Homework 4

Due: Wed May 13 @ 11:59pm

In this homework we will covering NLP, Topic Modeling and Recommendation Engines

We will generate recommendations on products from a department store based on product descriptions. We'll first transform the data into topics using Latent Dirichlet Approximation, and then generate recommendations based on this new representation.

Instructions Follow the comments below and fill in the blanks (__) to complete.

Please 'Restart and Run All' prior to submission.

When submitting to Gradescope, please mark on which page each question is answered.

Out of 26 points total.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=DeprecationWarning)

%matplotlib inline
np.random.seed(123)
```

LDA and Recommendation Engines

We are going to create a recommendation engine for products from a department store.
The recommendations will be based on the similarity of product descriptions.
We'll query a product and get back a list of products that are similar.
Instead of using the descriptions directly, we will first do some topic modeling using LDA to transform the descriptions into a topic space.

Transform product descriptions into topics and print sample terms from topics

In [2]:
```
# 1. (2pts) Load the Data

# The dataset we'll be working with is a set of product descriptions f
rom JCPenney.

# Load product information from ../data/jcpenney-products_subset.csv.z
ip
# This is compressed version of a csv file.
# Use pandas read_csv function with the default parameters.
# read_csv has a parameter compression with default value 'infer' that
will handle unzipping the data.
# Store the resulting dataframe as df_products.
df_products = pd.read_csv("../data/jcpenney-products_subset.csv.zip")

# print a summary of df_products using .info, noting the number of rec
ords (should be 5000)
df_products.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 6 columns):
uniq_id          5000 non-null object
sku              5000 non-null object
name_title       5000 non-null object
description      5000 non-null object
category         4698 non-null object
category_tree    4698 non-null object
dtypes: object(6)
memory usage: 234.5+ KB
```

In [3]:
```
# 2. (2pts) Print an Example

# The two columns of the dataframe we're interested in are:
#   name_title which is the name of the product stored as a string
#   description which is a description of the product stored as a stri
ng
#
# We'll print out the product in the first row as an example
# If we try to print both at the same time, pandas will truncate the s
trings
#   so we'll print them seperately

# print the product name_title in row 0 of df_products
print(df_products.name_title[0])

# print the product desciption in row 0 of df_products
print(df_products.description[0])
```

```
Alfred Dunner® Essential Pull On Capri Pant
You'll return to our Alfred Dunner pull-on capris again and again wh
en you want an updated, casual look and all the comfort you love.
elastic waistband approx. 19-21" inseam slash pockets polyester wash
able imported
```

In [4]:
```
# 3. (4pts) Transform Descriptions using TfIdf

# In order to pass our product descriptions to the LDA model, we first need to vectorize from strings to
#   fixed vectors of floats.
# To do this we will transform our documents into unigrams using Tf-Idf,
#    use both unigrams and bigrams
#    excluding terms which appear in less than 10 documents
#    excluding common English stop words and

# Import TfidfVectorizer from sklearn.feature_extraction.text
from sklearn.feature_extraction.text import TfidfVectorizer

#  Instantiate a TfidfVectorizer with
#   ngram_range=(1,2),
#   min_df=10,
#   stop_words='english'
# Store as tfidf
tfifd = TfidfVectorizer(ngram_range=(1,2),
                        min_df=10,
                        stop_words='english')

# fit_transform tfidf on the descriptions column of our dataframe, creating the transformed dataset X_tfidf
# Store as X_tfidf
X_tfidf = tfifd.fit_transform(df_products.description)

# Print the shape of X_tfidf (should be 5000 x 3979)
print(X_tfidf.shape)
```

```
(5000, 3979)
```

In [5]:
```
# 4. (3pts) Format Bigram Labels and Print Sample of Extracted Vocabul
ary

# The extracted vocabulary cat be retrieved from tfidf as a list using
get_feature_names()
# Store the extracted vocabulary as vocabulary
vocabulary = tfifd.get_feature_names()

# Sklearn joins bigrams with a space character.
# To make output easier to read, replace all spaces in our vocabulary
list with underscores.
# To do this we can use the string replace() method.
# For example x.replace(' ','_') with replace all ' ' in x with '_'.
# Store the result back in vocabulary.
vocabulary = [string.replace(' ', '_') for string in vocabulary]

# Print the last 5 terms in the vocabulary
print(vocabulary[-5:])
```

```
['zipper_pockets', 'zippered', 'zippers', 'zirconia', 'zone']
```

In [6]:
```
# 5. (4pts) Perform Topic Modeling with LDA

# Now that we have our vectorized data, we can use Latent Direchlet Al
location to learn
#   per-document topic distributions and per-topic term distributions.
# Though there are likely more, we'll model our dataset using 20 topic
s to keep things small.
# We'd like the model to run on all of the cores available in the mach
ine we're using.
#    `n_jobs` tells the model how many cores to use, while `n_jobs=-1`
indicates use all available.
# We'd also like the results to always be the same, so set random_stat
e=123

# Import LatentDirichletAllocation from sklearn.decomposition
from sklearn.decomposition import LatentDirichletAllocation

# Instantiate a LatentDirichletAllocation model with
#    n_components=20, n_jobs=-1, random_state=123
# Store as lda
lda = LatentDirichletAllocation(n_components=20,
                                n_jobs=-1,
                                random_state=123)

# Run fit_transform on lda using X_tfidf.
# Store the output (the per-document topic distributions) as X_lda
# NOTE: this step may take a minute or more depending on your setup.
X_lda = lda.fit_transform(X_tfidf)

# Print the shape of the X_lda (should be 5000 x 20)
print(X_lda.shape)
```

```
(5000, 20)
```

In [7]:
```
# 6. (4pts) Print Top Topic Terms

# To get a sense of what each topic is composed of, we can print the m
ost likely terms for each topic.
# We'd like a print statement that looks like this:
#    Topic #0 upper sole rubber synthetic rubber_sole
#
# For each topic print 'Topic #{idx} ' followed by the top 5 most like
ly terms in that topic.
# Hints:
#   Use vocabulary created above, but first convert from a list to np.
array to make indexing easier
#   The per topic term distributions are stored in model.components_
#   np.argsort returns the indices of an np.array sorted by their valu
e, in ascending order
#   [::-1] reverses the order of an np.array

for i in range(20):
    tmp = np.argsort(lda.components_)[::-1][i,:5]
    print(f'Topic #{i}', end=' ')
    print(' '.join(np.array(vocabulary)[tmp]))
```

```
Topic #0 screwdriver_needed belts_size inside_waistband inseam_rayon
inseam_petites
Topic #1 diamondscolor alfred hard_anodized stays_tucked recommended
_improve
Topic #2 grommet mattress_longer sleeves_vents sleeves_regular sleev
es_left
Topic #3 sock_size button_cuffs button_fly button_shirt button_zip
Topic #4 scoopneck_sleeveless underwire stoneware_construction stone
ware bodice
Topic #5 drawstring_seam thickness_8mm brushes buckle_movement buckl
emovement
Topic #6 prong_gallery powder pound_weight pound potassium_sorbate
Topic #7 closure_synthetic color_clarity color_enhanced color_multic
olor receive_refund
Topic #8 phillips phillips_screwdriver pillowcases_king plated_14k p
lated_sterling
Topic #9 resistant_rug running_shoes rubberwood rubber_toe collar_re
inforced
Topic #10 installation closure_inside innerspring infused closure_sl
ash
Topic #11 panels_sold underfoot ring_length ringdimensions ringdimen
sions_18
Topic #12 dress_shirt chainpendant seams_lay hills_polo tone_metalba
ck
Topic #13 palmitate energy_star enhanced_weight shut ethylhexylglyce
rin
Topic #14 misses_long mugs_stoneware gold_sterling sale_salon gold_s
tones
Topic #15 suede_upper textile_upper durable_polypropylene durable_ru
bber durable_stoneware
Topic #16 placket_cotton backing_latex short_27 flat_pocket shock_re
duce
Topic #17 propylene pores polystyrene rubber_toe rubberwood
Topic #18 shams_cotton king_cal polyester_shams black_brown king_com
forter
Topic #19 oxide shorts_arizona short_28 short_27 enhanced_weight
```

Generate recommendations using topics

In [8]:
```
# 7. (3pts) Generate Similarity Matrix

# We'll use Content Filtering to make recommendations based on a query
product.
# Each product will be represented by its LDA topic weights learned ab
ove.
# We'd like to recommend similar products in LDA space.
# We'll use cosine_similarity as measure of similarity.

# From sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics.pairwise import cosine_similarity

# Use cosine_similarity to generate similarity scores on our X_lda dat
a
# Store as similarities.
# NOTE: we only need to pass X_lda in once,
#   the function will calculate pairwise similarity for all elements i
n that matrix
similarities = cosine_similarity(X_lda)

# print the shape of the similarities matrix (should be 5000x5000)
print(similarities.shape)
```

```
(5000, 5000)
```

In [9]:
```
# 8.(4pts) Generate Recommendations

# Let's test our proposed recommendation engine using the product at r
ow 0 in df_products.
#   The name of this product is "Alfred Dunner® Essential Pull On Capr
i Pant"

# Print the names for the top 10 most similar products to this query.
# Suggested way to do this is:
#   get the cosine similarities from row 0 of the similarities matrix
#   get the indices of this array sorted by value using np.argsort
#   reverse the order of these indices (remember, we want high values
and np.argsort evaluates ascending)
#   get the first 10 indexes from this reversed array
#   use those indices to index into df_products.name_title and print t
he result

# HINT: The first two products should be:
#   'Alfred Dunner® Essential Pull On Capri Pant', (the original query
product)
#   'Alfred Dunner® Pull-On Pants - Plus',

df_products.name_title[np.argsort(similarities[0,])[::-1][:10]]
```

Out[9]:
```
0              Alfred Dunner® Essential Pull On Capri Pant
2091                   Alfred Dunner® Pull-On Pants - Plus
662                            Alfred Dunner® Pull On Pant
2973                 Levi’s® 511™ Slim Fit Jeans - Boys 4-7x
3251      Arizona Twill Camo Cargo Shorts - Boys 8-20, S...
3637                 Love Indigo Turquoise Back Pocket Capris
858                    Liz Claiborne® Emma Ankle Pants - Plus
2562                  Liz Claiborne® Pajama Pants - Petite
4814                               adidas® 3G Speed Shorts
83                           Stylus™ Crossover Ankle Pants
Name: name_title, dtype: object
```