Homework 4

Due: Fri Dec 2nd @ 11:59pm ET

NLP: Recommendations and Sentiment Analysis

In this homework we will perform two common NLP tasks:

1. Generate recommendations for products based on product descriptions using an LDA topic model.
2. Perform sentiment analysis based on product reviews using sklearn Pipelines.

Instructions

- Replace Name and UNI in the first cell and filename
- Follow the comments below and fill in the blanks (____) to complete.
- Where not specified, please run functions with default argument settings.
- Please **'Restart and Run All'** prior to submission.
- **Save pdf in Landscape** and **check that all of your code is shown** in the submission.
- When submitting in Gradescope, be sure to **select which page corresponds to which question.**

Out of 50 points total.

Part 0: Setup

In [256…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

In [257…

```
# 1. (2pts total) Homework Submission

# (1pt) The homework should be spread over multiple pdf pages, not one single pd
# (1pt) When submitting, assign each question to the pdf page where the solution
#        If there is no print statement for a question, assign the question to t
#        page where the code for the question is visible.
```

Part 1: Generate Recommendations from LDA Transformation

In this part we will transform a set of product descriptions using TfIdf and LDA topic modeling to generate product recommendations based on similarity in LDA space.

Load data and transform text using TfIDF

In [258…

```
# 2. (1pts) Load the Data

# The dataset we'll be working with is a set of product descriptions
#   from the JCPenney department store.

# Load product information from ../data/jcpenney-products_subset.csv.zip
# Use pandas read_csv function with the default parameters.
# Note that this is a compressed version of a csv file (has a .zip suffix).
# .read_csv() has a parameter 'compression' with default
#     value 'infer' that will handle unzipping the data for us.
# Store the resulting dataframe as df_jcp.
df_jcp = pd.read_csv('../data/jcpenney-products_subset.csv.zip',compression='inf

# print a summary of df_jcp using .info()
# there should be 5000 rows with 2 columns with no missing data
df_jcp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   name_title   5000 non-null   object
 1   description  5000 non-null   object
dtypes: object(2)
memory usage: 78.2+ KB
```

In [259…

```
# 3. (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 string
#
# 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 strings
#   so we'll print them seperately

# print the name_title column in row 0 of df_jcp
print(df_jcp['name_title'].iloc[0])

# printing a line of dashes
print('-'*50)

# print the desciption column in row 0 of df_jcp
print(df_jcp['description'].iloc[0])
```

```
Invicta® Sl Rally Mens Black Leather Strap Chronograph Watch 16012
--------------------------------------------------
A timepiece you can enjoy every day of the week, this sports car-inspired chrono
graph watch packs plenty of information into an easy-to-read dial.   Brand: Invi
cta Dial Color: Black Strap: Black leather Clasp: Buckle Movement: Quartz Water
Resistance: 100m Case Width: 48mm Case Thickness: 13.5mm Bracelet Dimensions: 21
```

```
0mm long; 22mm wide Model No.: 16012 Special Features: Stopwatch; 3 multifunctio
n sub dials   Jewelry photos are enlarged to show detail.
```

In [260…

```python
# 4. (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 length vectors of floats.
# To do this we will transform our documents into a TfIdf representation.

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

#  Instantiate a TfidfVectorizer that will
#    use both unigrams + bigrams
#    exclude terms which appear in less than 10 documents
#    exclude terms which appear in more than 10% of the documents
#    all other parameters leave as default
# Store as tfidf
tfidf = TfidfVectorizer(ngram_range = (1,2),
                        min_df = 10,
                        max_df = 0.1)

# fit_transform() tfidf on the description column of df_jcp,
#    creating the transformed dataset X_tfidf
# Store as X_tfidf
X_tfidf=tfidf.fit_transform(df_jcp['description'])

# Print the shape of X_tfidf (should be 5000 x 5678)
print(X_tfidf.shape)
```

```
(5000, 5678)
```

In [261…

```python
# 5: (1pts) Show The Terms Extracted From Row 0

# X_tfidf is a matrix of floats, one row per document, one column per vocab term
# We can see what terms were extracted, and kept, for the document at df_jcp row
#   using the .inverse_transform() function
# Print the result of calling:
#   the .inverse_transform() function of tfidf on the first row of X_tfidf
# You should see an array starting with 'jewelry photos'
tfidf.inverse_transform(X_tfidf[0])
```

Out[261…

```
[array(['jewelry photos', 'features stopwatch', 'special features',
        'model no', 'wide model', '22mm wide', 'long 22mm',
        'bracelet dimensions', 'case thickness', 'case width',
        'resistance 100m', 'water resistance', 'quartz water',
        'movement quartz', 'buckle movement', 'clasp buckle',
        'leather clasp', 'black leather', 'strap black', 'black strap',
        'color black', 'dial color', 'to read', 'easy to', 'an easy',
        'plenty of', 'of the', 'day of', 'every day', 'you can', 'sub',
        'stopwatch', 'special', 'no', 'model', 'wide', '22mm',
        'dimensions', 'bracelet', '5mm', '13', 'thickness', 'width',
        'case', '100m', 'resistance', 'water', 'quartz', 'movement',
        'buckle', 'clasp', 'leather', 'strap', 'black', 'color', 'brand',
        'dial', 'read', 'into', 'plenty', 'watch', 'chronograph',
        'inspired', 'car', 'sports', 'week', 'day', 'every', 'enjoy',
        'can'], dtype='<U24')]
```

In [262…

```
# 6. (3pts) Format Bigrams and Print Sample of Extracted Vocabulary

# The learned vocabulary can be retrieved from tfidf as a list using .get_featur
# Store the extracted vocabulary as vocab
vocab = tfidf.get_feature_names_out()

# Sklearn joins bigrams with a space character.
# To make our output easier to read, replace the spaces in each term in
#    vocab (a list of strings) with an underscore.
# To do this we can use the string .replace() method.
# For example x.replace(' ','_') will replace all ' ' in x with '_'.
# Store the result back into vocab

vocab = [i.replace(' ','_') for i in vocab]

# Print the last 5 terms in the vocab
#  The first term printed should be 'zipper_pocket'
print(vocab[-5:])
```

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

Transform product descriptions into topics and print sample terms from topics

In [263…

```
# 7. (3pts) Perform Topic Modeling with LDA

# Now that we have our vectorized data, we can use Latent Direchlet Allocation t
#   per-document topic distributions and per-topic term distributions.
# Though the documents are likely composed of more, we'll model our dataset usin
#     20 topics for ease of printing.

# Import LatentDirichletAllocation from sklearn
from sklearn.decomposition import LatentDirichletAllocation

# Instantiate a LatentDirichletAllocation model that will
#    produce 20 topics
#    use all available cores to train
#    random_state=512
# Store as lda
lda = LatentDirichletAllocation(n_components=20,n_jobs=-1,random_state=512)

# Run fit_transform on lda using X_tfidf.
# Store the output (the per-document topic distributions) as X_lda
X_lda=lda.fit_transform(X_tfidf)

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

Out[263…

```
(5000, 20)
```

In [264…

```
# 8. (5pts) Get Assigned Topics for Product at df_jcp row 0

# Get the assigned topic proportions for the document at row 0 of X_lda
# This will be a list of 20 floats between 0 and 1
# Round all values to a precision of 2
# Store as theta_0
```

```
theta_0 = X_lda[0].round(2)
print(f'{theta_0 = :}\n')

# LDA will assign a small weight (or proability) to each topic for a document
# How many of the topics in theta_0 have a (relatively) large weight (> .01)?
# Store in n_topics_assigned_0
s = sum(theta_0)
n_topics_assigned_0 = sum(i/s > 0.01 for i in theta_0)
print(f'{n_topics_assigned_0 = :}\n')

# What are the indices of the assigned topics, sorted descending by the values i
#  Use np.argsort() to return the indices sorted by value (ascending)
#  Use [::-1] to reverse the sorting order (from ascending to descending)
#  Return only the first n_assigned_0 indices, those with large probability
#  Store as assigned_topics_0
#  You should see n_topics_assinged_0 indices
assigned_topics_0 = np.argsort(theta_0)[::-1][0:n_topics_assigned_0]
print(f'{assigned_topics_0 = :}\n')

# Now that we have the topic indexes, we need to see what each topic looks like
#   using the per topic word distrutions stored in lda.components_ (next questio
```

```
theta_0 = [0.01 0.74 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
 0.16 0.01 0.01 0.01 0.01 0.01]

n_topics_assigned_0 = 2

assigned_topics_0 = [ 1 14]
```

In [265…

```
# 9. (5pts) Print Top Topic Terms

# To get a sense of what each topic is composed of, we can print the most likely
# We'd like a print statement that looks like this:
#     Topic # 0 : socks spandex fits shoe fits_shoe

# To make indexing easier, first convert vocab from a list to np.array()
# Store back into vocab
vocab = np.array(vocab)

# assert that vocab is the correct datatype
assert type(vocab) is np.ndarray, "vocab needs to be converted to a numpy array"

# For each topic print f'Topic #{topic_idx:2d} : ' followed by the top 5 most li
# Hints:
#   The per topic term distributions are stored in lda.components_
#      which should be a numpy array with shape (20, 5678)
#   Iterate through the rows of lda.components_, one row per topic
#   Use np.argsort() to get the sorted indices of the current row of lda.compone
#      sorted by the values in that row in ascending order
#   Use [::-1] to reverse the order of the sorted indices
#   Use numpy array indexing to get the first 5 index values
#   Use these indices to get the corresponding terms from vocab
#   Join the list of terms with spaces using ' '.join()
#   Each print statement should start with f'Topic #{topic_idx:2d} : '
#      where topic_idx is an integer 0 to 19
# Each line should look similar to the example shown above

# Use as many lines of code as you need
```

```
topic_term = lda.components_
def top_terms(weights):
    return list(vocab[np.argsort(weights)[::-1]][:5])
for topic_idx in range(len(topic_term)):
    print (f'Topic {topic_idx:2d} :', ' '.join(top_terms(topic_term[topic_idx]))
```

```
Topic  0 : upper heel sole synthetic synthetic_upper
Topic  1 : dial case color bracelet strap
Topic  2 : rug resistant yes backing pad
Topic  3 : upper sole rubber rubber_sole construction
Topic  4 : moisture wicking moisture_wicking fabric dri
Topic  5 : big through_seat just_below sits_just big_tall
Topic  6 : what skin what_it oil it_is
Topic  7 : only_imported clean_only only return in_its
Topic  8 : measures safe wipe set glass
Topic  9 : may sterling diamond jewelry_photos gold
Topic 10 : socks nylon support fits fits_shoe
Topic 11 : stainless_steel stainless steel cooking oven
Topic 12 : short short_sleeves tee crewneck shirt
Topic 13 : king comforter set shams pillow
Topic 14 : tone gold_tone silver_tone silver tone_metal
Topic 15 : garment 25½ fitting garment_is loose_fitting
Topic 16 : ci inseam_misses petite misses pop
Topic 17 : inseam waist pants zip leg
Topic 18 : crochet cute bay graphic_print st
Topic 19 : sleeveless line wash_line line_dry dry_imported
```

In [266…

```
# Looking at the description column of row 0, the assigned_topics_0 and
# the top terms per topic above, our LDA model seems to have generated
# topics that make sense given descriptions of department store goods,
# with some a better fit than others.
```

Generate recommendations using topics

In [267…

```
# 10. (3pts) Generate Similarity Matrix

# We'll use Content-Based Filtering to make recommendations based on a query pro
# Each product will be represented by its LDA topic weights learned above (X_lda
# We'd like to recommend similar products in LDA space.
# We'll use cosine distance as our measure of similarity, where lower distance m
#  more similar.
# Note that we're using "distance" where lower is better instead of "similarity"
#  as the default sorting is ascending and it makes indexing easier.

# Import cosine_distances (not cosine_similarity) from sklearn.metrics.pairwise
from sklearn.metrics.pairwise import cosine_distances

# Use cosine_distances to generate similarity scores on our X_lda data
# Store as distances
# NOTE: we only need to pass X_lda in once as an argument,
#   the function will calculate pairwise distance between all rows in that matri
distances=cosine_distances(X_lda)

# print the shape of the distances matrix (should be 5000 x 5000)
distances.shape
```

```
(5000, 5000)
```

Out[267…

In [268…

```
# 11. (4pts) Find Recommended Products

# Let's test our proposed recommendation engine using the product at row 0 in df
#   The name of this product is "Invicta® Sl Rally Mens Black Leather Strap Chro
#   Our system will recommend products similiar to this product.

# Print the names for the top 10 most similar products to this query.
# Suggested way to do this is:
#   get the cosine distances from row 0 of the distances matrix
#   get the indices of this first row of distances sorted by value ascending usi
#   get the first 10 indexes from this sorted array of indices
#   use those indices to index into df_jcp.name_title
#   to get the full string, use .values
#   print the resulting array

# HINT: The first two products will likely be:
#   'Invicta® Sl Rally Mens Black Leather Strap Chronograph Watch 16012',
#   'Timex® Easy Reader Womens White Leather Strap Watch T2H3917R',
distance_0 = distances[0]
indices = np.argsort(distance_0)[:10]
print(df_jcp.name_title[indices].values)
```

```
['Invicta® Sl Rally Mens Black Leather Strap Chronograph Watch 16012'
 'Seiko® Mens Two-Tone Brown Dial Chronograph Watch SSC142'
 'Despicable Me Minions Kids Flashing and Sound Digital Watch'
 'Citizen® Eco-Drive® Womens Crystal-Accent Stainless Steel Watch EX1320-54E'
 'Womens Crystal-Accent White Lizard Faux Leather Cuff Bangle Watch'
 'Star Wars® Stormtrooper Kids Flashing and Sound Digital Watch'
 'Casio® Mens Champagne Dial Black Resin Strap Sport Watch MW600F-9AV'
 'TKO ORLOGI Womens Crystal-Accent Chain-Link Blue Silicone Strap Stretch Watch'
 'Pulsar® Mens Silver-Tone Black Ion Watch PS9273'
 'Armitron® ProSport Womens Digital Sport Chronograph Watch 45/7036PNK']
```

Part 2: Sentiment Analysis Using Pipelines

Here we will train a model to classify positive vs negative sentiment on a set of pet supply product reviews using sklearn Pipelines.

In [269…

```
# 12. (2pts) Load the Data

# The dataset we'll be working with is a set of product reviews
#   of pet supply items on Amazon.
# This data is taken from https://nijianmo.github.io/amazon/index.html
#   "Justifying recommendations using distantly-labeled reviews and fined-graine
#   Jianmo Ni, Jiacheng Li, Julian McAuley
#   Empirical Methods in Natural Language Processing (EMNLP), 2019

# Load product reviews from ../data/amazon-petsupply-reviews_subset.csv.zips
# Use pandas read_csv function with the default parameters as in part 1.
# Store the resulting dataframe as df_amzn.
df_amzn = pd.read_csv('../data/amazon-petsupply-reviews_subset.csv.zip',compress

# print a summary of df_amzn using .info()
# there should be 10000 rows with 2 columns
```

```
df_amzn.info()

# print blank line
print()

# print the review in the first row of the dataframe as an example
print(df_amzn['review'].iloc[0])

# print the rating in the first row of the dataframe as an example
print(df_amzn['rating'].iloc[0])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   review  10000 non-null  object
 1   rating  10000 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 156.4+ KB

My cats are considerably more happy with this toy...and I don't have to leave th
e sofa to use it, given the long wand length. yay laziness!!
5
```

In [270…

```
# 13. (2pts) Transform Target

# The ratings are originally in a 5 point scale
# We'll turn this into a binary classification task to approximate positive vs n

# Print the proportions of values seen in the rating column
#  using value_counts() with normalize=True
#  round to a precision of 2
print(df_amzn['rating'].value_counts(normalize=True).round(2))

# Create a new binary target by setting
#  rows where rating is 5 to True
#  rows where rating is not 5 to False
# Store in y
y = (df_amzn['rating']==5).astype(bool)

# print a blank line
print()

# Print the proportions of values seen in y
#  using value_counts() with normalize=True
#  round to a precision of 2
# True here means a rating of 5 (eg positive)
# False means a rating less than 5 (eg negative)
y.value_counts(normalize=True).round(2)
```

```
5    0.66
4    0.14
3    0.09
1    0.06
2    0.05
Name: rating, dtype: float64
```

Out[270…

```
True     0.66
```

```
False    0.34
Name: rating, dtype: float64
```

In [271…

```
# 14. (2pts) Train-test split

# Import train_test_split from sklearn
from sklearn.model_selection import train_test_split

# Split df_amzn.review and y into a train and test set
#   using train_test_split
#   stratifying by y
#   with test_size = .2
#   and random_state = 512
# Store as reviews_train,reviews_test,y_train,y_test
reviews_train,reviews_test,y_train,y_test = train_test_split(df_amzn.review,
                                                              y,
                                                              test_size=0.2,
                                                              stratify=y,
                                                              random_state=512)


# print the proportion of values seen in y_train
#  round to a precision of 2
# visually compare this to the proportion of values seen in y
#  to confirm that the class distributions are the same
y_train.value_counts(normalize=True).round(2)
```

Out[271…

```
True     0.66
False    0.34
Name: rating, dtype: float64
```

In [272…

```
# 15. (4pts) Create a Pipeline of TfIdf transformation and Classification

# import Pipeline and GradientBoostingClassifier from sklearn
from sklearn.pipeline import Pipeline
from sklearn.ensemble import GradientBoostingClassifier

# Create a pipeline with two steps:
#  TfIdfVectorizer with min_df=5 and max_df=.5 named 'tfidf'
#  GradientBoostingClassifier with 20 trees named 'gbc'
# Store as pipe_gbc
pipe_gbc = Pipeline([('tfidf',TfidfVectorizer(min_df=5, max_df=.5)),
                     ('gbc',GradientBoostingClassifier(n_estimators=20))

])
# Print out the pipeline
# You should see both steps: tfidf and gbc
print(pipe_gbc)
```

```
Pipeline(steps=[('tfidf', TfidfVectorizer(max_df=0.5, min_df=5)),
                ('gbc', GradientBoostingClassifier(n_estimators=20))])
```

In [273…

```
# 16. (5pts) Perform Grid Search on pipe_gbc

# import GridSearchCV from sklearn
from sklearn.model_selection import GridSearchCV

# Create a parameter grid to test using:
#   unigrams or unigrams + bigrams in the tfidf step
```

```
#   max_depth of 2 or 10 in the gbc step
# Store as param_grid
param_grid = {'tfidf__ngram_range':[[1,1],[1,2]],
               'gbc__max_depth':[2,10]}

# Instantiate GridSearchCV to evaluate pipe_gbc on the values in param_grid
#   use cv=2 and n_jobs=-1 to reduce run time
# Fit on the training set of reviews_train,y_train
# Store as gs_pipe_gbc
gs_pipe_gbc = GridSearchCV(pipe_gbc, param_grid, cv=2, n_jobs=-1).fit(reviews_tr

# Print the best parameter settings in gs_pipe_gbc found by grid search
print(gs_pipe_gbc.best_params_)
# Print the best cv score found by grid search, with a precision of 2
print(gs_pipe_gbc.best_score_.round(2))
```

```
{'gbc__max_depth': 10, 'tfidf__ngram_range': [1, 2]}
0.74
```

In [274…

```
# 17. (1 pts) Evaluate on the test set

# Calculate the test set (reviews_test,y_test) score using the trained gs_pipe_g
#   to give confidence that we have not overfit
#   while still improving over a random baseline classifier
# Print the accuracy score on the test set with a precision of 2
print(gs_pipe_gbc.score(reviews_test,y_test).round(2))
```

```
0.75
```

In [275…

```
# 18. (1 pts) Evaluate on example reviews

# Generate predictions for these two sentences using the fit gs_pipe_gbc:
#   'This is a great product.'
#   'This product is not great.'
# You should see True for the first (rating of 5)
#   and False for the second (rating of less than 5)
print(gs_pipe_gbc.predict({'This is a great product.'}))
print(gs_pipe_gbc.predict({'This product is not great.'}))
```

```
[ True]
[False]
```