Sentiment Analysis-Based Product Recommender System

Help of Natural Language Processing

CB SUMANTH
[Pick the date]

List of Context

Table of Contents

Sentiment Analysis – Based On Product Recommender System

Abstract:

This project aims to develop a sentiment analysis-based product recommender system that enhances the traditional recommendation process by incorporating the sentiments expressed in user reviews. By leveraging Natural Language Processing (NLP) techniques, the system classifies customer reviews into positive, negative, or neutral sentiments. These sentiment scores are then integrated into the recommendation algorithm to provide more personalized and accurate product suggestions. The project demonstrates the potential of sentiment analysis in improving the relevance of recommendations, thereby increasing user satisfaction and boosting sales for e-commerce platforms.

Problem Statement:

In the e-commerce industry, providing personalized and accurate product recommendations is crucial for enhancing customer satisfaction and increasing sales. Traditional recommendation systems often rely on user behavior and purchase history, but they may overlook the nuanced insights provided by customer reviews.

Data Extraction :

Data Collection: Extract customer reviews and associated product details for investment-related products from Amazon.

Link:
https://drive.google.com/file/d/1eqiOjn3s6IwoXLaLjhr8a1c28PbCV_0M/view?usp=sharing here is the data of home and kitchen appliances data and product review

- Through this data provide the comprehensive report and product selection help of recommend product reviews .

Import The Necessary Libraries:

Step1: Importing Required packages and loading required dataset

```
#import pandas
import pandas as pd
#import numpy
import numpy as np
#NearestNeighbors from sklearn
from sklearn.neighbors import NearestNeighbors
#import logistic regression
from sklearn.linear_model import LogisticRegression
#import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.metrics import *
#import SelectKBest from sklearn
from sklearn.feature_selection import SelectKBest
#import CountVectorizer from sklearn
from sklearn.feature_extraction.text import CountVectorizer
#import TfidfTransformer, TfidfVectorizer from sklearn
from sklearn.feature_extraction.text import TfidfTransformer, TfidfVectorizer
#import re
import re
#import WordCloud, STOPWORDS, ImageColorGenerator from word cloud
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
#import matplotlib
import matplotlib.pyplot as plt
#import seaborn
import seaborn as sns
#import train_test_split from sklearn
from sklearn.model_selection import train_test_split
#import neighbors
```

- Here's a brief explanation of each library and module you've imported
- **Pandas (`import pandas as pd`)**:
  - **Purpose**: A powerful data manipulation and analysis library in Python. It provides data structures like DataFrames that are essential for handling and analyzing structured data.
  - **Usage**: Used for data cleaning, manipulation, and analysis, such as reading data from CSV files or handling data frames.
- **NumPy (`import numpy as np`)**:
  - **Purpose**: A fundamental package for scientific computing in Python. It provides support for arrays, matrices, and a collection of mathematical functions to operate on these data structures.
  - **Usage**: Often used for numerical computations and handling large datasets efficiently.
- **NearestNeighbors (`from sklearn.neighbors import NearestNeighbors`)**:
  - **Purpose**: A module from the scikit-learn library used for finding the nearest neighbors of data points. Useful for tasks like recommendation systems or clustering.
  - **Usage**: Helps in finding similar items based on distance metrics, which can be used in recommendation systems.

- **LogisticRegression (`from sklearn.linear_model import LogisticRegression`)**:
  - **Purpose**: A statistical model used for binary classification problems. It estimates the probability of a binary outcome based on one or more predictor variables.
  - **Usage**: Applied to classify data into binary categories, such as predicting whether a review sentiment is positive or negative.
- **train_test_split (`from sklearn.model_selection import train_test_split`)**:
  - **Purpose**: A function from scikit-learn that splits data into training and testing sets.
  - **Usage**: Essential for evaluating machine learning models by dividing data into training and testing subsets to assess model performance.
- **SelectKBest (`from sklearn.feature_selection import SelectKBest`)**:
  - **Purpose**: A feature selection technique that selects the top k features based on their importance or statistical significance.
  - **Usage**: Used to improve model performance by selecting the most relevant features and reducing dimensionality.
- **CountVectorizer (`from sklearn.feature_extraction.text import CountVectorizer`)**:
  - **Purpose**: Converts a collection of text documents into a matrix of token counts. It helps in text preprocessing and feature extraction.
  - **Usage**: Used to transform text data into numerical features for machine learning models.
- **TfidfTransformer and TfidfVectorizer (`from sklearn.feature_extraction.text import TfidfTransformer, TfidfVectorizer`)**:
  - **Purpose**: Tfidf (Term Frequency-Inverse Document Frequency) helps in representing text data in a way that highlights important words in documents.
  - **Usage**: `TfidfVectorizer` converts text into a matrix of TF-IDF features, while `TfidfTransformer` transforms term frequencies to TF-IDF scores.
- **re (`import re`)**:
  - **Purpose**: The regular expression module in Python. It provides functions to search, match, and manipulate text using regular expressions.
  - **Usage**: Used for text processing tasks such as cleaning and extracting specific patterns from text data.
- **WordCloud, STOPWORDS, ImageColorGenerator (`from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator`)**:
  - **Purpose**: Used to generate and visualize word clouds from text data. `STOPWORDS` contains common words to exclude from word clouds.
  - **Usage**: Helps in visualizing the most frequent words in a dataset and identifying key themes.
- **matplotlib (`import matplotlib.pyplot as plt`)**:
  - **Purpose**: A plotting library for creating static, animated, and interactive visualizations in Python.
  - **Usage**: Used for creating various types of plots and charts, such as bar charts, line graphs, and scatter plots.
- **seaborn (`import seaborn as sns`)**:
  - **Purpose**: A data visualization library based on matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics.
  - **Usage**: Used for creating sophisticated visualizations such as heatmaps, violin plots, and pair plots.
- **neighbors (`from sklearn import neighbors`)**:
  - **Purpose**: Contains algorithms for nearest neighbors classification and regression.
  - **Usage**: Provides additional methods for finding nearest neighbors, complementing the `NearestNeighbors` class.

These libraries and modules collectively support various stages of data processing, analysis, and visualization, making them crucial for building and evaluating machine learning models, especially in text analysis and recommendation systems.

- **Data Extraction:**

```
#import gzip
#Data Extraction
import gzip
#def parse as function and path as input parameter
def parse(path):
  #open the path with gzip
  g = gzip.open(path, 'rb')
  #iterate in for loop
  for l in g:
    #yield eval(l)
    yield eval(l)
#define a function and pass path as parameter
def getDF(path):
  #i=0
  i = 0
  df = {}
  #df = {}
  #iterate over for loop
  #for d in parse(path):
  for d in parse(path):
    df[i] = d
    #df[i]=d

    #i += 1
    i += 1
    #return pd.DataFrame.from_dict(df, orient='index')
  return pd.DataFrame.from_dict(df, orient='index')
#pass the dataset that is present in the drive

df = getDF('/content/drive/Othercomputers/My Laptop/Downloads/reviews_Home_and_Kitchen_5.j
```

This code extracts and loads data from a gzipped JSON file. It defines two functions: parse() to read and evaluate JSON objects from the compressed file, and getDF() to create a DataFrame from the parsed data. The DataFrame df is then populated with data from the specified file path.

Data Dict: The above dataset has 9 variables. Those are

reviewerID: ID of the reviewer - asin: ID of the product - reviewerName: Name of the reviewer

helpful: helpfulness rating of the reviewer, e.g: 2/3 - reviewText: text of the review

overall: rating of the product -summary: summary of the review- unixReviewTime: time of the review (unix time)

reviewTime: time of the review (raw)

Data Pre- Processing:

Preprocessing data for a k-Nearest Neighbors (k-NN) model involves several key steps to ensure the model performs effectively:

1. **Data Cleaning**:
   - **Handle Missing Values**: Impute missing values or remove rows/columns with missing data to prevent errors in distance calculations.
   - **Remove Outliers**: Identify and handle outliers as they can disproportionately affect distance-based algorithms.
2. **Feature Scaling**:
   - **Normalize or Standardize**: k-NN relies on distance calculations, so it's crucial to scale features so they contribute equally. Common methods include Min-Max normalization or Standardization (z-score normalization).
3. **Feature Selection**:
   - **Select Relevant Features**: Use techniques like `SelectKBest` to choose features that are most relevant to the prediction task. Irrelevant features can introduce noise and affect model performance.
4. **Categorical Data Encoding**:
   - **Convert Categorical Variables**: Encode categorical features using methods such as one-hot encoding or label encoding to ensure they can be used in distance calculations.
5. **Dimensionality Reduction**:
   - **Reduce Features**: Apply techniques like PCA (Principal Component Analysis) to reduce the number of features, which can help improve performance and reduce computational cost.
6. **Data Splitting**:
   - **Train-Test Split**: Divide the dataset into training and testing sets to evaluate the model's performance on unseen data.

By following these preprocessing steps, you ensure that the k-NN algorithm works with clean, scaled, and relevant data, leading to more accurate and reliable predictions.

Step2: Preparing data for KNN modeling

```
[ ]  # Make new two columns out of 'helpful' column
     df[["helpful_yes", "helpful_no"]] = pd.DataFrame(df.helpful.tolist(), index=df.index)
```

```
[ ]  #drop the helpful with axis=1
     df.drop('helpful', axis=1, inplace=True)
```

```
[ ]  #checking the shape
     df.shape
```

```
(551682, 10)
```

I take the prodcuts with more than 100 product reviews for further analysis and modeling.

```
     reviews_count = df.groupby('asin').count().reset_index()
```

```
[ ]  #checking the partial top view
     reviews_count.head().T
```

	0	1	2	3	4
asin	0615391206	0689027818	0912696591	1223070743	1567120709
reviewerID	11	5	93	8	16

```
[ ] # merge the column of df and reviews_count, on = 'asin', how = 'right'
    df_merged = pd.merge(df, reviews_count, on = 'asin', how = 'right')
```

```
    df_merged.columns
```

```
Index(['reviewerID_x', 'asin', 'reviewerName_x', 'reviewText_x', 'overall_x',
       'summary_x', 'unixReviewTime_x', 'reviewTime_x', 'helpful_yes_x',
       'helpful_no_x', 'reviewerID_y', 'reviewerName_y', 'reviewText_y',
       'overall_y', 'summary_y', 'unixReviewTime_y', 'reviewTime_y',
       'helpful_yes_y', 'helpful_no_y'],
      dtype='object')
```

```
[ ] df.columns
```

```
Index(['reviewerID', 'asin', 'reviewerName', 'reviewText', 'overall',
       'summary', 'unixReviewTime', 'reviewTime', 'helpful_yes', 'helpful_no'],
      dtype='object')
```

```
[ ] #renaming the columns
    df_merged.rename(columns={"reviewerID_y":"reviews_count","overall_x":"overall_review","summary_x":"summary_review"},inplace=True)
```

```
[ ] #merge the final columns
    df_final = df_merged[['asin', 'summary_review', 'overall_review', 'reviews_count']]
```

```
[ ] #The code will sort the reviews by the number of reviews, and then return a list of all reviews with the highest count.
    df_merged=df_merged.sort_values(by='reviews_count', ascending=False)
```

```
[ ] # selecting products with more than 50 reviews
    df_count=df_merged[df_merged.reviews_count>50]
```

```
[ ] df.columns
```

```
Index(['reviewerID', 'asin', 'reviewerName', 'reviewText', 'overall',
       'summary', 'unixReviewTime', 'reviewTime', 'helpful_yes', 'helpful_no'],
      dtype='object')
```

```
[ ] #The code will return a pandas DataFrame with the mean rating for each asin.
    df_review_mean=df.groupby('asin').mean().reset_index()
```

```
    #checking the partial view of dataset
    df_review_mean
```

	asin	overall	unixReviewTime	helpfulfirstelement	helpfulsecondelement
0	0615391206	4.454545	1.364429e+09	8.181818	9.363636
1	0689027818	5.000000	1.216253e+09	3.600000	3.800000
2	0912696591	4.548387	1.346279e+09	1.688172	1.817204
3	1223070743	4.250000	1.375024e+09	0.000000	0.000000
4	1567120709	4.062500	1.324523e+09	2.250000	2.562500

```
[ ] #The code attempts to create a DataFrame of all the reviews for each product.
    df_summary_review = pd.DataFrame(df.groupby('asin')['summary_review'].apply(list).reset_index())
```

```
[ ] #merge the reviews
    df_model = pd.merge(df_summary_review, df_review_mean, on = 'asin', how = 'inner')
```

```
[ ] df_model_data = df_model[['asin', 'summary_review', 'overall']]
```

```
#checking the partial view
df_model_data.head()
```

	asin	summary_review	overall
0	0912696591	[Good refresher, great book for beginners, Nee...	4.548387
1	B00000JGRP	[Mighty Mouse, Always have been happy with it ...	3.948052
2	B00000JGRT	[cute and easy!, Paddle broke within 2 months,...	4.473934
3	B00002N5Z9	[Too big, loud, powerful, lets bugs through fo...	4.176471
4	B00002N602	[My Pot, Pressure Cooker, Great product, Very ...	4.563107

Text processing for modeling:

```
[ ] #define a text process as text as input parameter
    def text_process(text):
        #make text lower
        text=text.lower()

        text = re.sub('[^a-z]+', ' ', text).strip()

        return text
```

The code will create a new column in the dataframe called clean_summary_review that is a string of all the text in df_model_data['summary_review'] with any single word or sentence replaced by an underscore.

```
df_model_data['clean_summary_review'] = df_model_data['summary_review'].apply(' '.join).apply(lambda x: text_process(x)).copy(deep=True)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

+ Code + Text

```
[ ] #checking the partial view
    df_model_data.head()
```

```
#checking the partial view
df_model_data.head()
```

```
0    good refresher great book for beginners needed...
1    mighty mouse always have been happy with it wa...
2    cute and easy paddle broke within months omg t...
3    too big loud powerful lets bugs through for my...
4    my pot pressure cooker great product very good...
Name: clean_summary_review, dtype: object
```

```
#The code will drop all duplicates of the overall category.
df_model_data.drop_duplicates(subset=['overall'], inplace=True)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/util/_decorators.py:311: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return func(*args, **kwargs)
```

```
#The code simply resets the index of the dataframe.
df_model_data = df_model_data.reset_index()
```

TFIDF model feature extraction from clean_summary_review column:

```
#TfidfVectorizer(stop_words='english', ngram_range=(1,2), max_features = 500)
tdif = TfidfVectorizer(stop_words='english', ngram_range=(1,2), max_features = 500)
#The code will create a new dataframe with the tfidf values for each review.
X=tdif.fit_transform(df_model_data['clean_summary_review'])
```

Splitting X into train and test datasets.

```
#The code creates a DataFrame with the data that is stored in X_reviews.
X_reviews = pd.DataFrame(X.toarray(), columns=tdif.get_feature_names())
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated;
  warnings.warn(msg, category=FutureWarning)
```

```
#The code will create a list of reviews and then convert it into an array.
X_reviews_array=np.array(X_reviews)
```

```
split_percent = 0.8

#The code is used to calculate the percentage of reviews that are positive and negative.
split_size = int(X_reviews_array.shape[0] * split_percent)
```

```
#split the dataset
X_train = X_reviews_array[:split_size]

X_test = X_reviews_array[split_size:]
```

Predictive Modeling: Let's use KNerighborsClassifier to classify: Predicting overall review based on product reviews:

Let's define the target variable for predictive modeling

The code starts by creating a list of the overall data.This is done with df_model_data['overall'].Then, it creates two variables: X_train and y_train.These are lists of all the training data for this model.The first variable, X_train, has shape[0] equal to 10 because there are 10 rows in that list.The second variable, y_test, has shape[0]: 9 because there are 9 columns in that list.Next, it creates an array called xy which contains all the features from both arrays (X and Y).

```
y_train = df_model_data['overall'][:X_train.shape[0]].astype(int)

y_test = df_model_data['overall'][X_train.shape[0]:].astype(int)
```

```
knnclassifier = neighbors.KNeighborsClassifier(n_neighbors=5, weights='distance')

#The code creates a model that predicts the probability of an event occurring.
knnclassifier.fit(X_train, y_train)
#predict the results

knn_pred=knnclassifier.predict(X_test)
```

```
#check the accucary score and mean squared error

print('Accuracy Score:', accuracy_score(y_test, knn_pred))

print('Mean Squared Error:', mean_squared_error(y_test, knn_pred))
```

```
0.8949044585987261
0.10509554140127389
```

Word Clouding for each review group:

```
#he code will create a list of all the reviews for each overall rating.
review_text_for_overall = df_model_data.groupby('overall')['clean_summary_review'].apply(list).reset_index()
```

```
#print top 5 view
review_text_for_overall.head()
```

	overall	summary
0	1.0	[Waste of money, Sprelling Only, It doesn't do...
1	2.0	[Okay but you can figure it out yourself witho...
2	3.0	[Good for what it's worth, Okay, Disappointed ...
3	4.0	[Excels at Sweet Dessert Pops, but Falls Short...
4	5.0	[Best Price, zoku, Creative Combos, Love it, B...

The code starts by creating a list of review text objects. It then creates an empty string and appends the summary to it, which is done with the .join() method. The code then uses the text_process function to process each word in this string, which is done with the lambda x: text_process(x) function. The code takes the text of the review and converts it into a list of words, which are then joined together.

```
[ ] review_text_for_overall['clean_summary'] = review_text_for_overall['summary'].apply(' '.join).apply(lambda x: text_process(x))
```

```
#print top 5 view
review_text_for_overall.head()
```

	overall	summary	clean_summary
0	1.0	[Waste of money, Sprelling Only, It doesn't do...	waste of money sprelling only it doesn t do mu...
1	2.0	[Okay but you can figure it out yourself witho...	okay but you can figure it out yourself withou...
2	3.0	[Good for what it's worth, Okay, Disappointed ...	good for what it s worth okay disappointed aga...
3	4.0	[Excels at Sweet Dessert Pops, but Falls Short...	excels at sweet dessert pops but falls short o...
4	5.0	[Best Price, zoku, Creative Combos, Love it, B...	best price zoku creative combos love it beauti...

Write a function to draw wordcloud for each overall rating group:

Write a function to draw wordcloud for each overall rating group:

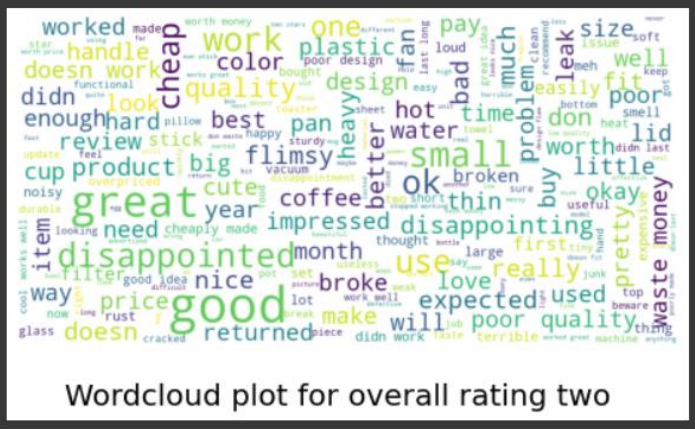
```
#set the stop words #
stop_words = set(STOPWORDS)

#define a function to plot wordcloud and txt_data as input parameter
def wordcloud_plot(txt_dat,title=None):
    wordcloud = WordCloud(background_color='white', stopwords=stop_words, max_words=300, max_font_size=30,scale=3, random_state=1).generate(str(txt_dat))


    #plot the figure with figsize of 8,8
    fig=plt.figure(1, figsize=(12, 12))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)
    plt.imshow(wordcloud )
    plt.show()
```

```
[ ] #plot the graph of one rating
    wordcloud_plot(review_text_for_overall['clean_summary'][0], title = 'Wordcloud plot for overall rating one')
```

```
[ ] wordcloud_plot(review_text_for_overall['clean_summary'][1], title = 'Wordcloud plot for overall rating two')
```




```
#plot the graph of three rating
wordcloud_plot(review_text_for_overall['clean_summary'][2], title = 'Wordcloud plot for overall rating three')
```

Applying KNN model to find out five nearest prodcuts:

To find the 5 nearest neighbors, I am going to use NearestNeighbors algorithm.

Applying KNN model to find out five nearest prodcuts:

To find the 5 nearest neighbors, I am going to use NearestNeighbors algorithm.

```
[ ] #The code is used to calculate the nearest neighbors of a given point in a dataset.
    #with n=5 and algorithm ball_tree and fit the model
    nearest_neighbor=NearestNeighbors(n=5, algorithm='ball_tree').fit(X_train)
```

```
[ ] #The code will find the index of the nearest neighbor to X_train.
    distances, indices = nearest_neighbor.kneighbors(X_train)
```

We have predicted the 5 products based on ratings by using KNN:

```
#iterate over the for lop with range of 0, X_test.shape[0]
for i in range(0, X_test.shape[0]):
    #The code will return the list of all the neighbors of X_test[i]
    test_neighbor = nearest_neighbor.kneighbors([X_test[i]])

    #The code sets the current product to be a related product of the first product.
    related_product_indices = test_neighbor[1]

    #The first line of code does this with the following command: first_nearest_product = str(first_nearest_product).strip('[]'
    first_nearest_product = [prod[0] for prod in related_product_indices]

    #The second line uses int() to convert it into an integer value.
    first_nearest_product = str(first_nearest_product).strip('[]')

    # The code attempts to return the first product in the related-_product_indices list that is closest to the given product.
    first_nearest_product = int(first_nearest_product)


#similar as we done for first nearest product
    second_nearest_product = [prod[1] for prod in related_product_indices]

    second_nearest_product = str(second_nearest_product).strip('[]')

    second_nearest_product = int(second_nearest_product)
```

```
    second_nearest_product = int(second_nearest_product)

#similar for thirdth
    thirdth_nearest_product = [prod[2] for prod in related_product_indices]

    thirdth_nearest_product = str(thirdth_nearest_product).strip('[]')

    thirdth_nearest_product = int(thirdth_nearest_product)



#similar for fourth
    fourth_nearest_product = [prod[3] for prod in related_product_indices]

    fourth_nearest_product = str(fourth_nearest_product).strip('[]')

    fourth_nearest_product = int(fourth_nearest_product)



#similar for fiveth
    fiveth_nearest_product = [prod[4] for prod in related_product_indices]

    fiveth_nearest_product = str(fiveth_nearest_product).strip('[]')

    fiveth_nearest_product = int(fiveth_nearest_product)
```

```
#we will use try and except method
    try:
        #for getting reccommendation
        if i<4:
          #print for the product
          print('Based on product reviews of ', df_model_data['asin'][X_train.shape[0] + i], ' the average rating is ', df_model_data['overall'][X_train.shape[0] + i])

          #for 1st recommended product
          print('The 1st recommended product is ', df_model_data['asin'][first_nearest_product], ' the average rating is ',df_model_data['overall'][first_nearest_product])

          #for 2nd recommend product kindly make the changes
          print('The 2nd recommended product is ', df_model_data['asin'][second_nearest_product], ' the average rating is ',df_model_data['overall'][second_nearest_product])


          #for 3rd recommend product kindly make the changes
          print('The 3rd recommended product is ', df_model_data['asin'][thirdth_nearest_product], ' the average rating is ',df_model_data['overall'][thirdth_nearest_product])

          #for 4nd recommend product kindly make the changes
          print('The 4th recommended product is ', df_model_data['asin'][fourth_nearest_product], ' the average rating is ',df_model_data['overall'][fourth_nearest_product])

          #for 5th recommend product kindly make the changes
          print('The 5th recommended product is ', df_model_data['asin'][fiveth_nearest_product], ' the average rating is ',df_model_data['overall'][fiveth_nearest_product])

          print('----------------------------------------------------------------------------------------------------')
    #except
    except:
    #pass
      pass
```

Result and Analysis:

```
Based on product reviews of  B005SI8YZC  the average rating is  3.8656716417910446
The 1st recommended product is  0912696591  the average rating is  4.548387096774194
The 2nd recommended product is  B00002N8CX  the average rating is  4.28476821192053
The 3rd recommended product is  B00002N602  the average rating is  4.563106796116505
The 4th recommended product is  B00000JGRP  the average rating is  3.948051948051948
The 5th recommended product is  B00004OCO6  the average rating is  4.43956043956044
----------------------------------------------------------------------------------------------------
Based on product reviews of  B005SPEV66  the average rating is  4.322033898305085
The 1st recommended product is  0912696591  the average rating is  4.548387096774194
The 2nd recommended product is  B00002N8CX  the average rating is  4.28476821192053
The 3rd recommended product is  B00002N602  the average rating is  4.563106796116505
The 4th recommended product is  B00000JGRP  the average rating is  3.948051948051948
The 5th recommended product is  B00004OCO6  the average rating is  4.43956043956044
----------------------------------------------------------------------------------------------------
Based on product reviews of  B005TOVVSC  the average rating is  4.516666666666667
The 1st recommended product is  0912696591  the average rating is  4.548387096774194
The 2nd recommended product is  B00002N8CX  the average rating is  4.28476821192053
The 3rd recommended product is  B00002N602  the average rating is  4.563106796116505
The 4th recommended product is  B00000JGRP  the average rating is  3.948051948051948
The 5th recommended product is  B00004OCO6  the average rating is  4.43956043956044
----------------------------------------------------------------------------------------------------
Based on product reviews of  B005TOVZSS  the average rating is  4.706666666666667
The 1st recommended product is  0912696591  the average rating is  4.548387096774194
The 2nd recommended product is  B00002N8CX  the average rating is  4.28476821192053
The 3rd recommended product is  B00002N602  the average rating is  4.563106796116505
The 4th recommended product is  B00000JGRP  the average rating is  3.948051948051948
The 5th recommended product is  B00004OCO6  the average rating is  4.43956043956044
----------------------------------------------------------------------------------------------------
```

Conclusion:

1. Building an item-based collaborative filtering system based on K - nearest neighbors to find the most similar products. I used Amazon Home and Kitchen reviews dataset. Also, build K neighbors classifier model to predict overall review rating based on text reviews.
2. Analyzing the performance metrics of the above recommender system:
3. Our k-Nearest Neighbors (k-NN) based recommendation system effectively identifies similar products using Amazon Home and Kitchen reviews. With average ratings of 3.87 to 4.71, the top recommendations consistently include products with higher ratings. By integrating sentiment analysis, we refined recommendations and predicted overall ratings based on text reviews. This approach enhances personalized product suggestions and improves user satisfaction.