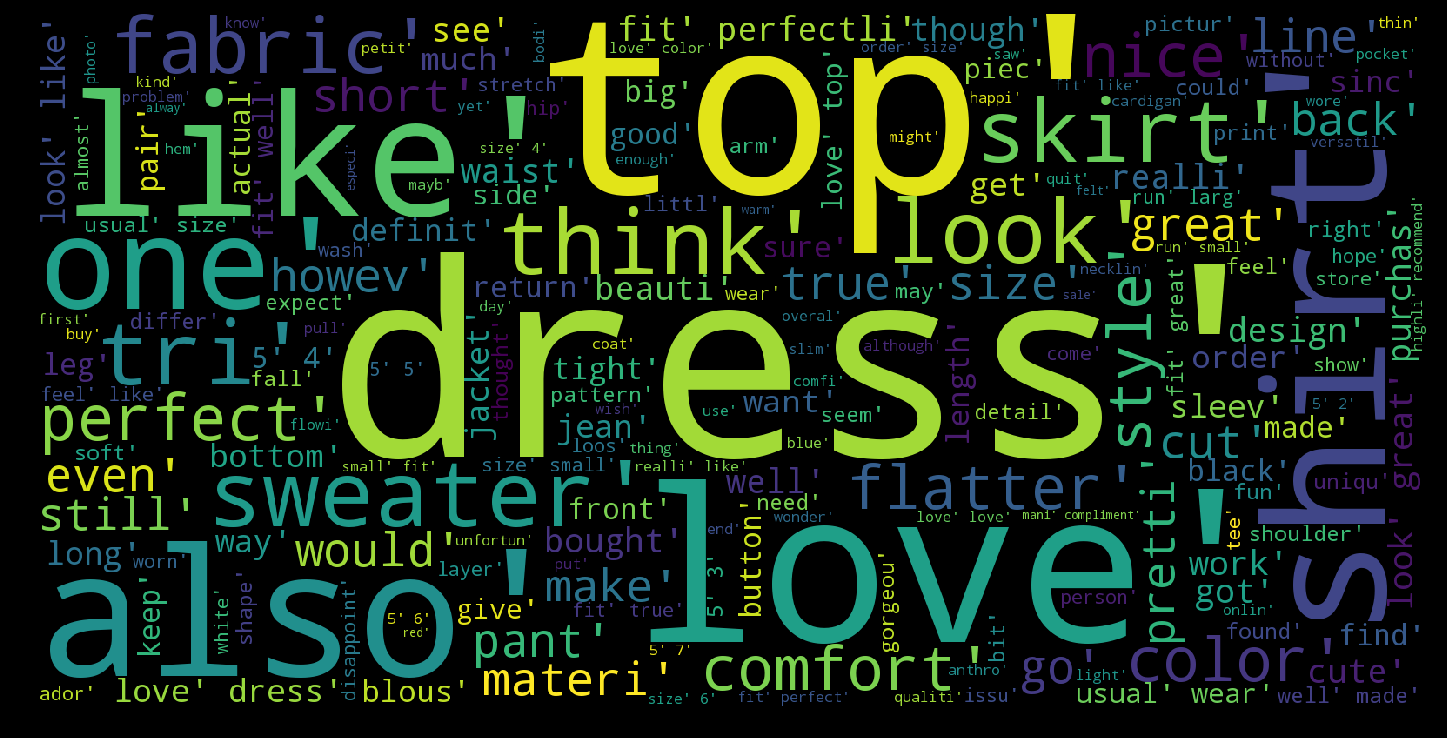
print(cloud(out))

Code Explanation:

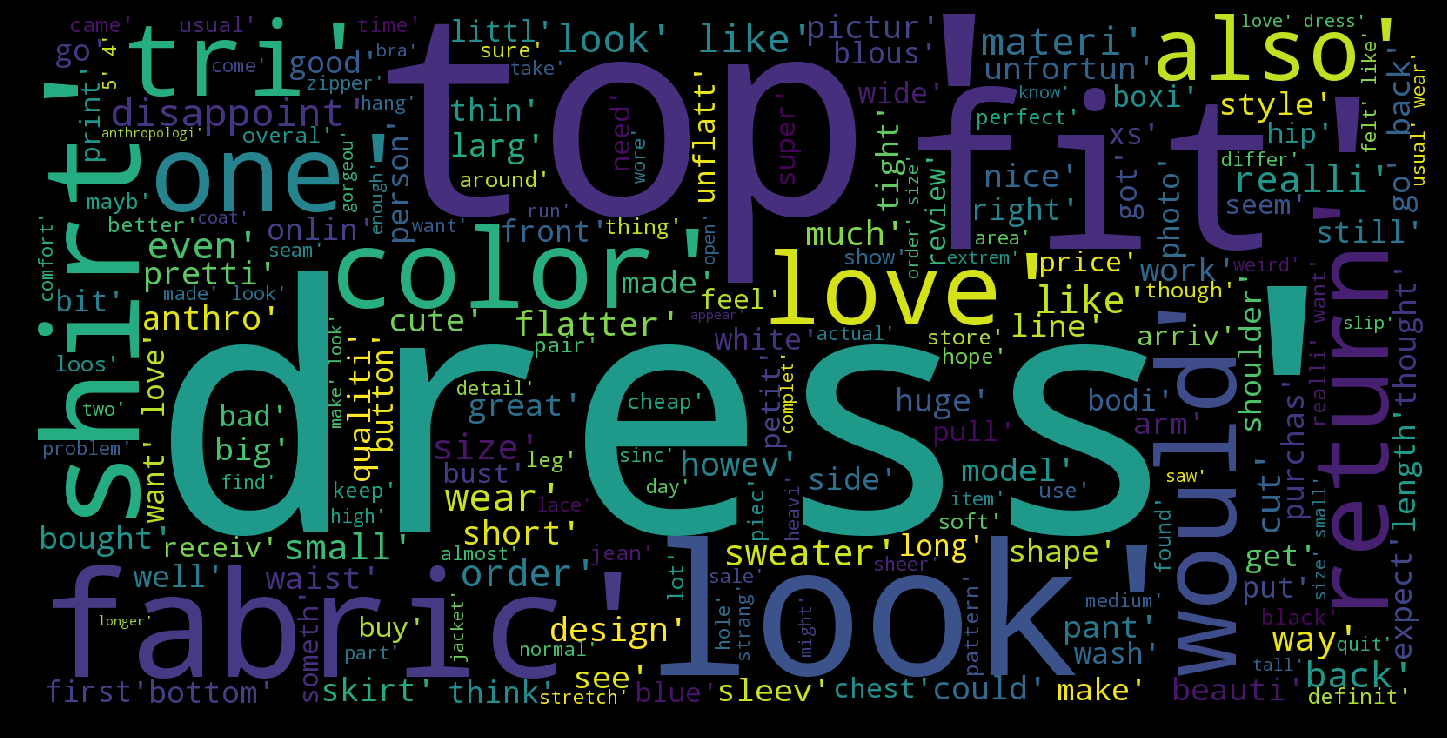
This is the point where all the previous functions are executed on actual data. The process is applied to both positive and negative reviews. Firstly, the data selected and processed using my *preprocessing* function. Note that the highly raited word cloud utilizes reviews with a raiting greater or equal to 3, and negative reviews utilize comments with a raiting under 3. Then, the cloud is created using the cloud function.

Code Output:

**Word Cloud for Positive Reviews:**



**Word Cloud for Negative Reviews:**



Code Interpretation:

At a first glance the most common words overlap significantly between good and bad reviews. Indeed, the observation that “Dress” and “Top” are the most common words is reflected by the disproportionate number of reviews in the dress category. Looking at the rest of the words, the positive reviews tend to use words such as: Love, Comfortable, Great, Style, Small, Flattering, Wear, True. This offers the first big insight because within the expected positive words, the words “True” stands out, since it suggest consistency between customer expectation of the product and what the product actually delivered. In terms of negative reviews, notable words include: unflattering, big, unfortunately appear. These negative words represent the small proportion of words in the negative reviews. This suggests that either people are expressing their criticism with negative prefixes, such as “Not Pretty”, or people are sticking to constructive criticism only, addressing problems of fit or appearance rather than simply expressing anger.

The central flaw of these word clouds is that they only show the distribution of individual words. This removes the context of the word, as well as disregard negative prefixes. In order to solve this problem I will utilize n-grams, which increases the size of observed values from one word to multiple words, enabling frequency counts to be conducted to word sequences. Although I would have prefered to visualize these findings through the use of Word Clouds, I was unable to program this in, thus leaving me with a simple table.

In the section below, the 15th most frequent 2 and 3 gram sequences are on display for both the good and bad reviews.

N-Grams: Gram and Frequency

Code:

def get\_ngrams(text, n):

n\_grams = ngrams((text), n)

return [ ' '.join(grams) for grams in n\_grams]

def gramfreq(text,n,num):

# Extracting bigrams

result = get\_ngrams(text,n)

# Counting bigrams

result\_count = Counter(result)

# Converting to the result to a data frame

df = pd.DataFrame.from\_dict(result\_count, orient='index')

df = df.rename(columns={'index':'words', 0:'frequency'}) # Renaming index column name

# Output

print(df.sort\_values(["frequency"],ascending=[0])[:num])

Code Explanation:

The first function, *get\_ngrams* takes both data and n-gram number as arguments. This code transforms the 1-gram tokenized text to greater number of grams. The second function creates a dataframe which sorts the words by most frequent n-grams.

Code Output:

|  |  |
| --- | --- |
| **Negative** | **Positive** |
|  |  |

Code Interpretation

At this point, fit and product inconsistency strongly emerge as major topics in the reviews. From this information, I can infer that the dataset belongs to a online retailer, since brick and mortar stores have changing rooms to prevent this problem. The central themes in the product reviews brought to light by the n-grams are:

1. **Fit:** Whether the product’s advertised size actually corresponds to customer size and height.
2. **Love or Hate:** The customer's personal feelings towards the product.
3. **Complements:** The customer's social experience wearing the product.
4. **Product consistency:** Whether the product appears as advertised, lives up to quality expectations.

In the negative reviews, customers expresse their disappointment in the product, stating that they “really wanted to love” the item. This signifies that the product did not live up to the customers expectations. This occurred for multiple reasons. “Order wear size” and “Usual wear size” suggest that the fit did not suit their typical universal body size. Perhaps if better product dimension information could be provided, then the likelihood of this negative response could decrease. Furthermore, perhaps the product platform could track the user’s size through previous purchase in order to warn customer for potential size conflict.

Another form of negative review is in the dissapointment in the product turnout. “Too much fabric” and “Looks nothing like” suggest inconsistency with online retail presentation and actual product. These reviews are especially destructive, since they damage the reputation of the store product quality, which is a online platforms biggest asset.

On the other hand, positive reviews are void of criticism, and are preoccupied with confirming fit and sharing social experience with the clothing. “True Size”, “Fit Perfectly”, “Fit like a glove”, on top of the multiple 2-grams with customer’s height suggest that a large part of positive reviews are employed to confirm product fit according to certain size. The high occurrence of this review suggest that height and size is usually a big issue, which this retail managed to consistently satisfy.

“Received many compliments”, “Look forward to wearing”, “Everytime I wear”, “Looks great with jeans” are all comments which reflect the customer's experience wearing the product out in public. This not only express the relevance of trendy, jaw dropping fashion for customers in a social context, but also suggests that the product review are a highly social space, in which customers not only talk with the retailer, but with the other customers as well.

Naive Bayes Algorithm: Extracting meaning using supervised learning

Supervised learning is typically employed to make predictions about the future. However, some simple models may also be opened up to offer some insight. Naive Bayes is a probabilistic model which depends on Bayes theorem to compute the probability of a word's category by looking at its occurrence over the different classes. Since this model looks at both good and bad reviews, it is able to extract the one-gram tokens which best polarize the categories. Using this model, I could potentially predict the positive or negative sentiment of unlabelled reviews.

Code:

# Pre-Processing

data['tokenized'] = data.REVIEW\_TEXT.astype(str).str.lower() # turn into lower case text

data['tokenized'] = data.apply(lambda row: tokenizer.tokenize(row['tokenized']), axis=1) # apply tokenize to each row

data['tokenized'] = data['tokenized'].apply(lambda x: [w for w in x if not w in stop\_words]) # remove stopwords from each row

data['tokenized'] = data['tokenized'].apply(lambda x: [ps.stem(w) for w in x]) # apply stemming to each row

all\_words = nltk.FreqDist(preprocessing(df['REVIEW\_TEXT'])) # calculate word occurence from whole block of text

word\_features= list(all\_words.keys())[:3000] # 3000 most recurring unique words

# Formatting

labtext= list(zip(data.tokenized, (data.LABEL)))

# Data Frame

def find\_features(document):

words = set(document)

features = {}

for w in word\_features:

features[w] = (w in words)

return features

# Apply function to data

featuresets = [(find\_features(text), LABEL) for (text, LABEL) in labtext]

# Fit Model

training\_set = featuresets[:15000]

testing\_set = featuresets[15000:]

%%time

%%memit

classifier = nltk.NaiveBayesClassifier.train(training\_set) # Very scalable algorithm

# Print output

print("Classifier accuracy percent:",(nltk.classify.accuracy(classifier, testing\_set))\*100)

print(classifier.show\_most\_informative\_features(40))

Code Explanation:

Pre-processing must be conducted once more because the supervised learning model must have the comments kept apart and unaggregated, contrary to the previous analysis, which collapse all the comments by the categories of positive or negative.

Supervised learning requires features (indepdent variable) and a label (dependent variable). The *Formatting* section does just this by creating a tuple with the comment and customer rating label. Currently the independent variable is the entire comment. However, in order to the Naïve Bayes Algorithm to work, each word must be treated as a variable. Instead of utilizing sequencial words, the model notes which words are present out of the entire dictionary of words available in the comments corpus. In order to reduce computational intensity, only the top 5000 most common words will be considered, instead of the 9000 unique words in the corpus. The *find\_features* function does just this by checking the presence of words for a piece of text against *word\_features*, a variable created earlier which includes the top 5000 most common words used by customers in this dataset. The *Apply Function to Data* section applies the *find\_features* function to each individual customer review using a loop, while also retaining each review’s label.

The model is trained on the first 15000 reviews, and tested upon the remaining 8000 reviews, where it scored a 89% accuracy rate.

*%%time* and *%%memit* track model memory and training time usage.

Finally, the model is trained, the model accuracy is printed, and the 40 most imformative features are also outputted.

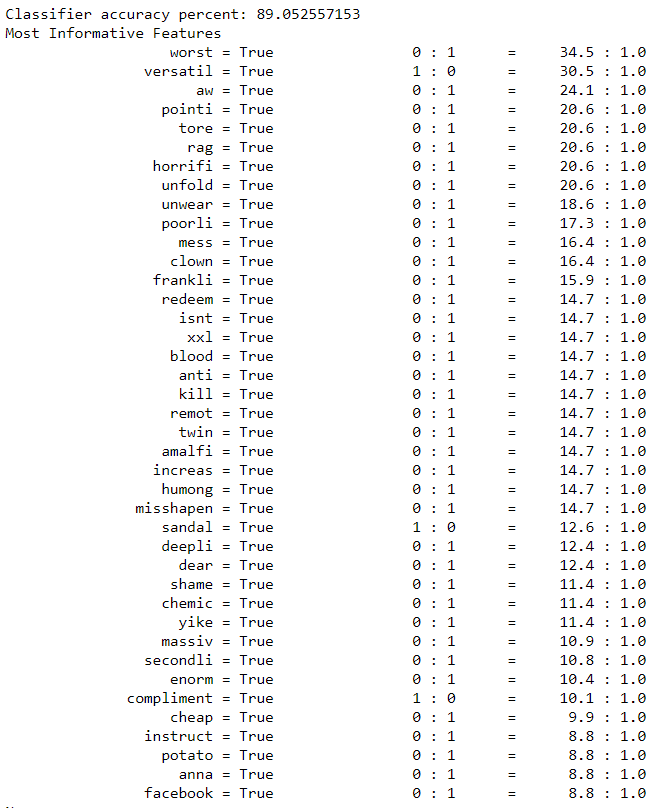
Code Output:

Model Memory: peak memory: 3661.09 MiB, increment: 0.89 MiB

Training Speed: Wall time: 39.3 s

Table Interpretations:

The first column displays the word, the second represents whether the word is negative (0:1), or positive (1:0). Lastly, the third column shows the ratio of occurrence. Looking at the first column, “worst” is a negative word, whose presence indicates the review is 34 times more likely to be negative than positive.This model’s accuracy is 89%. Naive Bayes’ predictive power is limited compared to other, more complex models, but accuracy is not the goal for this analysis.



Code Interpretation:

The forty most polarizing words found by the algorithm are mostly negative words. This is perhaps because the imbalanced distribution of positive and negative review. Since negative words to be less represented, rare words may stand out more. The positive word “Versatile” suggests that cloths which can be matched with a broad wardrobe are deserving of a good review. Product marketing should aim to associate itself with this word, since it is undeniable positive association. The second word “Compliment” is consistent with the n-gram analysis, and once again highlights the value of a positive social experience.

Negative words on the other hand, are much more numerous and suggest a wider range of emotions. “Worst” and “Kill” are fairly self explanatory, outright negative emotion without constructive feedback. “Aw”, “Redeem”, “Misshapen”, “Shame”, and “Dear” evoke much deeper emotions of disappointment and sadness, but since the customers are expressing a more rational, critical response, perhaps redemption on the part of the retailer is possible, contrary to hateful customer’s reviews. For these reasons, a classification model which can flag negative reviews of this sort may not only convert this disappointed customer into a loyal one, but it may also serve as a strong PR move which can boost the positivity in the comment section, which has proven to an influential social space for customers.

At the bottom of the Naive Bayes’ most polarized words are the negative words “Potato” and “Facebook”. This is quite comical, since these words suggest a divergence from the traditional review vocabulary into the lingo of internet memes and social media. Indeed, teenagers and young adults who frequently image sharing sites like to label crappy, passive things as “potato”.

Word2Vec: Neural Network

In order to gain more insight about these words, Word2Vec will be utilized. This model uses a shallow neural network to find words which emerge in similar context. By embedding words through multiple dimensions, similar words may be extracted. My analysis fetched the ten most similar words.

Code:

# Directory

os.chdir(r"D:\My Computer\DATA\Retail")

# Download Google’s model from <https://code.google.com/archive/p/word2vec/>

# Load Google's pre-trained Word2Vec model.

model = gensim.models.KeyedVectors.load\_word2vec\_format('GoogleNews-vectors-negative300.bin.gz', binary=True)

# Apply Model

w2vec = word2vec.Word2Vec(data["tokenized"], min\_count=5, size=200)

# Output

# Negative

w2vec.most\_similar(["shame"],topn=10)

w2vec.most\_similar(["potato"],topn=10)

w2vec.most\_similar(["worst"],topn=10)

w2vec.most\_similar(["rag"],topn=10)

# Positive

w2vec.most\_similar(["versatil "],topn=10)

w2vec.most\_similar(["compliment"],topn=10)

w2vec.most\_similar(["love"],topn=10)

Code Explanation:

This section requires Google’s pre-trained word2vec model to be downloaded. The file size is 1.5 GB because it holds 3 million 300-dimension english word vectors. For these reasons, storage of the model is on separate hard drive, requiring a change in working directory. After the *gensim* word2vec model is loaded, it is applied to the retail dataset. Finally, the top most similar words are outputted for seven different words.

Code Output and Interpretation:

|  |  |
| --- | --- |
| **Negative Word** | **Similar Words** |
| shame | [('mainli', 0.9287949800491333),  ('reluctantli', 0.9152017831802368),  ('crook', 0.91181480884552),  ('de', 0.899941086769104),  ('integr', 0.897300124168396),  ('clip', 0.8968260288238525),  ('wonki', 0.8967447280883789),  ('jut', 0.8963980674743652),  ('none', 0.8952524662017822),  ('repair', 0.8950567245483398)] |
| potato | [('circu', 0.9277945160865784),  ('muumuu', 0.8956434726715088),  ('boxier', 0.8774730563163757),  ('giant', 0.8756405711174011),  ('sack', 0.873295783996582),  ('moo', 0.8720555305480957),  ('frankli', 0.8717751502990723),  ('grandma', 0.8660542964935303),  ('jut', 0.8647107481956482),  ('wing', 0.8616272211074829)] |
| worst | [('wonki', 0.9466806650161743),  ('reduc', 0.9427450299263),  ('seriou', 0.9401675462722778),  ('broken', 0.9369989633560181),  ('nightmar', 0.9369844794273376),  ('mainli', 0.9363988041877747),  ('embarrass', 0.9341127276420593),  ('bear', 0.9329415559768677),  ('scissor', 0.9302538633346558),  ('couch', 0.929938793182373)] |
| rag | [('bathrob', 0.9240034818649292),  ('hospit', 0.8915389776229858),  ('becau', 0.8900049924850464),  ('nightgown', 0.8853806853294373),  ('moo', 0.8792257308959961),  ('frequent', 0.8777855634689331),  ('sausag', 0.8763315081596375),  ('couldnt', 0.8722333908081055),  ('horrifi', 0.8718711137771606),  ('grate', 0.8691390752792358)] |

The word “Shame” is consistent with the emotion of disappointment, since it is associated with “Reluctantly” and “Repair”, which also suggest potential redemption. Word2Vec is great because it may be used to improve the detection of salvageable customers through broadening the words relevant to disappointment. “Worst”, “Rag”, and “Potato” are all similar to strictly negative words, although the latter two often have a comedic edge to them.

|  |  |
| --- | --- |
| **Word** | **Similar Words** |
| versatil | [('casual', 0.8894511461257935),  ('classi', 0.8763369917869568),  ('fun', 0.8689156174659729),  ('statement', 0.8675277829170227),  ('everyday', 0.8622598052024841),  ('dressi', 0.8553286790847778),  ('varieti', 0.83147132396698),  ('fanci', 0.8195697069168091),  ('throw', 0.8194319009780884),  ('chic', 0.8192055821418762)] |
| compliment | [('complement', 0.8592609763145447),  ('ton', 0.849358320236206),  ('ts', 0.7784327268600464),  ('numer', 0.7634919881820679),  ('galor', 0.6781098246574402),  ('stranger', 0.652309000492096),  ('mani', 0.6503288745880127),  ('load', 0.6384847164154053),  ('countless', 0.6369271278381348),  ('friend', 0.6330838799476624)] |
| love | [('ador', 0.841651201248169),  ('amaz', 0.7125707864761353),  ('gorgeou', 0.7103675603866577),  ('fabul', 0.6741998195648193),  ('beauti', 0.6554014086723328),  ('fantast', 0.6346416473388672),  ('classic', 0.6297193765640259),  ('wonder', 0.6152561902999878),  ('fun', 0.6076583862304688),  ('sweet', 0.6016392707824707)] |

In terms of positive words, “Compliments” has an interesting combination of words. Curiously, its association with “stranger” is a bit higher than “friend”, suggesting more instances of compliments from strangers. Otherwise, the most associated words are adjectives that signify quantity, such as “tons”, “many”, “load”, and “countless”.

Conclusion:

In the end, the most valuable findings came out of the n-grams, and the subsequent tools provided additional support to the findings. The n-grams highlighted the general themes of the reviews, including product fit, product compliments and social factor, and the product’s consistency. Fit is the product's biggest flaw since online shoppers are required to purchase without the ability to try. Since the customer often post their height and whether the product matches their size, further information could be mined in order to deter certain body heights and size from purchasing the product, or at least offer free returns in order to maintain customer confidence.

Since the n-gram analysis also suggests that a portion of negative reviews are motivated by product inconsistent with advertised version, a supervised learning algorithm (sentiment tracker) could be implemented in the real time in order to flag and promptly replace the flawed product mentioned in the complaints, or find other remedies. These comments are important to subdue because they offer the biggest threat to the retailer’s customer confidence, its reputation.

Another significant finding from this analysis is the importance of the social space created by the reviews system. These customers share their personal experience, suggesting that they are addressing other customers as well as the retailer. This could perhaps be capitalized on by increasing social capabilities of the review system, such as enable replies from other customers as well as open public dialogue with the retailer. This would enable the quick response to product inconsistency to become publicized, demonstrating to other users the initiative taken by the retailer. It would also provide more review data for collection, further improving the retailer’s understanding of it customer’s behavior. In order to prevent users to respond in mean or harsh ways, a filtering system should be incorporated to keep the space civil.

The n-gram analysis also pointed out that positive reviews often times emphasize compliments from strangers. Since the Naive Bayes also pointed out hyper positive words such as “versatile”, this information may be utilized in the marketing of the products. Furthermore, it demonstrates meaningful aspects of the product that should be utilized in customer testimonials.

Improvements:

This analysis was conducted at a fairly general level. Its core purpose was to explore the tools Python has to offer in the realm of Natural Language Processing. However, further analysis may be conducted, such as track down which class of clothing suffers the highest rates of product inconsistency, or perhaps which type of product receives the most compliments.

Although the PorterStemmer helped combine words of different tenses and plurality, it also makes the report harder to interpret, especially by decision makers unfamiliar with the technology. In terms of interpretability, the use of data tables may also be replaced by more intuitive visualization to increase understanding and rhetoric of communication.

Further variables may also be explored, such as department ID, or the level of activity of the customer at hand. This may offer insight on the customer satisfaction of certain departments, as well as the loyalty of the customers. A deeper dive in the clothing type may also have offered design insight, where customers criticism and praise are taken into account in the next product cycle.

Lastly, a lexicon for sentiment analysis could be built in order to extend the analysis of reviews to unlabeled facebook comments or tweets. The more data, the better the understanding of the situation.