Unsupervised Text Classification of Amazon Reviews

This is the final project for DTSA 5799 (Unsupervised Text Classification For Marketing Analytics)

Reviews of Nike products with less than 4 stars are extracted from the Amazon review dataset and split into topics using k-means clustering in an effort to gain insight on what may influence a customer to leave a negative review.

You can also view the project on my GitHub repository here: https://github.com/arwhit/reviews-topic-modeling

```
#Install and Import Required Packages
!pip install textblob
!pip install -U scikit-learn
import gzip
import json
import requests
import numpy as np
from textblob import TextBlob
import spacy
import re
import sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
```

```
Requirement already satisfied: textblob in /usr/local/lib/python3.10/dist-packages (0.17.1)
Requirement already satisfied: nltk>=3.1 in /usr/local/lib/python3.10/dist-packages (from textblob) (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (2024.5.15)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (4.66.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.0)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.25.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
```

Data Extraction and Exploration

Note: This section describes the process of extracting Nike reviews with less than 4 stars for topic modeling following much of the methodology established in the course labs.

While you can follow along using the reference files provided in class and saving them to your Google Drive, the default behavior utilizes pre-extracted data hosted in a public Git repository. This approach reduces runtime and ensures usability even if the original reference files become unavailable or their URLs change.

```
#Set boolean value to determine starting point
full_extract=False
#Case for full extraction with matching google drive file path (following tutorial from lab coursework)
if full_extract==True:
  #mount drive
  from google.colab import drive
  drive.mount('/content/drive')
  #extract asins
  asins = []
  with gzip.open("/content/drive/MyDrive/Masters_Degree/Text_Marketing_Analytics/HW_Files/meta_Clothing_Shoes_and_Jewelry.gz") as products:
    for product in products:
        data = json.loads(product)
        categories = []
        for category_list in data.get("categories", []):
            _catlist = []
            for item in category_list:
                _catlist.append(item.lower())
            categories += _catlist
        if "nike" in categories:
            asins.append(data["asin"])
  #extract reviews with less than 4 stars
  bad_reviews = {}
  with gzip.open("/content/drive/MyDrive/Masters_Degree/Text_Marketing_Analytics/HW_Files/reviews_Clothing_Shoes_and_Jewelry.gz") as reviews:
    for i, review in enumerate(reviews):
        review = json.loads(review)
        if review["overall"] < 4 and review["asin"] in asins:
          _id = "%s.%s" % (review["asin"], review["reviewerID"])
          if review['asin'] in asins:
            bad_reviews[_id]=review
  print(len(bad_reviews), ' bad reviews extracted for further analysis')
  #write to file
  with open("/content/drive/MyDrive/Masters_Degree/Text_Marketing_Analytics/HW_Files/extracted_bad_reviews.jsonl", "w") as bad_reviews_file:
    for k, v in bad_reviews.items():
        review = json.dumps(v)
        bad_reviews_file.write(f"{review}\n")

#Case for referencing extracted reviews from git repository
else:
  extracted_bad_reviews=[]
  #specify url of jsonl file with the bad reviews
  url="https://raw.githubusercontent.com/arwhit/reviews-topic-modeling/main/extracted_bad_reviews.jsonl"
  response = requests.get(url)
  #Iterate through each line:
  for line in response.text.splitlines():
    review = json.loads(line)
    review_text=review["reviewText"]
    extracted_bad_reviews.append(review_text)

  print(len(extracted_bad_reviews), 'bad reviews have been extracted. Here is an example: \n')
```

```
print(extracted_bad_reviews[0])
```

```
3903 bad reviews have been extracted. Here is an example:

the colour i received is not blue as shown but yellow.Couldnt change it because it was a birthday present for my daughter and havent got time.She really
```

Preprocessing

A total of 3,903 negative reviews were extracted. Manually labeling such a large volume of data would be time-consuming, highlighting the potential of unsupervised classification as an initial approach to uncover categories within the reviews.

To optimize model performance and reduce the document-term matrix (DTM) size, we employ the following pre-processing steps:

- **Lowercase:** This will help ensure words from different reviews that have different cases will be identified as the same
- **Spellcheck:** This data set has many spelling error. For example "didn,t" instead of "didn't" in the first review. We will use Textblobs built in corrector to address this.
- **Lemmatization:** This process converts words to their base form (e.g., "running" becomes "run"). Lemmatization helps capture the underlying meaning of words and improves model generalizability.
- **Stopword removal:** We eliminate common words with minimal semantic meaning (e.g., "the", "a", "and"). Removing stopwords reduces noise and allows the model to focus on content-bearing words.
- **Special character and punctuation removal:** Non-alphanumeric characters and punctuation are removed from the text (e.g., "@", "$", "?"). This step ensures the model processes meaningful textual content.

```
#Create Empty List for Cleaned Reviews
preprocessed_bad_reviews=[]

#Specify lemmatizer
spacy.require_gpu()
lemmatizer = spacy.load('en_core_web_sm')

#Iterate through list and preprocess the reviews
for review in extracted_bad_reviews:
  #lowercase
  review = review.lower()
  #spellcheck
  blob = TextBlob(review)
  corrected_review = blob.correct()
  filtered_text = str(corrected_review)
  #lemmatize
  doc = lemmatizer(filtered_text)
  #lemmatized_text = " ".join([token.lemma_ for token in doc])
  #Remove Stopwords
  filtered_words = [token.text for token in doc if not token.is_stop]
  filtered_text = " ".join(filtered_words)
  #remove special characters and punctuation
  punc_removed_text = re.sub(r"[^\w\s]", ' ', filtered_text)
  preprocessed_bad_reviews.append(punc_removed_text)

print('All reviews have been preprocessed. Here is an example: \n')
print(preprocessed_bad_reviews[0])
```

```
All reviews have been preprocessed. Here is an example:

colour received blue shown yellow couldn change birthday present daughter haven got time she didn   t like
```

We now proceed to create the document-term matrix (DTM) that will serve as input for the topic models. Following common practice in unsupervised text classification, we employ Term Frequency-Inverse Document Frequency (TF-IDF) to weight terms within the DTM.

To further reduce the matrix size and focus on informative terms, we adopt the following filtering strategies:

- **Exclusion of common words:** We disregard words appearing in more than 50% of the reviews, as these are likely to be stop words with minimal discriminatory power.
- **Rare word removal:** We exclude words that occur in less than 5 reviews, as their infrequency limits their contribution to topic identification.

```
tfidf_vectorizer = TfidfVectorizer(max_df=0.5, min_df=5, stop_words='english')
tfidf = tfidf_vectorizer.fit_transform(preprocessed_bad_reviews)
print(f"n_samples: {tfidf.shape[0]}, n_features: {tfidf.shape[1]}\n")
print(f"Nonzero terms: {100*tfidf.nnz / np.prod(tfidf.shape):.3f}%")
```

```
n_samples: 3903, n_features: 2077

Nonzero terms: 0.872%
```

That's still a lot of features and nonzero terms. We will further reduce the number of features using Single Value Decomposition (SVD) to make the matrix less sparse in an attempt to avoid the "curse of dimmensionality". This should also help reduce the runtime.

```
#Reduce to 100 Components
reduced = make_pipeline(TruncatedSVD(n_components=100, random_state=5), Normalizer(copy=False))
SVD_reduced = reduced.fit_transform(tfidf)
explained_variance = reduced[0].explained_variance_ratio_.sum()
print(f"n_components:{100}, explained variance: {100*explained_variance:.1f}%")
```

```
n_components:100, explained variance: 32.1%
```

We see over 30% of the feature space can be represented by only 100 components.

Data Modeling

Now that the data has been vectorized and reduced we can start topic modeling. We will attempt to use k-means clustering as an usupervised learning approach. Since we do not have labels for the data, we need to establish some other metrics to determine the ideal model to use. We will attempt to use both the within clustter sum of squares and silloute score to select the optimal numbers of clusters

```
#Fit models from 2 to 10 cluster size
n_clusters=list(range(2,11))
WCSS=[]
Silhouette=[]
for clusters in n_clusters:
  kmeans = KMeans(
    n_clusters=clusters,
    max_iter=100,
    n_init=1,
    random_state=5)
  kmeans.fit(SVD_reduced)
  #Record the Within-Cluster Sum of Squares
  WCSS.append(kmeans.inertia_)
  #Record the Silhouette Score
  Silhouette.append(silhouette_score(SVD_reduced, kmeans.labels_))
```

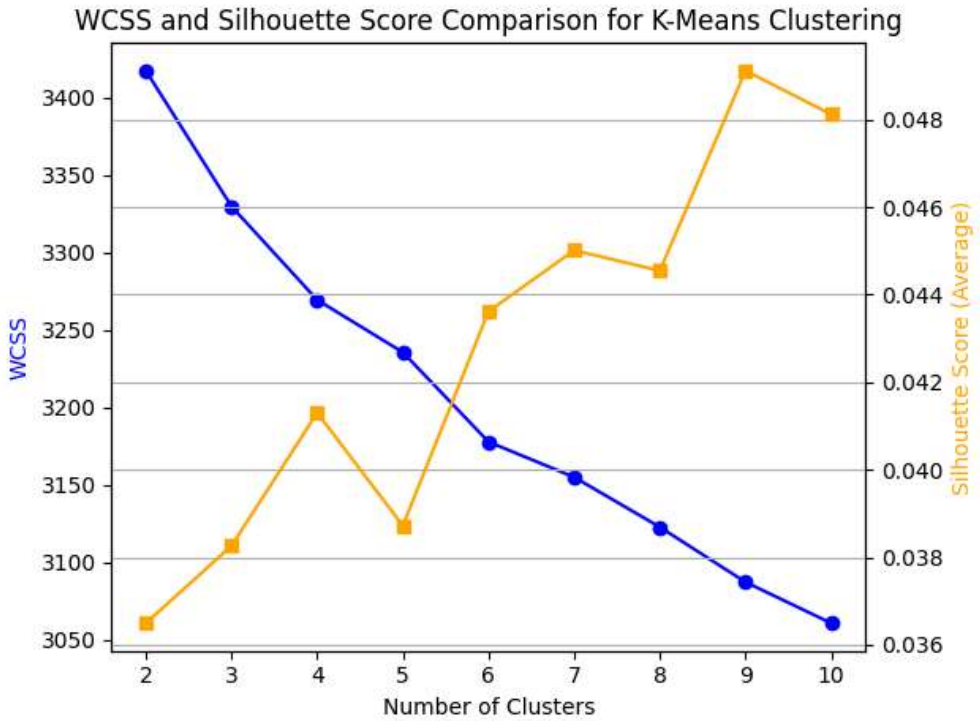
```
# Define distinct colors and markers for WCSS and Silhouette
wcss_color = 'blue'
wcss_marker = 'o'
silhouette_color = 'orange'
silhouette_marker = 's'

# Create the figure and primary axis
fig, ax1 = plt.subplots()

# WCSS plot on primary axis
ax1.set_xlabel('Number of Clusters')
ax1.set_ylabel('WCSS', color=wcss_color)
ax1.plot(n_clusters, WCSS, marker=wcss_marker, linestyle='-', color=wcss_color, label='WCSS')

# Silhouette Score plot on secondary axis (using twinx)
ax2 = ax1.twinx()
ax2.set_ylabel('Silhouette Score (Average)', color=silhouette_color)
ax2.plot(n_clusters, Silhouette, marker=silhouette_marker, linestyle='-', color=silhouette_color, label='Silhouette Score')

# Customize plot appearance for readability
plt.title('WCSS and Silhouette Score Comparison for K-Means Clustering')
plt.xticks(n_clusters)  # Set x-axis ticks for clarity
plt.grid(True)
plt.tight_layout()
plt.show()
```



Looking at the plot of WCSS and Silhouette score it appears 7 clusters is optimal for the k-means model. Let's extract the top words and see if they are meaningful. We will define a helpful word plotting funciton adapted from the sklearn documentaiton (see sources section)

```
#Use optimal number of clusters
kmeans = KMeans(
    n_clusters=7,
    max_iter=100,
    n_init=1,
    random_state=5)
kmeans.fit(SVD_reduced)

#Convert the Model back to words and explore the top words in each cluster
```

Model Evaluation and Summary