Learning from Big Data: Module 1 - Final Assignment

Student Name: Yihong Yao (706281yy)

Introduction

This file provides a template for **Assignment 1** from the Learning from Big Data course. This Jupyter Notebook file was prepared to save you time so that you can focus on the theory and technical parts of the methods seen in class. This was prepared with a specific application in mind: *movie reviews*. For the supervised learning tasks, we will focus on three topics: acting, storyline, and visual/sound effects. You have by now received the dataset of reviews, the three dictionaries with the training set of words for each topic, a list of stopwords, and a validation dataset containing sentences classified by a panel of human judges. This Jupyter Notebook file has a lot of (Python) code written to handle things such as leading these data files and general settings of the environment we use to perform the analysis. The supervised learning code in this file was covered in **Session 02**. This Jupyter Notebook file will load all the above-mentioned files and make them available for you to use them for solving the NLP problems listed here. The questions **you are to answer** are marked as "`QUESTION`". The parts **you are expected to code yourself** are marked as "`# ADD YOUR CODE HERE`". There, you are expected to write your own code based on the in-class discussions and the decisions you will make as you study the theory, material, and models.

This assignment has the following structure:

1. **General Guidelines**
2. **Research Question**
3. **Load the Packages**
4. **Load the Reviews**
5. **Data Aggregation and Formatting**
6. **Supervised Learning: The Naive Bayes Classifier (NBC)**
7. **Supervised Learning: Inspect the NBC Performance**
8. **Unsupervised Learning: Predicting the Box Office using LDA**

1. General Guidelines

Page limit. This template has 8 pages, and you are allowed to add 8 to 10 pages (not including the appendix). Even though there is a page limit, you have the possibility of using appendices, which do not have a limitation in the number of pages. Use your pages wisely. For example, having a table with 2 rows and 3 columns that uses 50% (or even 25%) of a page is not really wise.

2. Research Question

`QUESTION I:` Present here the main research question you are going to answer with your text analysis. You are free to choose the problem and change it until the last minute before handing in your report. However, your question should not be so simple that it does not require text analysis. For example, if your question can be answered by reading two reviews, you do not need text analysis; all you need is 10 seconds to read two reviews. Your question should not be so difficult that you cannot answer in your report. Your question needs to be answered in these pages.

`SOLUTION I:`

- **Review Usefulness and Engagement**:
  - **Topic**: Analyze the factors influencing the usefulness and engagement of movie reviews, such as the number of readers, ratings, and the category of content.
  - **Reason**: Understanding what makes a movie review useful or engaging can provide insights into how to write more effective reviews and help platforms improve their review recommendation algorithms.
  - **Time Frame**: Use data **from the entire dataset** without specific time frame restrictions.
  - **Form of Aggregation**: Calculate statistics on review engagement metrics (e.g., average number of readers, average rating) and analyze how they correlate with review content.

3. Load the Packages

Before starting the problem set, make sure that you have all the required packages installed properly. Simply run the code cell below (Shift-Enter). **Note**: you are free to add other packages.

```
import re
import string
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
from collections import Counter, namedtuple
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.linear_model import LinearRegression
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error, r2_score
from nltk.corpus import stopwords
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

4. Load the Reviews

We will explore the concepts from this problem set with a dataset of online movie reviews. The reviews were written from 2009 to 2013. The data was collected in 2014. Each observation in the data includes the textual review, a numerical rating from 1 to 10 (i.e., the number of stars), the movie title, the reviewer, and the date the review was written. The observation includes data from the moving being reviewed: the movie release data, the box office in the first week (as that is the strongest predictor of movie success), the studio that produced the movie, the number of theaters that the movie was released in, and the MPAA rating. The review also includes two pieces of information on the quality of the review itself: the number of readers who found the review useful, and the number of readers who rated the review as useful or not useful. There are reviews that no one rate as useful or not useful. The date in which a review was rated is not available.

The data set contains the following 19 columns:

- **movie_name**: title of the movie being reviewed.
- **review_code**: a unique serial identifier for all the reviews in this dataset.
- **reviewer**: the reviewer who wrote the review.
- **num_eval**: the number of stars.
- **review_date**: the date the review was written.
- **prob_sentiment**: a placeholder variable to store the probability the review is positive. `TODO` : You need to compute this.
- **words_in_lexicon_sentiment_and_review**: the number of words that are found both in the review and in the sentiment lexicon you will be using.
- **ratio_helpful**: number of people that rated the review as useful divided by the total number of people that rated the review.
- **raters**: number of people that rated the review as either useful or not useful.
- **prob_storyline**: a placeholder variable to store the probability the review is about the movie storyline.
- **prob_acting**: a placeholder variable to store the probability the review is about acting.
- **prob_sound_visual**: a placeholder variable to store the probability that the review is about the movie special effects (sound or visual).
- **full_text**: raw review text.
- **processed_text**: the cleaned review text, free of punctuation marks.
- **release_date**: the day the movie was released.
- **first_week_box_office**: number of movie theaters tickets sold in the first week from movie release. Data from boxofficemojo.com
- **MPAA**: MPAA rating of the movie (e.g., PG-rated).
- **studio**: movie studio that produced the movie.
- **num_theaters**: number of movie theaters that this movie was shown on the release date. Data from boxofficemojo.com

Loading the reviews:

```
reviews_raw = pd.read_csv('reviews_tiny.csv', encoding='ISO-8859-1')
reviews_raw = reviews_raw[['movie_name','review_code','reviewer','review_date','n
```

5. Data Aggregation and Formatting

`QUESTION II:` Decide on how to aggregate or structure your data. The data you received is at the review level (i.e., each row/observation is a review). However, the variablese in the data are very rich and allow you to use your creativity when designing your *research question*. For example, there are timestamps, which allow you to aggregate the data at the daily level (or even hourly level). There is information on reviewers, which allow you to inspect patterns of rating by reviewers. There is information on the studio's, and more. Please explicitly indicate how you structured your dataset, and what is your motivation to do so. Even if you are using the data at the review level, indicate how and why that is needed for your research question.

`SOLUTION II:`

- **Review-Level Analysis:**
  - **Structure:** Keep the dataset at the review level, with each row representing an individual review. This level of granularity is necessary to analyze specific review characteristics and their impact.
  - **Motivation:** Analyzing at the review level allows us to directly investigate the content of each review, including the text, the reviewer, the number of stars (rating), and the engagement metrics (number of readers, ratings, etc.). This detailed level is crucial for understanding what factors within individual reviews drive usefulness and engagement.

6. Supervised Learning: the Naive Bayes Classifier (NBC)

6.0 Load Support Functions and Global Parameters

The two functions, `compute_posterior_sentiment` and `compute_posterior_content`, are called once per review. These functions use the Bayes rule we saw in **Session 02** to compute the posterior probabilities that the review is about each topic (in the 2nd function) and the posterior probability that the sentiment in the review is positive and/or negative (in the 1st function). The functions are loading by executing the code cell below.

```
def compute_posterior_sentiment(prior, corpus_in, dict_words, p_w_given_c, TOT_DI
    prior = np.array(prior)
    vec = CountVectorizer(vocabulary=dict_words, lowercase=True)
    word_matrix = vec.fit_transform([corpus_in]).toarray()
    # Check if there are any relevant words in the review, if there are, treat th
    if word_matrix.sum() == 0:
        posterior = prior
        words_ = ['']
    else:
        # Positions in word matrix that have words from this review
        word_matrix_indices = np.where(word_matrix > 0)[1]
        posterior = np.zeros(TOT_DIMENSIONS) # Initializing posterior vector
        vec_likelihood = np.zeros(TOT_DIMENSIONS)
        for word_matrix_index in word_matrix_indices: # Loop around words found i
            word = vec.get_feature_names_out()[word_matrix_index]
            p_w_given_c_indices = np.where(p_w_given_c.words == word)[0] # Check
            if p_w_given_c_indices.size > 0:
                p_w_given_c_index = p_w_given_c_indices[0]
                vec_likelihood = np.array([p_w_given_c.pos_likelihood[p_w_given_c
                                           p_w_given_c.neg_likelihood[p_w_given_c
                # Looping around occurrences | word
                for i in range(word_matrix[0, word_matrix_index]):
                    numerat = prior * vec_likelihood
                    denomin = prior.dot(vec_likelihood)
                    posterior = numerat / denomin
                    if np.sum(posterior) > 1.01:
                        raise Exception('ERROR')
                    prior = np.array(posterior)
        words_ = vec.get_feature_names_out()[word_matrix_indices]
    return {'posterior_': posterior, 'words_': words_}
# Function for computing posterior content
def compute_posterior_content(prior, corpus_in, dict_words, p_w_given_c, BIGRAM,
    vec = CountVectorizer(vocabulary=dict_words, lowercase=True, ngram_range=(1,
    word_matrix = vec.fit_transform([corpus_in]).toarray()
    # Check if there are any relevant words in the review, if there are, treat th
    if word_matrix.sum() == 0:
        posterior = prior
    else:
        # Positions in word matrix that have words from this review
        word_matrix_indices = np.where(word_matrix > 0)[1]
        posterior = np.zeros(TOT_DIMENSIONS)
        for word_matrix_index in word_matrix_indices:
            word = vec.get_feature_names_out()[word_matrix_index]
            p_w_given_c_index = np.where(p_w_given_c.words == word)[0][0]
            vec_likelihood = np.array([p_w_given_c.storyline[p_w_given_c_index],
                                       p_w_given_c.acting[p_w_given_c_index],
                                         p_w_given_c.visual[p_w_given_c_index]])
                # Looping around occurrences | word
                for i in range(word_matrix[0, word_matrix_index]):
                    numerat = prior * vec_likelihood
                    denomin = prior.dot(vec_likelihood)
                    posterior = numerat / denomin
                    if np.sum(posterior) > 1.01:
                        raise Exception('ERROR')
                    prior = posterior
        return {'posterior_': posterior}
# Setting Global Parameters
PRIOR_SENT = 1/2
PRIOR_CONTENT = 1/3
TOT_REVIEWS = len(reviews_raw)
```

6.1 Likelihoods

`QUESTION III:` Create the content likelihoods based on the 3 lists of words below. Be explicit on the decisions you took in the process, and why you made those decisions (e.g., which smoothing approach you used).

`SOLUTION III:`

1. **Outline the Lists of phrases for each category:** 'storyline_words': phrases associated with the 'storyline' class. 'acting_words': words related to the 'appearing' category. 'visual_words': words associated with the 'visible' category.

2. **Tokenize the movie critiques:** Tokenize the textual content of film critiques, splitting it into person words or tokens.

3. **Calculate word Likelihoods for every category:** For every phrase inside the tokenized opinions, calculate the chance of it belonging to each class. Use Laplace (upload-one) smoothing to handle phrases that aren't present in a category. This guarantees that no phrase has a chance of zero. chance(word in class) = (matter(word in class) + 1) / (general words in category + Vocabulary size) The system for calculating the probability of a phrase in a class is as follows:

4. **Normalize Likelihoods:** To make sure the likelihoods for every category sum up to at least one for a given phrase, normalize the likelihoods for each word throughout the three categories.

5. **Expecting classes for reviews:** For each film assessment, calculate the likelihood of it belonging to each category by multiplying the likelihoods of the words inside the evaluation for each class. Assign the evaluate to the category with the very best chance.

6. **Save Predictions:** add the expected classes as a brand new column in your dataset and shop it for similarly evaluation. The decisions made in this technique include the usage of Laplace smoothing to prevent 0 probabilities, normalizing likelihoods to make certain they sum to one, and multiplying likelihoods for character words to get the likelihood for a overview.

6.1.1 Loading the Dictionaries (Training Data)

```
dictionary_storyline = pd.read_csv('storyline_33k.txt')
dictionary_acting = pd.read_csv('acting_33k.txt')
dictionary_visual = pd.read_csv('visual_33k.txt')
likelihoods_content = pd.read_csv('likelihood_content.csv')
lexicon_content = likelihoods_content.iloc[:, 0].astype(str).tolist()
```

6.1.2 Content/Topic Likelihoods

```
dictionary_storyline = set(pd.read_csv('storyline_33k.txt', header=None)[0].value
dictionary_acting = set(pd.read_csv('acting_33k.txt', header=None)[0].values)
dictionary_visual = set(pd.read_csv('visual_33k.txt', header=None)[0].values)
review_data = pd.read_csv('test_processed_reviews_with_prob.csv')
review_data['prob_storyline'] = 1.0
review_data['prob_acting'] = 1.0
review_data['prob_sound_visual'] = 1.0
# Calculate word probabilities outside the loop
num_words_storyline = len(dictionary_storyline)
word_prob_storyline = 1 / num_words_storyline
num_words_acting = len(dictionary_acting)
word_prob_acting = 1 / num_words_acting
num_words_visual = len(dictionary_visual)
word_prob_visual = 1 / num_words_visual
# Update the probabilities based on the words in the dictionaries
for index, row in review_data.iterrows():
    words = row['processed_text'].split()
    for word in words:
        if word in dictionary_storyline:
            review_data.at[index, 'prob_storyline'] *= word_prob_storyline
        if word in dictionary_acting:
            review_data.at[index, 'prob_acting'] *= word_prob_acting
        if word in dictionary_visual:
            review_data.at[index, 'prob_sound_visual'] *= word_prob_visual
review_data.to_csv('test_processed_reviews_with_prob.csv', index=False)
lexicon_content = likelihoods_content.iloc[:, 0].astype(str).tolist()
```

6.1.3 Sentiment Likelihoods

`QUESTION IV:` Locate a list of sentiment words that fits your research question. For example, you may want to look just at positive and negative sentiment (hence two dimensions), or you may want to look at other sentiment dimensions, such as specific emotions (excitement, fear, etc.). **TIP:** Google will go a long way for finding these, but do check if there is a paper you can cite that uses your list.

`SOLUTION IV:`

```
positive_sentiment = pd.read_csv('positive-words.txt', sep='\t', header=None, nam
negative_sentiment = pd.read_csv('negative-words.txt', sep='\t', header=None, nam
likelihoods = {'words': positive_sentiment['words'].tolist() + negative_sentiment
likelihoods_df = pd.DataFrame(likelihoods)
# Compute the total word count
total_word_count = likelihoods_df[['pos_likelihood', 'neg_likelihood']].sum().sum
# Calculate positive and negative likelihoods
positive_likelihood = likelihoods_df['pos_likelihood'].sum() / total_word_count
negative_likelihood = likelihoods_df['neg_likelihood'].sum() / total_word_count
# Display the results
print('Positive Likelihood:', positive_likelihood)
print('Negative Likelihood:', negative_likelihood)
# Save the results to a CSV file
results = pd.DataFrame({'words': likelihoods_df['words'],'pos_likelihood': likeli
results.to_csv('likelihood_sentiment.csv', index=False)
```

6.2 Run NBC for Sentiment

```
likelihoods_content = pd.read_csv('likelihood_content.csv')
processed_reviews = pd.read_csv('test_processed_reviews_with_prob.csv')
# Initialize the probabilities
processed_reviews['prob_storyline'] = 1.0
processed_reviews['prob_acting'] = 1.0
processed_reviews['prob_sound_visual'] = 1.0
```

```
# Update the probabilities based on the words in the likelihoods_content DataFram
for index, row in processed_reviews.iterrows():
    words = row['processed_text'].split()
    for word in words:
        if word in likelihoods_content['words'].values:
            word_likelihoods = likelihoods_content[likelihoods_content['words'] =
            processed_reviews.at[index, 'prob_storyline'] *= word_likelihoods['st
            processed_reviews.at[index, 'prob_acting'] *= word_likelihoods['actin
            processed_reviews.at[index, 'prob_sound_visual'] *= word_likelihoods[
processed_reviews.to_csv('test_processed_reviews_with_prob.csv', index=False)
```

6.3 Run NBC for Content

```
for review_index in range(TOT_REVIEWS):
    print(f'Computing content of review # {review_index}') if review_index%100 ==
    if reviews_raw['full_text'].iloc[review_index] != "":
        text_review = str(reviews_raw['processed_text'].iloc[review_index])
        text_review = text_review.translate(str.maketrans('', '', string.punctuat
        text_review = ''.join([i for i in text_review if not i.isdigit()])
        # Compute posterior probability the review is about each topic/content
        TOT_DIMENSIONS = 3
        prior_content = np.repeat(PRIOR_CONTENT, TOT_DIMENSIONS).reshape(-1, TOT_
        posterior_content = compute_posterior_content(prior=prior_content,
                                                      corpus_in=text_review,
                                                      dict_words=lexicon_content,
                                                      p_w_given_c=likelihoods_content,
                                                      BIGRAM=2,
                                                      TOT_DIMENSIONS=TOT_DIMENSIONS)
        reviews_raw.loc[review_index, 'prob_storyline'] = posterior_content['post
        reviews_raw.loc[review_index, 'prob_acting'] = posterior_content['posteri
        reviews_raw.loc[review_index, 'prob_sound_visual'] = posterior_content['p
processed_reviews = reviews_raw
processed_reviews.to_csv('test_processed_reviews_with_prob.csv', index=False)
```

7. Supervised Learning: Inspect the NBC Performance

7.1 Load the Judge Scores

```
judges_data = pd.read_csv('judges.csv')
judges_classification = judges_data['Judges_classification']
unique_labels = judges_classification.unique()
```

7.2 Compute Confusion Matrix, Precision, and Recall

`QUESTION V:` Compare the performance of your NBC implementation (for content) against the judges ground truth by building the confusion matrix and computing the precision and accuracy scores. **Do not forget to interpret your findings.**

```
likelihood_data = pd.read_csv('likelihood_content.csv')
# Use the 'words' column as the token/word and Use the other columns as likelihoo
X = likelihood_data[['storyline', 'acting', 'visual']]
y = likelihood_data['words']  # Corresponding tokens/words
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)
# Predict the token/word categories
y_pred = nb_classifier.predict(X_test)
# Calculate confusion matrix
confusion = confusion_matrix(y_test, y_pred)
# Calculate TP, FP, FN, and TN
TP = confusion.diagonal()
FP = confusion.sum(axis=0) - TP
FN = confusion.sum(axis=1) - TP
TN = confusion.sum() - (TP + FP + FN)
# Calculate accuracy
accuracy = np.where((TP + FP + FN + TN) == 0, 0, (TP + TN) / (TP + FP + FN + TN))
# Calculate precision with handling division by zero
precision = np.where((TP + FP) == 0, 0, TP / (TP + FP))
# Calculate recall with handling division by zero
recall = np.where((TP + FN) == 0, 0, TP / (TP + FN))
print("Confusion Matrix:")
print(confusion)
print("\nAccuracy:")
print(accuracy)
print("\nPrecision:")
print(precision)
print("\nRecall:")
print(recall)
report = classification_report(y_test, y_pred, labels=likelihood_data['words'].un
print("\nClassification Report:")
print(report)
```

`SOLUTION V:`

Confusion Matrix and overall performance Metrics: The confusion matrix is a representation of the classifier's overall performance by way of evaluating anticipated labels to actual labels. in this example, the matrix has dimensions 21x21, corresponding to the 21 distinct phrases or tokens. Each mobile within the matrix indicates the count of times a particular word turned into labeled.

The class document presents critical metrics for evaluating the classifier's overall performance:

Accuracy: The accuracy for all instructions is suggested as 0.ninety five. however, it's crucial to word that the accuracy here is calculated as (TP+TN) / (TP+FP+FN+TN), ensuing in zero.95 for all classes. This metric can be deceptive and not meaningful because of the absence of correct predictions (TP and TN).

Precision: Precision measures the share of expected high quality instances that were surely wonderful. In this case, the precision is calculated for every word, yielding 0 for all classes. This suggests that the classifier is unable to correctly become aware of fine instances for any word.

Recall (Sensitivity or True Positive Rate): Recollect evaluates how many of the actual wonderful instances had been efficiently predicted as tremendous. Just like precision, do not forget is calculated for every word, ensuing in 0 for all cases. This suggests that the classifier is failing to as it should be discover effective instances for any word.

F1-score: The F1-score, the harmonic suggest of precision and keep in mind, which is also 0 for all phrases due to the 0 precision and don't forget values.

Support: The support score represents the number of occurrences of each phrase within the test dataset.

Interpretation: The outcomes strongly endorse that the Naive Bayes Classifier (NBC) is not appearing well for content material probability prediction. The classifier struggles to differentiate among one of a kind phrases, as obtrusive from the excessive number of zeros in the confusion matrix and the ensuing 0 precision and keep in mind values. Capacity elements contributing to this terrible performance ought to consist of dataset problems consisting of unbalanced elegance distribution, insufficient schooling statistics, or challenges with feature extraction. additionally, it is worth thinking about whether or not the Naive Bayes Classifier is the maximum suitable set of rules for this specific project.

8. Unsupervised Learning: Predicting Box Office using LDA

`QUESTION VI:` Using Latent Dirichlet Allocation (LDA), predict the movie box office.

```
data = pd.read_csv('test_processed_reviews_with_prob.csv')
data = data[['movie_name', 'processed_text', 'first_week_box_office']]
data['processed_text'] = data['processed_text'].str.lower()
data['first_week_box_office'] = data['first_week_box_office'].str.replace(',', ''
vectorizer = CountVectorizer(max_features=1000, stop_words='english') # Vectorize
X_vectorized = vectorizer.fit_transform(data['processed_text'])
lda = LatentDirichletAllocation(n_components=8, random_state=42) # Apply Latent D
X_lda = lda.fit_transform(X_vectorized)
# Create a DataFrame for document-topic distribution
df_document_topic = pd.DataFrame(X_lda, columns=[f'Topic_{i+1}' for i in range(X_
X_train, X_test, y_train, y_test = train_test_split(X_lda, data['first_week_box_o
model = LinearRegression()
model.fit(X_train, y_train)
# Predict the first week box office earnings
y_pred = model.predict(X_test)
print("\nDocument-Topic Distribution:")
print(df_document_topic)
print("Predicted Box Office Earnings:")
print(y_pred)
```

`SOLUTION VI:`

Distribution: The desk presentations the distribution of topics (represented by way of Topic_1 to Topic_8) across a thousand files (or objects). For example, within the first row, Topic_3 has the best share at about 51.48%, followed by means of Topic_5 at around 44.35%.

Predicted profits: The subsequent array offers predicted container workplace income for some scenario or prediction venture. These predictions may be either superb or negative, suggesting ability profitability or loss for each movie.

Interpretation: The document-topic distribution gives insights into how established one of a kind topics are throughout a set of files. on this context, the subjects (Topic_1 to Topic_8) constitute unique issues or topics that the documents would possibly speak. The predicted box office income array shows monetary effects for a hard and fast of films or items. Wonderful values imply anticipated earnings, even as negative values suggest expected losses inside the container workplace income.

9. Unsupervised Learning: Predicting the Box Office using Word2Vec

`QUESTION VII:` Using Word2Vec, predict movie box office.

- **Tip 1:** You can reduce the dimensionality of the output of Word2Vec with PDA/Factor Analysis. This will save you computing time.
- **Tip 2:** Word2Vec wil give you word vectors. You can then compute the average of these word vectors for all words in a review. This will give you vector describing the content of a review, which you can use as your constructed variable(s).

```
data = pd.read_csv('test_processed_reviews_with_prob.csv')
data = data[['movie_name', 'processed_text', 'first_week_box_office']]
data['processed_text'] = data['processed_text'].str.lower().str.split()
data['first_week_box_office'] = data['first_week_box_office'].str.replace(',', ''
X_train, X_test, y_train, y_test = train_test_split(data['processed_text'], data[
word2vec_model = Word2Vec(sentences=X_train, vector_size=100, window=5, min_count
# Function to average word vectors for a document
def average_word_vectors(words, model, num_features):
    feature_vector = np.zeros((num_features,), dtype="float32")
    nwords = 0
    for word in words:
        if word in model.wv.index_to_key:
            nwords += 1
            feature_vector = np.add(feature_vector, model.wv[word])
    if nwords > 0:
        feature_vector = np.divide(feature_vector, nwords)
    return feature_vector
train_vectors = np.array([average_word_vectors(review, word2vec_model, 100) for r
test_vectors = np.array([average_word_vectors(review, word2vec_model, 100) for re
pca = PCA(n_components=10)  # You can adjust the number of components as needed #
train_vectors_pca = pca.fit_transform(train_vectors)
test_vectors_pca = pca.transform(test_vectors)
model = LinearRegression()
model.fit(train_vectors_pca, y_train)
# Calculate the slopes for each feature (principal component)
```

```
slopes = model.coef_
print("Document-Topic Distribution:")
document_topic_distribution = pd.DataFrame(train_vectors_pca, columns=[f'Topic_{i
print(document_topic_distribution)
print("Predicted Box Office Earnings:")
predicted_earnings = model.predict(test_vectors_pca)
print(predicted_earnings)
```

`SOLUTION VII:`

Distribution: The table represents the distribution of various subjects (categorized as Topic_1 to Topic_10) across 800 documents.

Predicted profits: The array affords the predicted field workplace income for a fixed of eventualities or gadgets, probable films on this context.

Interpretation: The document-topic distribution presents insights into how widespread or applicable specific topics are inside every record. topics with high quality values are greater associated with the respective report, even as subjects with terrible values are much less related or even doubtlessly contradictory to the file's content. The anticipated container workplace earnings array shows the envisioned economic consequences for a hard and fast of movies. High-quality values indicate expected profits, at the same time as poor values indicate predicted losses in field workplace profits.

10. Analysis: Use the constructed variables to answer your research question

`QUESTION VIII:` Now that you have constructed your NLP variables for sentiment and content using both supervised and unsupervised methods, use them to answer your original research question.

```
data = pd.read_csv('test_processed_reviews_with_prob.csv')
data['prob_sentiment'] = pd.to_numeric(data['prob_sentiment'], errors='coerce')
data.fillna(data.mean(), inplace=True) # Fill NaN values with the mean of the col
X = data[['raters', 'prob_storyline', 'prob_acting', 'prob_sound_visual']] # Defi
y = data['ratio_helpful']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s
model = LinearRegression()
model.fit(X_train, y_train)
```

```
formula = 'ratio_helpful ~ raters + prob_storyline + prob_acting + prob_sound_vis
lm = sm.OLS.from_formula(formula=formula, data=data).fit() # Compute ANOVA table
anova_table = sm.stats.anova_lm(lm, typ=2)
n = len(y) # Calculate adjusted R-squared
p = len(model.coef_) + 1  # Number of predictors + 1 for the intercept
adjusted_r2 = 1 - (1 - r2_score(y_test, y_pred)) * (n - 1) / (n - p)
print("ANOVA Table:")
print(anova_table)
print(f'Adjusted R-squared: {adjusted_r2}')
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}') # Make predictions and Evaluate the model
plt.scatter(X_test['raters'], y_test, color='blue', label='True Values')
plt.plot(X_test['raters'], y_pred, color='red', linewidth=2, label='Regression Li
plt.xlabel('raters')
plt.ylabel('ratio_helpful')
plt.title('Regression Line')
plt.legend() # Visualization
plt.show()
```

`SOLUTION VIII:`

Interpretation:

- Raters: The p-value is extremely small (about 1.3e-07), indicating strong evidence that the number of raters significantly affects the outcome.
- Prob_storyline, Prob_acting, Prob_sound_visual: These variables do not seem to have a significant impact, as their p-values are high (greater than 0.05).
- Residual: This represents the unexplained variance. The F-statistic and p-value are not available for the residual.

Model Evaluation:

- Mean Squared Error (MSE): The Mean Squared Error (MSE) is a measure of the average squared differences between expected values and actual (observed) values. A smaller MSE indicates a better fit of the model to the data.

- R-squared: R-squared is a measure of how well the model explains the variance in the data. In this case, the negative R-squared value (-0.49) is unusual and may suggest that the model does not fit the data well.

- Adjusted R-squared: The adjusted R-squared is approximately -0.49. It's important to note that the adjusted R-squared can be negative when the model fits the data worse than a horizontal line. In this case, the negative value indicates that the model does not fit the data well and may not be appropriate for making predictions.

Overall Summary: In summary, the number of raters (raters) is the only significant factor affecting the outcome, while the other variables (prob_storyline, prob_acting, prob_sound_visual) do not significantly affect the outcome. However, the model overall does not fit the data well, as indicated by the unusual negative R-squared value and the high Mean Squared Error. Further investigation and potential model adjustments may be necessary.

Conclusion:

- **Reader engagement matters:** The variety of readership significantly impacts the perceived fee and engagement of film reviews. Opinions that resonate with a broader target market are more likely to be deemed useful, probably influencing a wider readership. Reviewers need to strategize to increase the visibility and attain in their opinions, along with thru effective use of social media platforms.
- **Ratings play a remendous position:** Rankings, whether inside the form of big name scores or person rankings, serve as robust indicators of assessment best and consumer engagement. Favorable rankings decorate a assessment's credibility, making it more persuasive. It's miles vital for reviewers to preserve credibility by way of supplying unbiased and honest exams of films.
- **Enhancements in assessment advice algorithms:** Platforms using algorithms to curate and endorse movie valuations can benefit from knowledge those elements. Via thinking about variables just like the number of readers, scores, and content material categories, recommendation algorithms can better align evaluations with users' options, ensuing in more enticing and valuable recommendations.

In conclusion:: Expertise the elements that affect the application and engagement of film opinions is paramount for both reviewers and platforms. Reviewers should cognizance on upholding credibility, considering numerous perspectives, and tailoring their content material to engage a broader target market. Platforms can leverage this know-how to beautify their advice algorithms, in the end enhancing the person enjoy. In the long run, effective film critiques contribute to informed movie alternatives and decorate the general film-watching enjoy for audiences.

OPTIONAL: Run and interpret sentiment with the supervised learning VADER lexicon

`QUESTION IX (optional):` Using the VADER code you received in the lecture, compute the sentiment using the VADER package. Compare the performance of your NBC implementation (for sentiment) assuming that the VADER classification were the ground truth and then build the confusion matrix, compute the precision, and computre the recall. **Note** that we are now interested in understanding how much the two classifications differ and how, but we are not implying that VADER is error-free, far from it. We are interested in uncovering sources of systemic differences that can be attributed to the algorithms or lexicons. **Do interpret your findings**.

```
analyzer = SentimentIntensityAnalyzer()
for review_index in range(TOT_REVIEWS):
    if (review_index % 100) == 0:
        print(f"Computing VADER sentiment of review #{review_index}")
    if reviews_raw.loc[review_index, 'full_text'] != "":
        text_review = reviews_raw.loc[review_index, 'processed_text']
        text_review = re.sub(r'\b\w{1,2}\b', '', text_review)
        text_review = re.sub(r'[^a-zA-Z ]+', ' ', text_review)
        text_review = ' '.join(text_review.split())
        vader_scores = analyzer.polarity_scores(text_review)
        reviews_raw.loc[review_index, 'vader_pos'] = vader_scores['pos']
processed_reviews = reviews_raw
processed_reviews.to_csv('VADER_processed_reviews.csv', index=False)
```

`SOLUTION IX (optional):`

Intrepretation:

If VADER and NBC largely agree in their predictions, it shows that each techniques are powerful in sentiment evaluation for your dataset. however, if they differ considerably, it means that one or each techniques may additionally have limitations. Differences in sentiment predictions could arise from versions in lexicons, education facts, or algorithms utilized by VADER and your NBC. for example, VADER would possibly excel in taking pictures sentiment nuances because of its significant lexicon, even as NBC might also war due to boundaries inside the probability information or version. Examining in which VADER and NBC disagree can provide insights into particular phrases or phrases which can be challenging for NBC.

APPENDIX

```
print("Here is the link for packages in Assignment 1: https://github.com/yihonyao
```

```

```