Table of Contents

- 1 Dataset Overview
- 2 Part A: Data Preprocessing
- 3 Part B: Exploratory Data Analysis
- 4 Part C: Feature Construction
- 5 Part D: Model Training and Evaluation
- 6 Part E: Model Deployment
  - 6.1 Deployment specification
- 7 Part F: Follow-up questions

Final Exam Project: Amazon Review Rating Prediction

In this project, you are provided with a dataset of Amazon product reviews. You will apply what you have learned in this course to analyze the dataset and build a predictive model of review ratings. Your deliverables consist of:

1. **scoring_uri.txt**. After training your model, you will deploy it to a public endpoint on Azure, similar to what you did in Project 6 and 7. Your endpoint URI should be included in this file. The autograder will send test data to your endpoint and evaluate the accuracy of your model's predictions. This part is worth 30 points.
   - 15 points are awarded when your model is deployed successfully. The autograder will send a request to your deployed URI, then check that the status code is 200 and the response JSON is correctly formatted.
   - 15 points are awarded when your deployed model also meets the accuracy requirement. The grader will compare your predicted labels with the ground truth labels to determine the accuracy score.
2. **This notebook**. While the autograder will **not** be running your code, there are a number of follow-up questions in Part F that you should answer. **You are encouraged to look at these questions before you start coding.** These are manually graded and worth 40 points.
3. **references**. Include your reference sources here.

In each of the following sections, we will provide some high-level suggestions. You are free to decide on your implementations, or choose a different direction. You can also import any Python package that you find useful, or reuse any of the code you had written in prior projects (remember to mention any reused code in the `references` file).

One point to keep in mind as you begin planning your data processing pipeline is that any processing object that has been fit on the training data (e.g., sklearn's `TfidfVectorizer` or `LogisticRegression`) should later be saved to pickle files and included in the deployment

step, so that you can use them to transform or predict the test data. See Section 1 in the Model deployment primer on the common pitfalls in loading custom functions or classes in the Azure deployment environment. Note that it is possible to complete every task in this exam without defining any custom function.

We start by importing some basic packages. Feel free to remove or add any others as you progress through the tasks.

In [1]:
```
import pickle, re, requests, json, os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import nltk
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet
nltk.download("wordnet", quiet = True)
nltk.download("punkt", quiet = True)
nltk.download('averaged_perceptron_tagger', quiet = True)
nltk.download("stopwords", quiet = True)
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

Dataset Overview

Our dataset is included in the tab-delimited file `amazon_reviews.tsv` . Here is a brief description of the dataframe columns:

Column name	Description
marketplace	2 letter country code of the marketplace where the review was written.
customer_id	Random identifier that can be used to aggregate reviews written by a single author.
review_id	The unique ID of the review.
product_id	The unique Product ID the review pertains to. In the multilingual dataset the reviews for the same product in different countries can be grouped by the same product_id.
product_parent	Random identifier that can be used to aggregate reviews for the same product.
product_title	Title of the product.
product_category	Broad product category that can be used to group reviews (also used to group the dataset into coherent parts).
helpful_votes	Number of helpful votes.
total_votes	Number of total votes the review received.
vine	Review was written as part of the Vine program.
verified_purchase	The review is on a verified purchase.
review_headline	The title of the review.
review_body	The review text.

Column name	Description
review_date	The date the review was written.
star_rating	The 1-5 star rating of the review.

The column which we will need to predict is `star_rating`, which contains integer values from 1 to 5 (inclusive). While the nature of this problem is ordinal regression, for simplicity in this project, we will only treat it as a multi-class classification problem. In other words, we assume each review is associated with one label (1, 2, 3, 4 or 5), and attempt to predict a review's label based on its content and metadata.

We provide the starting code to read the data for you. Note that there are some malformed rows in the original tsv file, so we set `error_bad_lines` to `False`.

In [2]:
```
def read_data():
    return pd.read_csv("amazon_reviews.tsv", sep="\t", header=0, error_bad_lines
        "marketplace": str,
        "customer_id":str,
        "review_id":str,
        "product_id":str,
        "product_parent":str,
        "product_title":str,
        "product_category":str,
        "star_rating":int,
        "helpful_votes":int,
        "total_votes":int,
        "vine":str,
        "verified_purchase":str,
        "review_headline":str,
        "review_body":str,
        "review_date":str
    })

df_reviews = read_data()
```

```
/var/folders/n_/shg_7khn6w345nmddwxlyglc0000gn/T/ipykernel_77003/1628184016.py:
2: FutureWarning: The error_bad_lines argument has been deprecated and will be r
emoved in a future version. Use on_bad_lines in the future.


  return pd.read_csv("amazon_reviews.tsv", sep="\t", header=0, error_bad_lines=F
alse, dtype={
```

Part A: Data Preprocessing

Since this project only involves review rating prediction, you can remove the following irrelevant features:

- `marketplace`
- `customer_id`
- `review_id`

- `product_id`
- `product_parent`
- `product_category` (since all the products belong to the same category)
- `review_date`

The primary rating predictor will be the review content (i.e., `review_body`). We suggest you perform the following preprocessing tasks from **Project 4** on `review_body`.

- Remove all html tags.
- Clean punctuations and special characters.
- Apply lower casing.
- Tokenize the content, then apply stemming or lemmatization to each token.
- Remove English stopwords.

In [4]:

```
## YOUR DATA PREPROCESSING CODE HERE ##
df = pd.read_csv('processed.csv', index_col=0)

def clean_text(text):
    tags = re.compile('<.*?>')
    text = re.sub(tags, '', text)
    #convert to lower case
    text1 = text.lower()
    #remove 's
    text2 = text1.replace("'s ", ' ')
    text2 = re.sub('\'s$', '', text2)
    #renive apostrophe character '
    text3 = text2.replace("'", "")
    return text3.strip()

def tokenize(cleaned_text):
    text1 = nltk.word_tokenize(cleaned_text)
    regex_exp = re.compile('[a-zA-Z0-9]+')
    result = sum([regex_exp.findall(s) for s in text1], [])

    return result

def lemmatize(tokens):
    lemmatizer = WordNetLemmatizer()
    english_stopwords = set(nltk.corpus.stopwords.words('english'))
    lemmatized_results = []

    for i in tokens:
        pos_tag = nltk.pos_tag(list([i]))[0][1]

        if pos_tag.startswith('J'):
            pos = wordnet.ADJ
        elif pos_tag.startswith('V'):
            pos = wordnet.VERB
        elif pos_tag.startswith('R'):
            pos = wordnet.ADV
        else:
            pos = wordnet.NOUN

        lemmatized_token = lemmatizer.lemmatize(word=i, pos=pos)
```

```
        if len(lemmatized_token) >= 2 and lemmatized_token not in english_stopwo
            lemmatized_results.append(lemmatized_token)

    return lemmatized_results

def preprocess_text(text):
    cleaned_text = clean_text(text)
    tokens = tokenize(cleaned_text)
    return lemmatize(tokens)

def preprocess(df):
    df = df.drop(['marketplace', 'customer_id', 'review_id', 'product_id', 'prod
    df['processed_review_body'] = df['review_body'].map(preprocess_text)
    df['verified_purchase'] = df['verified_purchase'].replace(['Y', 'N'], [1, 0]
    df['join_review_body'] = df['processed_review_body'].str.join(" ")
#     df['processed_review_headline'] = df['review_headline'].map(preprocess_tex
    return df

# df = preprocess(df_reviews)
```

Part B: Exploratory Data Analysis

You should investigate the feature and label distributions in your exploratory data analysis. A good starting point is the techniques and tools from **Project 3**, in particular the Pandas Profiler library. Pay attention to rows with missing data or outliers, and think about whether to include or remove them. In case the rating distribution is highly imbalanced, you may also consider oversampling the minority classes (i.e., making random copies of the rows with the minority classes) to get a more balanced sample. However, oversampling should only be performed in part D on the training data only, after you have performed a train-test split.

In [16]:
```
## YOUR EXPLORATORY DATA ANALYSIS CODE HERE ##
from pandas_profiling import ProfileReport
display(df.head())
display(df.describe())
df.boxplot(by ='verified_purchase', column =['star_rating'], grid = False)
df.boxplot(by ='vine', column =['star_rating'], grid = False)
df.boxplot(by ='star_rating', column =['helpful_votes'], grid = False)
df['star_rating'].value_counts().plot(kind='bar')
profile = ProfileReport(df)
profile
```

	product_title	star_rating	helpful_votes	total_votes	vine	verified_purchase	review_headline
0	Jawbone Jambox Wireless Bluetooth Speaker - Bl...	5	0	0	N	1	Good thing in a small package
1	Green Silicone skin sport nano case for Apple ...	3	6	6	N	0	Good enough

	product_title	star_rating	helpful_votes	total_votes	vine	verified_purchase	review_headline
2	original brand new proscan TV remote control F...	1	0	0	N	1	... it to even though it was advertised to wor...
3	Bluetooth Headphones - Fit Acoustics Wireless ...	5	1	1	N	1	Excellent product excellent customer service
4	TsirTech 18 Items Luxury Accessory Bundle for ...	5	0	0	N	1	great bundle, love the texting glove,

	star_rating	helpful_votes	total_votes	verified_purchase
count	99909.000000	99909.000000	99909.000000	99909.000000
mean	3.898818	1.496962	1.934190	0.842727
std	1.462946	10.741559	11.758621	0.364060
min	1.000000	0.000000	0.000000	0.000000
25%	3.000000	0.000000	0.000000	1.000000
50%	5.000000	0.000000	0.000000	1.000000
75%	5.000000	1.000000	1.000000	1.000000
max	5.000000	1219.000000	1234.000000	1.000000

Overview

Dataset statistics

Number of variables	10
Number of observations	99909
Missing cells	99
Missing cells (%)	< 0.1%
Duplicate rows	24
Duplicate rows (%)	< 0.1%
Total size in memory	10.4 MiB
Average record size in memory	109.2 B

Variable types

Categorical	7
Numeric	2
Boolean	1

Alerts

Dataset has 24 (< 0.1%) duplicate rows	**Duplicates**
`product_title` has a high cardinality: 34662 distinct values	**High cardinality**
`review_headline` has a high cardinality: 66026 distinct values	**High cardinality**

Out[16]:

Part C: Feature Construction

The next task is to construct numerical features for prediction. We recommend starting with frequency-based features, such as TF and TF-IDF from **Project 5**, based on the review text content. To avoid potential memory issues, make sure to select the best data representation for

your feature matrix. For example, TF-IDF matrices should be in sparse format, which you worked with in **Project 2**.

Once you have finished the model deployment, feel free to revisit this section and experiment with other features (such as those from **Project 7**) to improve your model accuracy. Pay attention to how different data structures can be used to optimize feature construction (e.g., list vs set vs iterator), which you learned in **Project 1**.

In [44]:

```
## YOUR FEATURE CONSTRUCTION CODE HERE ##
label_encoder = LabelEncoder()
df['star_rating'] = label_encoder.fit_transform(df['star_rating'])

def feature_construction_test(df_train, df_test=None):
    tfidf_vectorizer = TfidfVectorizer(analyzer=str.split, tokenizer=str.split,
    x = tfidf_vectorizer.fit(df_train['join_review_body'].values.astype(str))
    tfidf_train = x.transform(df_train['join_review_body'].values.astype(str))

    tfidf_dftrain=pd.DataFrame(tfidf_train.toarray(), columns=tfidf_vectorizer.g
    tfidf_dftrain.reset_index(drop=True, inplace=True)
    df_train.reset_index(drop=True, inplace=True)
    traindf = pd.concat([df_train, tfidf_dftrain], axis=1)

    features = []
    features = tfidf_vectorizer.get_feature_names_out()
    features.append('verified_purchase')

    train_data = traindf[features]
    train_labels = traindf['star_rating']

    if df_test is None:
        test_data = []
        test_labels = []

    else:
        tfidf_test = x.transform(df_test['join_review_body'].values.astype(str))
        tfidf_dftest=pd.DataFrame(tfidf_test.toarray(), columns=tfidf_vectorizer
        tfidf_dftest.reset_index(drop=True, inplace=True)
        df_test.reset_index(drop=True, inplace=True)
        testdf = pd.concat([df_test, tfidf_dftest], axis=1)
        test_data = testdf[features]
        test_labels = testdf['star_rating']

    return train_data, train_labels, test_data, test_labels, tfidf_vectorizer

def feature_construction(df_train):
    tfidf_vectorizer = TfidfVectorizer(analyzer=str.split, tokenizer=str.split,
    x = tfidf_vectorizer.fit(df_train['join_review_body'].values.astype(str))
    tfidf_train = x.transform(df_train['join_review_body'].values.astype(str))

    return tfidf_train, tfidf_vectorizer
train_data, train_labels, test_data, test_labels, tf_idf = feature_construction_
```

```
/Users/iriszhang/opt/anaconda3/lib/python3.9/site-packages/sklearn/feature_extra
ction/text.py:524: UserWarning: The parameter 'token_pattern' will not be used s
ince 'tokenizer' is not None'
  warnings.warn(
/Users/iriszhang/opt/anaconda3/lib/python3.9/site-packages/sklearn/feature_extra
```

```
ction/text.py:559: UserWarning: The parameter 'tokenizer' will not be used since
'analyzer' != 'word'
  warnings.warn(
/Users/iriszhang/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/depreca
tion.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature
_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_n
ames_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Part D: Model Training and Evaluation

It is now time to train machine learning models to predict review ratings. Your goal in this part is to come up with a model that can achieve an **accuracy score of at least 0.60 on the test data**. Given the ground truth labels `y_true` and your predicted labels `y_pred` (each of which contains integer values from 1 to 5), the grader environment will compute the accuracy score as `sklearn.metrics.accuracy_score(y_true, y_pred)`.

As a starting point, you can try out the classical learning algorithms supported by Sklearn:

- Logistic regression
- Support vector machine
- K-nearest neighbor

You can also experiment with some ensemble methods based on these classifiers, for example voting classifier or boosting. You can tune the hyperparameters for each learning algorithm, using the cross validation process in **Project 5**, to identify the best fit model.

Notes:

- Similar to **Project 6**, your trained model will be evaluated on a secret test dataset. Its accuracy on this dataset will be shown in the Sail() Feedback view when you make a submission.

In [ ]:

```
## YOUR MODEL TRAINING CODE HERE ##
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics, svm

train, test = train_test_split(df, test_size=0.2,random_state=0)

def oversampling(train):
    class4,class3,class2,class1,class0=train['star_rating'].value_counts()
    c4 = train[train['star_rating']==4]
    c3 = train[train['star_rating']==3]
    c2 = train[train['star_rating']==2]
    c1 = train[train['star_rating']==1]
    c0 = train[train['star_rating']==0]
    df3 = c3.sample(class4, replace = True)
    df2 = c2.sample(class4, replace = True)
    df1 = c1.sample(class4, replace = True)
    df0 = c0.sample(class4, replace = True)
    oversampled_df = pd.concat([c4,df3,df2,df1,df0], axis=0)
```

```
    return oversampled_df
oversampled_df = oversampling(train)
# X_train, y_train, X_test, y_test, tf_idf = feature_construction_test(oversampl
X_train, y_train, X_test, y_test, tf_idf = feature_construction_test(df, test)

lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
lr_y_pred_train = lr.predict(X_train)
lr_y_pred_test = lr.predict(X_test)
print('lr training score:', metrics.accuracy_score(y_train, lr_y_pred_train))
print('lr testing score:', metrics.accuracy_score(y_test, lr_y_pred_test))

rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)
rf_y_pred_train = lr.predict(X_train)
rf_y_pred_test = rf.predict(X_test)
print('rf training score:', metrics.accuracy_score(y_train, rf_y_pred_train))
print('rf testing score:', metrics.accuracy_score(y_test, rf_y_pred_test))

clf = svm.SVC(kernel='linear')
clf.fit(X_train, y_train)
clf_y_pred_train = lr.predict(X_train)
clf_y_pred_test = clf.predict(X_test)
print('svm training score:', metrics.accuracy_score(y_train, clf_y_pred_train))
print('svm testing score:', metrics.accuracy_score(y_test, clf_y_pred_test))

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
knn_y_pred_train = lr.predict(X_train)
knn_y_pred_test = knn.predict(X_test)
print('svm training score:', metrics.accuracy_score(y_train, knn_y_pred_train))
print('svm testing score:', metrics.accuracy_score(y_test, knn_y_pred_test))
```

Once you have identified your best model, you can fit it on the entire dataset and save all relevant data processing objects to files in a `models` directory, so that you can use it as your deployment directory in the next part.

Notes:

- Your model file names should not contain any space character.

In [56]:

```
## YOUR MODEL SAVING CODE HERE ##
lr = LogisticRegression(max_iter=1000)
train_wholedata, tf_idf = feature_construction(df)
lr.fit(train_wholedata, df["star_rating"].values)

pickle.dump(tf_idf, open('models/tf-idf.pkl', 'wb'))
pickle.dump(lr, open('models/lr.pkl', 'wb'))
```

```
/anaconda/envs/azureml_py38/lib/python3.8/site-packages/sklearn/linear_model/_lo
gistic.py:938: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
```

```
n
  n_iter_i = _check_optimize_result(
```

Part E: Model Deployment

Now you will deploy your model to a local endpoint, test it on some sample data, and then deploy the model to a public endpoint -- this is the same process you performed in **Project 6 and 7**. You can follow the steps in the Model deployment primer. Make sure to pay attention to how it handles custom functions and classes, if you used them in previous steps. Note also that for `myenv.yml`, you should list all the pip packages that you have used in this project, along with their version number.

Deployment specification

Our grader will send a POST request to your endpoint, where the JSON content will be structured as follows:

```
request_content = """{
    "n_reviews" : 2,
    "reviews" : [
        {
            "marketplace" : "US",
            "customer_id" : "20422322",
            "review_id" : "R8MEA6IGAHO0B",
            "product_id" : "B00MC4CED8",
            "product_parent" : "217304173",
            "product_title" : "BlackVue",
            "product_category" : "Mobile_Electronics",
            "helpful_votes" : 0,
            "total_votes" : 0,
            "vine" : "N",
            "verified_purchase" : "Y",
            "review_headline" : "Very Happy!    As advertised.",
            "review_body" : "Everything works perfectly, I'm happy",
            "review_date" : "2015-08-31"
        },
        {
            "marketplace" : "US",
            "customer_id" : "15204307",
            "review_id" : "RZGDLDMGMGKLB",
            "product_id" : "B000I3F91O",
            "product_parent" : "90621845",
            "product_title" : "Sangean H205 AM/FM Weather Alert
Waterproof",
            "product_category" : "Electronics",
            "helpful_votes" : 0,
            "total_votes" : 0,
            "vine" : "N",
            "verified_purchase" : "Y",
            "review_headline" : "Best shower radio I have ever owned",
            "review_body" : "love shower radio own shower radio best
```

```
one",
            "review_date" : "2014-11-08"
        },
    ]
}
"""
```

Here `n_reviews` indicates the number of reviews included in the request, and `reviews` is a list of dictionaries, each having the same schema as a row in the provided dataset (but without the `star_rating` attribute, which is what you need to predict).

The expected response is a JSON with the following format:

```
response_content = {
    "predictions" : [2, 3]
}
```

where the key `predictions` maps to a list of predicted labels, one for each input review in `request_content`. Keep in mind that this list should be a standard Python list, not a Numpy array or Pandas series, which are not JSON serializable.

Notes:

- If you have put your custom functions / classes in a separate Python file, for example `my_custom_code.py` in the current directory, this file will be sent to the deployment environment as well. Therefore, you can include `import my_custom_code` in your scoring script.
- If you use `nltk` in your data processing, note that the Docker environment used for deployment does not have any NLTK data by default. Therefore, you may need to explicitly download them in your `init` function, for example `nltk.download("stopwords")`. If you use the text processing code from Project 4, refer to the `nltk.download` commands provided at the beginning of the Project 4 notebook.

In [22]:

```
## YOUR ENVIRONMENT SETUP CODE HERE ##
import azureml.core
from azureml.core.workspace import Workspace
from azureml.core.model import Model
from azureml.core.webservice import AciWebservice
from azureml.core.model import InferenceConfig
from azureml.core.webservice import Webservice
from azureml.core.environment import Environment
from azureml.core.webservice import LocalWebservice

ws = Workspace.from_config()
register_lr_model = Model.register(
    workspace = ws, model_path = "models/lr.pkl",
    model_name = "lr",
    description = "A lr model trained on simple corpus"
)
register_tfidf_model = Model.register(
    workspace = ws, model_path = "models/tf-idf.pkl",
    model_name = "tfidf",
    description = "A tfidf model trained on simple corpus"
)
```

```
Registering model lr
Registering model tfidf
```

In [28]:

```
%%writefile ./score.py
## YOUR SCORE.PY CODE HERE ##
import pickle
import json
import re
import pandas as pd
import nltk
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet
# import my_custom_code
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from azureml.core.model import Model

def init():
    nltk.download("stopwords", quiet = True)
    nltk.download("wordnet", quiet = True)
    nltk.download("punkt", quiet = True)
    nltk.download('averaged_perceptron_tagger', quiet = True)

    tfidf_path = Model.get_model_path("tfidf")
    lr_path = Model.get_model_path("lr")

    # load the model saved in tfidf.pkl and store it in a global variable
    global lr, tfidf
    with open(lr_path, "rb") as f:
        lr = pickle.load(f)
    with open(tfidf_path, "rb") as g:
        tfidf = pickle.load(g)

def run(input_data):
    # convert input_data from string to JSON
    data = json.loads(input_data)['reviews']
    df = pd.DataFrame(data)

    df2 = preprocess(df)

    data = tfidf.transform(df2['join_review_body'])

    y_pred = lr.predict(data)

    return {
        "predictions" : y_pred.tolist()
    }

def clean_text(text):
    tags = re.compile('<.*?>')
    text = re.sub(tags, '', text)
    #convert to lower case
    text1 = text.lower()
    #remove 's
    text2 = text1.replace("'s ", ' ')
    text2 = re.sub('\'s$', '', text2)
    #renive apostrophe character '
    text3 = text2.replace("'", "")
```

```
    return text3.strip()

def tokenize(cleaned_text):
    text1 = nltk.word_tokenize(cleaned_text)
    regex_exp = re.compile('[a-zA-Z0-9]+')
    result = sum([regex_exp.findall(s) for s in text1], [])

    return result

def lemmatize(tokens):#, stopwords = {}):
    lemmatizer = WordNetLemmatizer()
    english_stopwords = set(nltk.corpus.stopwords.words('english'))
    lemmatized_results = []

    for i in tokens:
        pos_tag = nltk.pos_tag(list([i]))[0][1]

        if pos_tag.startswith('J'):
            pos = wordnet.ADJ
        elif pos_tag.startswith('V'):
            pos = wordnet.VERB
        elif pos_tag.startswith('R'):
            pos = wordnet.ADV
        else:
            pos = wordnet.NOUN

        lemmatized_token = lemmatizer.lemmatize(word=i, pos=pos)

        if len(lemmatized_token) >= 2 and lemmatized_token not in english_stopwo
            lemmatized_results.append(lemmatized_token)

    return lemmatized_results

def preprocess_text(text):#, stopwords = {}):
    # do not modify this function
    cleaned_text = clean_text(text)
    tokens = tokenize(cleaned_text)
    return lemmatize(tokens)#, english_stopwords)

def preprocess(df):
    df = df.drop(['marketplace', 'customer_id', 'review_id', 'product_id', 'prod
    df['processed_review_body'] = df['review_body'].map(preprocess_text)
    df['verified_purchase'] = df['verified_purchase'].replace(['Y', 'N'], [1, 0]
    df['join_review_body'] = df['processed_review_body'].str.join(" ")
    return df
```

```
Overwriting ./score.py
```

In [29]:

```
!cat score.py
```

```
## YOUR SCORE.PY CODE HERE ##
import pickle
import json
import re
import pandas as pd
import nltk
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet
# import my_custom_code
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.linear_model import LogisticRegression
from azureml.core.model import Model

def init():
    nltk.download("stopwords", quiet = True)
    nltk.download("wordnet", quiet = True)
    nltk.download("punkt", quiet = True)
    nltk.download('averaged_perceptron_tagger', quiet = True)

    tfidf_path = Model.get_model_path("tfidf")
    lr_path = Model.get_model_path("lr")

    # load the model saved in tfidf.pkl and store it in a global variable
    global lr, tfidf
    with open(lr_path, "rb") as f:
        lr = pickle.load(f)
    with open(tfidf_path, "rb") as g:
        tfidf = pickle.load(g)

def run(input_data):
    # convert input_data from string to JSON
    data = json.loads(input_data)['reviews']
    df = pd.DataFrame(data)

    df2 = preprocess(df)

    data = tfidf.transform(df2['join_review_body'])

    y_pred = lr.predict(data)

#     results = y_pred+1

    # return the desired response JSON
    return {
        "predictions" : y_pred.tolist()
    }

def clean_text(text):
    tags = re.compile('<.*?>')
    text = re.sub(tags, '', text)
    #convert to lower case
    text1 = text.lower()
    #remove 's
    text2 = text1.replace("'s ", ' ')
    text2 = re.sub('\'s$', '', text2)
    #renive apostrophe character '
    text3 = text2.replace("'", "")
    return text3.strip()

def tokenize(cleaned_text):
    text1 = nltk.word_tokenize(cleaned_text)
    regex_exp = re.compile('[a-zA-Z0-9]+')
    result = sum([regex_exp.findall(s) for s in text1], [])

    return result

def lemmatize(tokens):#, stopwords = {}):
    lemmatizer = WordNetLemmatizer()
    english_stopwords = set(nltk.corpus.stopwords.words('english'))
    lemmatized_results = []
```

```
    for i in tokens:
        pos_tag = nltk.pos_tag(list([i]))[0][1]

        if pos_tag.startswith('J'):
            pos = wordnet.ADJ
        elif pos_tag.startswith('V'):
            pos = wordnet.VERB
        elif pos_tag.startswith('R'):
            pos = wordnet.ADV
        else:
            pos = wordnet.NOUN

        lemmatized_token = lemmatizer.lemmatize(word=i, pos=pos)

        if len(lemmatized_token) >= 2 and lemmatized_token not in english_stopwo
rds:
            lemmatized_results.append(lemmatized_token)

    return lemmatized_results

def preprocess_text(text):#, stopwords = {}):
    # do not modify this function
    cleaned_text = clean_text(text)
    tokens = tokenize(cleaned_text)
    return lemmatize(tokens)#, english_stopwords)

def preprocess(df):
    df = df.drop(['marketplace', 'customer_id', 'review_id', 'product_id', 'prod
uct_parent', 'product_category','review_date'], axis=1)
    df['processed_review_body'] = df['review_body'].map(preprocess_text)
    df['verified_purchase'] = df['verified_purchase'].replace(['Y', 'N'], [1,
0])
    df['join_review_body'] = df['processed_review_body'].str.join(" ")
    return df
```

In [30]:

```
from azureml.core.conda_dependencies import CondaDependencies

environment_file = CondaDependencies.create(pip_packages=[
    'azureml-defaults', 'scikit-learn>=0.22.2', 'pandas', 'nltk'])

with open("myenv.yml","w") as f:
    f.write(environment_file.serialize_to_string())
```

In [31]:

```
# your code here
# create environment variable
myenv = Environment.from_conda_specification(name = "myenv", file_path = "myenv.
myenv.register(workspace = ws)
# provide the scoring script and environment variable to InferenceConfig
inference_config = InferenceConfig(source_directory = '.', entry_script = "score
# set deployment destination to localhost:8890
local_deployment_target = LocalWebservice.deploy_configuration(port = 8890)
# create a service instance
local_service = Model.deploy(
    ws, "local-service",
    [register_lr_model, register_tfidf_model], inference_config,
    local_deployment_target
```

```
)
```

```
/tmp/ipykernel_28629/1590133359.py:10: FutureWarning: azureml.core.model:
To leverage new model deployment capabilities, AzureML recommends using CLI/SDK
v2 to deploy models as online endpoint,
please refer to respective documentations
https://docs.microsoft.com/azure/machine-learning/how-to-deploy-managed-online-e
ndpoints /
https://docs.microsoft.com/azure/machine-learning/how-to-deploy-managed-online-e
ndpoint-sdk-v2 /
https://docs.microsoft.com/azure/machine-learning/how-to-attach-kubernetes-anywh
ere
For more information on migration, see https://aka.ms/acimoemigration.
To disable CLI/SDK v1 deprecation warning set AZUREML_LOG_DEPRECATION_WARNING_EN
ABLED to 'False'
  local_service = Model.deploy(
Downloading model lr:13 to /tmp/azureml_coe659xi/lr/13
Downloading model tfidf:15 to /tmp/azureml_coe659xi/tfidf/15
Generating Docker build context.
Package creation Succeeded
Logging into Docker registry 619eaff98fef44a2bb8772b3d1526a9f.azurecr.io
Logging into Docker registry 619eaff98fef44a2bb8772b3d1526a9f.azurecr.io
Building Docker image from Dockerfile...
Step 1/5 : FROM 619eaff98fef44a2bb8772b3d1526a9f.azurecr.io/azureml/azureml_3a14
88d3dc44600ef9dab44f15c69deb
 ---> 69cf44f076c7
Step 2/5 : COPY azureml-app /var/azureml-app
 ---> 812776c6266f
Step 3/5 : RUN mkdir -p '/var/azureml-app' && echo eyJhY2NvdW50Q29udGV4dCI6eyJzd
WJzY3JpcHRpb25JZCI6ImJiYTIxY2IwLTMzMmYtNGFjMC04ZDQ3LTg2OTNmZjQ1NWE2MCIsInJlc291c
mNlR3JvdXBOYW1lIjoiY2xvdWQtc2hlbGwtc3RvcmFnZS1lYXN0dXMiLCJhY2NvdW50TmFtZSI6ImZpb
mFscyIsIndvcmtzcGFjZUlkIjoiNjE5ZWFmZjktOGZlZi00NGEyLWJiODctNzJiM2QxNTI2YTlmIn0sI
m1vZGVscyI6e30sIm1vZGVsc0luZm8iOnt9fQ== | base64 --decode > /var/azureml-app/mod
el_config_map.json
 ---> Running in f944d92146ad
 ---> 4def846da9bf
Step 4/5 : RUN mv '/var/azureml-app/tmp5__jer_p.py' /var/azureml-app/main.py
 ---> Running in 988f53709d0d
 ---> 7cdfacc7482d
Step 5/5 : CMD ["runsvdir","/var/runit"]
 ---> Running in c84f642af734
 ---> da05460d23fc
Successfully built da05460d23fc
Successfully tagged local-service:latest
Container has been successfully cleaned up.
Image sha256:34d69476999fa2a35a2ec18089acb9b509c818d9268072c8520f6fc164324124 su
ccessfully removed.
Starting Docker container...
Docker container running.
Checking container health...
Local webservice is running at http://localhost:8890
```

We provide the code for testing your local service as follows. This assumes you have previously called `local_service.wait_for_deployment()` . If you use a different variable name for your local service, simply change it accordingly in the provided code.

In [32]:

```
example_json = json.dumps({
    "n_reviews" : 2,
    "reviews" : [
```

```
        {
            "marketplace" : "US",
            "customer_id" : "20422322",
            "review_id" : "R8MEA6IGAHO0B",
            "product_id" : "B00MC4CED8",
            "product_parent" : "217304173",
            "product_title" : "BlackVue",
            "product_category" : "Mobile_Electronics",
            "helpful_votes" : 0,
            "total_votes" : 0,
            "vine" : "N",
            "verified_purchase" : "Y",
            "review_headline" : "Very Happy!    As advertised.",
            "review_body" : "Everything works perfectly, I'm happy",
            "review_date" : "2015-08-31"
        },
        {
            "marketplace" : "US",
            "customer_id" : "15204307",
            "review_id" : "RZGDLDMGMGKLB",
            "product_id" : "B000I3F91O",
            "product_parent" : "90621845",
            "product_title" : "Sangean H205 AM/FM Weather Alert Waterproof",
            "product_category" : "Electronics",
            "helpful_votes" : 0,
            "total_votes" : 0,
            "vine" : "N",
            "verified_purchase" : "Y",
            "review_headline" : "Best shower radio I have ever owned",
            "review_body" : "love shower radio own shower radio best one",
            "review_date" : "2014-11-08"
        },
    ]
})
example_json = bytes(example_json, encoding = "utf8")
output = local_service.run(example_json)
print(output)
```

```
{'predictions': [5, 5]}
```

In [74]:

```
print(local_service.get_logs())
```

```
2022-12-14T03:33:40,040256958+00:00 - iot-server/run
/bin/bash: /azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/libtinfo.s
o.6: no version information available (required by /bin/bash)
/bin/bash: /azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/libtinfo.s
o.6: no version information available (required by /bin/bash)
/bin/bash: /azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/libtinfo.s
o.6: no version information available (required by /bin/bash)
/bin/bash: /azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/libtinfo.s
o.6: no version information available (required by /bin/bash)
2022-12-14T03:33:40,043698271+00:00 - rsyslog/run
2022-12-14T03:33:40,046720858+00:00 - nginx/run
bash: /azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/libtinfo.so.6:
no version information available (required by bash)
2022-12-14T03:33:40,060834032+00:00 - gunicorn/run
2022-12-14T03:33:40,062587241+00:00 | gunicorn/run |
2022-12-14T03:33:40,071305481+00:00 | gunicorn/run | ##########################
####################
```

```
2022-12-14T03:33:40,073789735+00:00 | gunicorn/run | AzureML Container Runtime I
nformation
2022-12-14T03:33:40,076339492+00:00 | gunicorn/run | ##########################
####################
2022-12-14T03:33:40,078366318+00:00 | gunicorn/run |
2022-12-14T03:33:40,083378928+00:00 | gunicorn/run |
2022-12-14T03:33:40,086397315+00:00 | gunicorn/run | AzureML image information:
openmpi4.1.0-ubuntu20.04, Materializaton Build:20221010.v9
2022-12-14T03:33:40,087919109+00:00 | gunicorn/run |
2022-12-14T03:33:40,090206551+00:00 | gunicorn/run |
2022-12-14T03:33:40,092236177+00:00 | gunicorn/run | PATH environment variable:
/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/bin:/opt/miniconda/bin:/u
sr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
2022-12-14T03:33:40,094332307+00:00 | gunicorn/run | PYTHONPATH environment vari
able:
2022-12-14T03:33:40,096206923+00:00 | gunicorn/run |
2022-12-14T03:33:40,098172844+00:00 | gunicorn/run | Pip Dependencies (before dy
namic installation)

EdgeHubConnectionString and IOTEDGE_IOTHUBHOSTNAME are not set. Exiting...
/bin/bash: /azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/libtinfo.s
o.6: no version information available (required by /bin/bash)
2022-12-14T03:33:40,128218305+00:00 - iot-server/finish 1 0
2022-12-14T03:33:40,129774301+00:00 - Exit code 1 is normal. Not restarting iot-
server.
adal==1.2.7
argcomplete==2.0.0
attrs==22.1.0
azure-common==1.1.28
azure-core==1.26.1
azure-graphrbac==0.61.1
azure-identity==1.7.0
azure-mgmt-authorization==2.0.0
azure-mgmt-containerregistry==10.0.0
azure-mgmt-core==1.3.2
azure-mgmt-keyvault==10.1.0
azure-mgmt-resource==21.2.1
azure-mgmt-storage==20.1.0
azureml-core==1.47.0
azureml-dataprep==4.5.7
azureml-dataprep-native==38.0.0
azureml-dataprep-rslex==2.11.4
azureml-dataset-runtime==1.47.0
azureml-defaults==1.47.0
azureml-inference-server-http==0.7.7
backports.tempfile==1.0
backports.weakref==1.0.post1
bcrypt==4.0.1
cachetools==5.2.0
certifi @ file:///opt/conda/conda-bld/certifi_1655968806487/work/certifi
cffi==1.15.1
charset-normalizer==2.1.1
click==8.1.3
cloudpickle==2.2.0
configparser==3.7.4
contextlib2==21.6.0
cryptography==38.0.4
distro==1.8.0
docker==6.0.1
dotnetcore2==3.1.23
```

```
Flask==2.2.2
Flask-Cors==3.0.10
fusepy==3.0.1
google-api-core==2.11.0
google-auth==2.15.0
googleapis-common-protos==1.57.0
gunicorn==20.1.0
humanfriendly==10.0
idna==3.4
importlib-metadata==5.1.0
importlib-resources==5.10.1
inference-schema==1.5
isodate==0.6.1
itsdangerous==2.1.2
jeepney==0.8.0
Jinja2==3.1.2
jmespath==1.0.1
joblib==1.2.0
json-logging-py==0.2
jsonpickle==2.2.0
jsonschema==4.17.3
knack==0.10.1
MarkupSafe==2.1.1
msal==1.20.0
msal-extensions==0.3.1
msrest==0.7.1
msrestazure==0.6.4
ndg-httpsclient==0.5.1
nltk==3.8
numpy==1.23.5
oauthlib==3.2.2
opencensus==0.11.0
opencensus-context==0.1.3
opencensus-ext-azure==1.1.7
packaging==21.3
pandas==1.5.2
paramiko==2.12.0
pathspec==0.10.3
pkginfo==1.9.2
pkgutil_resolve_name==1.3.10
portalocker==2.6.0
protobuf==4.21.11
psutil==5.9.4
pyarrow==9.0.0
pyasn1==0.4.8
pyasn1-modules==0.2.8
pycparser==2.21
Pygments==2.13.0
PyJWT==2.6.0
PyNaCl==1.5.0
pyOpenSSL==22.1.0
pyparsing==3.0.9
pyrsistent==0.19.2
PySocks==1.7.1
python-dateutil==2.8.2
pytz==2022.6
PyYAML==6.0
regex==2022.10.31
requests==2.28.1
requests-oauthlib==1.3.1
```

```
rsa==4.9
scikit-learn==1.2.0
scipy==1.9.3
SecretStorage==3.3.3
six==1.16.0
tabulate==0.9.0
threadpoolctl==3.1.0
tqdm==4.64.1
typing_extensions==4.4.0
urllib3==1.26.13
websocket-client==1.4.2
Werkzeug==2.2.2
wrapt==1.12.1
zipp==3.11.0

2022-12-14T03:33:40,671643856+00:00 | gunicorn/run |
2022-12-14T03:33:40,673585076+00:00 | gunicorn/run | ##########################
####################
2022-12-14T03:33:40,675631003+00:00 | gunicorn/run | AzureML Inference Server
2022-12-14T03:33:40,677622626+00:00 | gunicorn/run | ##########################
####################
2022-12-14T03:33:40,679493742+00:00 | gunicorn/run |
2022-12-14T03:33:41,559746351+00:00 | gunicorn/run | Starting AzureML Inference
Server HTTP.

Azure ML Inferencing HTTP server v0.7.7


Server Settings
---------------
Entry Script Name: /var/azureml-app/final_exam_handout_v1/score.py
Model Directory: azureml-models/
Worker Count: 1
Worker Timeout (seconds): 300
Server Port: 31311
Application Insights Enabled: false
Application Insights Key: None
Inferencing HTTP server version: azmlinfsrv/0.7.7
CORS for the specified origins: None


Server Routes
---------------
Liveness Probe: GET   127.0.0.1:31311/
Score:          POST  127.0.0.1:31311/score

Starting gunicorn 20.1.0
Listening at: http://0.0.0.0:31311 (12)
Using worker: sync
Booting worker with pid: 67
Initializing logger
2022-12-14 03:33:42,134 | root | INFO | Starting up app insights client
logging socket was found. logging is available.
logging socket was found. logging is available.
2022-12-14 03:33:42,135 | root | INFO | Starting up app insight hooks
2022-12-14 03:33:43,727 | root | INFO | Found user script at /var/azureml-app/fi
nal_exam_handout_v1/score.py
2022-12-14 03:33:43,727 | root | INFO | run() is not decorated. Server will invo
ke it with the input in JSON string.
2022-12-14 03:33:43,727 | root | INFO | Invoking user's init function
```

```
00000000-0000-0000-0000-000000000000,/azureml-envs/azureml_d778e71475c08b4077c2b
334ee6055be/lib/python3.8/site-packages/sklearn/base.py:288: UserWarning: Trying
to unpickle estimator LogisticRegression from version 0.22.1 when using version
1.2.0. This might lead to breaking code or invalid results. Use at your own ris
k. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-
limitations
  warnings.warn(
00000000-0000-0000-0000-000000000000,/azureml-envs/azureml_d778e71475c08b4077c2b
334ee6055be/lib/python3.8/site-packages/sklearn/base.py:288: UserWarning: Trying
to unpickle estimator TfidfTransformer from version 0.22.1 when using version 1.
2.0. This might lead to breaking code or invalid results. Use at your own risk.
For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-
limitations
  warnings.warn(
00000000-0000-0000-0000-000000000000,/azureml-envs/azureml_d778e71475c08b4077c2b
334ee6055be/lib/python3.8/site-packages/sklearn/base.py:288: UserWarning: Trying
to unpickle estimator TfidfVectorizer from version 0.22.1 when using version 1.
2.0. This might lead to breaking code or invalid results. Use at your own risk.
For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-
limitations
  warnings.warn(
2022-12-14 03:33:44,337 | root | INFO | Users's init has completed successfully
2022-12-14 03:33:44,338 | root | INFO | Swaggers are prepared for the following
versions: [2, 3].
2022-12-14 03:33:44,338 | root | INFO | Scoring timeout setting is not found. Us
e default timeout: 3600000 ms
2022-12-14 03:33:44,338 | root | INFO | AML_FLASK_ONE_COMPATIBILITY is set. Patc
hed Flask to ensure compatibility with Flask 1.
2022-12-14 03:34:04,224 | root | ERROR | Encountered Exception: Traceback (most
recent call last):
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/azureml_inference_server_http/server/user_script.py", line 129, in in
voke_run
    run_output = self._wrapped_user_run(**run_parameters, request_headers=dict(r
equest.headers))
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/azureml_inference_server_http/server/user_script.py", line 156, in <l
ambda>
    self._wrapped_user_run = lambda request_headers, **kwargs: self._user_run(**
kwargs)
  File "/var/azureml-app/final_exam_handout_v1/score.py", line 40, in run
    y_pred = lr.predict(data.values)
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/scipy/sparse/_base.py", line 771, in __getattr__
    raise AttributeError(attr + " not found")
AttributeError: values not found

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/azureml_inference_server_http/server/routes.py", line 210, in handle_
score
    timed_result = app.user_script.invoke_run(request, timeout_ms=app.scoring_ti
meout_in_ms)
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/azureml_inference_server_http/server/user_script.py", line 136, in in
```

```
voke_run
    raise UserScriptException(ex) from ex
azureml_inference_server_http.server.user_script.UserScriptException: Caught an
unhandled exception from the user script

2022-12-14 03:34:04,224 | root | INFO | 500
127.0.0.1 - - [14/Dec/2022:03:34:04 +0000] "POST /score HTTP/1.0" 500 16 "-" "DS
VM_workstation/76318178_22.11.11 azureml-sdk-core/1.47.0"
2022-12-14 03:34:06,240 | root | ERROR | Encountered Exception: Traceback (most
recent call last):
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/azureml_inference_server_http/server/user_script.py", line 129, in in
voke_run
    run_output = self._wrapped_user_run(**run_parameters, request_headers=dict(r
equest.headers))
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/azureml_inference_server_http/server/user_script.py", line 156, in <l
ambda>
    self._wrapped_user_run = lambda request_headers, **kwargs: self._user_run(**
kwargs)
  File "/var/azureml-app/final_exam_handout_v1/score.py", line 40, in run
    y_pred = lr.predict(data.values)
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/scipy/sparse/_base.py", line 771, in __getattr__
    raise AttributeError(attr + " not found")
AttributeError: values not found

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/azureml_inference_server_http/server/routes.py", line 210, in handle_
score
    timed_result = app.user_script.invoke_run(request, timeout_ms=app.scoring_ti
meout_in_ms)
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/azureml_inference_server_http/server/user_script.py", line 136, in in
voke_run
    raise UserScriptException(ex) from ex
azureml_inference_server_http.server.user_script.UserScriptException: Caught an
unhandled exception from the user script

2022-12-14 03:34:06,240 | root | INFO | 500
127.0.0.1 - - [14/Dec/2022:03:34:06 +0000] "POST /score HTTP/1.0" 500 16 "-" "DS
VM_workstation/76318178_22.11.11 azureml-sdk-core/1.47.0"
2022-12-14 03:34:10,259 | root | ERROR | Encountered Exception: Traceback (most
recent call last):
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/azureml_inference_server_http/server/user_script.py", line 129, in in
voke_run
    run_output = self._wrapped_user_run(**run_parameters, request_headers=dict(r
equest.headers))
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/azureml_inference_server_http/server/user_script.py", line 156, in <l
ambda>
    self._wrapped_user_run = lambda request_headers, **kwargs: self._user_run(**
kwargs)
  File "/var/azureml-app/final_exam_handout_v1/score.py", line 40, in run
    y_pred = lr.predict(data.values)
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
```

```
e-packages/scipy/sparse/_base.py", line 771, in __getattr__
    raise AttributeError(attr + " not found")
AttributeError: values not found

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/azureml_inference_server_http/server/routes.py", line 210, in handle_
score
    timed_result = app.user_script.invoke_run(request, timeout_ms=app.scoring_ti
meout_in_ms)
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/azureml_inference_server_http/server/user_script.py", line 136, in in
voke_run
    raise UserScriptException(ex) from ex
azureml_inference_server_http.server.user_script.UserScriptException: Caught an
unhandled exception from the user script

2022-12-14 03:34:10,259 | root | INFO | 500
127.0.0.1 - - [14/Dec/2022:03:34:10 +0000] "POST /score HTTP/1.0" 500 16 "-" "DS
VM_workstation/76318178_22.11.11 azureml-sdk-core/1.47.0"
2022-12-14 03:34:18,282 | root | ERROR | Encountered Exception: Traceback (most
recent call last):
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/azureml_inference_server_http/server/user_script.py", line 129, in in
voke_run
    run_output = self._wrapped_user_run(**run_parameters, request_headers=dict(r
equest.headers))
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/azureml_inference_server_http/server/user_script.py", line 156, in <l
ambda>
    self._wrapped_user_run = lambda request_headers, **kwargs: self._user_run(**
kwargs)
  File "/var/azureml-app/final_exam_handout_v1/score.py", line 40, in run
    y_pred = lr.predict(data.values)
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/scipy/sparse/_base.py", line 771, in __getattr__
    raise AttributeError(attr + " not found")
AttributeError: values not found

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/azureml_inference_server_http/server/routes.py", line 210, in handle_
score
    timed_result = app.user_script.invoke_run(request, timeout_ms=app.scoring_ti
meout_in_ms)
  File "/azureml-envs/azureml_d778e71475c08b4077c2b334ee6055be/lib/python3.8/sit
e-packages/azureml_inference_server_http/server/user_script.py", line 136, in in
voke_run
    raise UserScriptException(ex) from ex
azureml_inference_server_http.server.user_script.UserScriptException: Caught an
unhandled exception from the user script

2022-12-14 03:34:18,282 | root | INFO | 500
127.0.0.1 - - [14/Dec/2022:03:34:18 +0000] "POST /score HTTP/1.0" 500 16 "-" "DS
VM_workstation/76318178_22.11.11 azureml-sdk-core/1.47.0"
```

Once your local service is ready, you can now deploy it to a public endpoint. While creating an `AciWebservice`, you can set `cpu_cores=3.8` and `memory_gb=15` to maximize the processing power in the deployment environment.

If Azure complains that the specified specs are too high, you can check the specific upper bounds for your region here.

Once your service has deployed succesfully, save your `scoring_uri` to a text file called `scoring_uri.txt`. This file will be collected by the submitter when you submit your code.

In [33]:

```
## YOUR SCORING_URI CODE HERE ##
# new config variable for ACI deployment (instead of local deployment)
# record useful metadata in the tags parameter
aci_deployment_config = AciWebservice.deploy_configuration(
    cpu_cores=3.8, memory_gb=15, description = 'Public endpoint for Sklearn lr m
    tags={'data' : 'dummy corpus', 'method' : 'tf-idf', 'framework' : 'lr'},
)

# create a new service instance
aci_service4 = Model.deploy(
    ws, "aci-service4",
    [register_lr_model, register_tfidf_model], inference_config,
    aci_deployment_config
)

aci_service4.wait_for_deployment(True)
```

```
/tmp/ipykernel_28629/957597771.py:10: FutureWarning: azureml.core.model:
To leverage new model deployment capabilities, AzureML recommends using CLI/SDK
v2 to deploy models as online endpoint,
please refer to respective documentations
https://docs.microsoft.com/azure/machine-learning/how-to-deploy-managed-online-e
ndpoints /
https://docs.microsoft.com/azure/machine-learning/how-to-deploy-managed-online-e
ndpoint-sdk-v2 /
https://docs.microsoft.com/azure/machine-learning/how-to-attach-kubernetes-anywh
ere
For more information on migration, see https://aka.ms/acimoemigration.
To disable CLI/SDK v1 deprecation warning set AZUREML_LOG_DEPRECATION_WARNING_EN
ABLED to 'False'
  aci_service4 = Model.deploy(
Tips: You can try get_logs(): https://aka.ms/debugimage#dockerlog or local deplo
yment: https://aka.ms/debugimage#debug-locally to debug if deployment takes long
er than 10 minutes.
Running
2022-12-14 04:33:14+00:00 Creating Container Registry if not exists.
2022-12-14 04:33:14+00:00 Registering the environment.
2022-12-14 04:33:17+00:00 Use the existing image.
2022-12-14 04:33:17+00:00 Generating deployment configuration.
2022-12-14 04:33:18+00:00 Submitting deployment to compute.
2022-12-14 04:33:22+00:00 Checking the status of deployment aci-service4..
2022-12-14 04:37:02+00:00 Checking the status of inference endpoint aci-service
4.
Succeeded
ACI service creation operation finished, operation "Succeeded"
```

In [34]:
```
deployed_uri = aci_service4.scoring_uri
print(deployed_uri)
```

```
http://c4a36f04-3940-4519-9be2-56c66c0d1d3b.eastus.azurecontainer.io/score
```

In [35]:
```
with open('scoring_uri.txt', 'w') as f:
    f.write(deployed_uri)
    print("Saved scoring_uri to file!")
```

```
Saved scoring_uri to file!
```

We provide the code to send some example POST requests to your deployment URI as follows. You can run it to check that your deployed model is working. Our grader will run the same code to communicate with your endpoint:

In [36]:
```
def ping_endpoint(json_data, uri = None):
    if uri is None:
        uri = open("scoring_uri.txt", "r").read()
    headers = {'Content-Type' : 'application/json'}
    response = requests.post(uri, json_data, headers = headers)
    return response.status_code, response.text
```

Let's test your model on the provided `example_json`. Check that the status code is 200 and the predicted labels are what you expect.

In [37]:
```
# test the deployed model with 2 samples
status_code, predicted_labels = ping_endpoint(example_json)
print(status_code)
print(predicted_labels)
```

```
200
{"predictions": [5, 5]}
```

Now you can test your model on a 5000-row subset of the training dataset. Again check that the status code is 200 and the predict labels are a list of 5000 integers. If you encounter any error, check your `service.get_logs()`. One common issue is that the request times out; in other words, your deployed model takes too long to process the test data and generate predictions. To address this, you would need to optimize your data processing steps and redeploy them. Our grader will also send 5000 rows of test data to your endpoint in one single request.

Notes:

- Azure ACI enforces a one-minute timeout for handling the input request. If your code takes longer than one minute, the endpoint will return a 502 or 504 error.

In [38]:
```
from sklearn.metrics import accuracy_score

def construct_json_input():
    df_training_subset = read_data().sample(n = 5000, random_state = 200)
    true_labels = df_training_subset["star_rating"]
    training_subset_json = json.dumps({
```

```
        "n_reviews" : len(df_training_subset),
        "reviews" : df_training_subset.drop(columns = "star_rating").to_dict("re
    })
    return bytes(training_subset_json, encoding = "utf8"), true_labels

# test the deployed model with a subset of training data
def test_model_deployment():
    data, true_labels = construct_json_input()
    status_code, response = ping_endpoint(data)
    print("Status code", status_code)
    predicted_labels = json.loads(response)["predictions"]
    print("Model accuracy", accuracy_score(true_labels, predicted_labels))

test_model_deployment()
```

```
Status code 200
Model accuracy 0.695
```

Part F: Follow-up questions

Now that you have completed the coding portions, answer the following questions. Write your answers as if you are composing a technical report for your supervisor; limit your responses to **one or two paragraphs** (5-10 sentences per paragraph) in each question.

You should write your answers in markdown within this notebook, below the **Your response** line for each question below.

Question 1 (6pts): In Part A (Data Cleaning), did you exclude any rows in the dataset? If you did, what are the exclusion criteria and how many rows were excluded? Did you remove or clean any column? Explain in sufficient details so that someone who reads your description could replicate the code from scratch.

Grading rubric:

Rubric item	Score
Describe all the row-based operations, e.g. whether and how you excluded rows	2pts
Describe all the column-based operations, e.g. whether and how you removed or cleaned columns	2pts
Provide sufficiently detailed descriptions that can be used to reconstruct the code	2pts

Your response: I did not exclude any rows because there aren't any rows that contain NaN or 0 values in any columns. However, I removed the columns that are irrelevant features for the review rating prediction, whih are `marketplace`, `customer_id`, `review_id`, `product_id`, `product_parent`, `product_category`, `review_date` using drop function. Since the primary rating predictor is the `review_body`, so I performed the data preprocessing for this column mainly. The data cleaning for this column includes three steps. First is to clean the text by covert the string to lower case, remove all the html tags or any instance of `'s` that is either followed by any whitespace character or at the end of the string, apostrophe character and leading and trailing space. Then, I used the `nltk.word_tokenize` to tokenize the content and used the `string.ascii_letters` to further break tokens. Third, I used the `WordNetLemmatizer()` to lemmatize each token, using the `nltk.pos_tag()` to

specify the part-of-speech parameter. Then, I used the English stopwords to remove any word that belongs to the stopwords. I applied all these functions to the `review_body` column. I then joined all the lemmatized word into a string so we can extract the feature in the later steps. As the `verified_purchase` is a binary column with 'Y' or 'N' string values, so I replace those values into '1' and '0' so that this feature can be used as well in numerical format.

Question 2 (5pts): In Part B (Exploratory Data Analysis), which types of visualization did you employ (e.g., scatter plot, boxplot)? Why did you choose these types? Which columns or relationships between columns did you investigate? Summarize your findings.

Note: Even if you feel tabular explorations are enough or if you use EDA tools, you should still perform at least one visualization analysis, so that you can respond to this question.

Grading rubric:

Rubric item	Score
List all of the graph types used along with their goal (e.g., "I used violinplot to see the distribution of data in column X," instead of simply "I used violinplot")	1pt
List all the columns or relationships between columns that were examined.	2pts
Describe the findings from the exploratory data analysis in layman's terms, so that a reader not familiar with the dataset or your code can still understand them.	2pts

Your response: I used the boxplot to visualize the relationships between `star_rating` and `verified_purchase` to see if there is a correlation between them. I also used the bar plot to visualize the counts for `star_rating` to see the distribution of the labels. I used the heatmap to examine the correlations between each of the columns of `helpful_votes`, `total_votes`, `star_rating`, `vine`, and `verified_purchase`. As expected `total_votes` and `helpful_votes` is strongly correlated, and there is little correlation between `verified_purchase` and `star_rating`. There are 34,662 distinct values for `product_title`. Both `total_votes` and `helpful_votes` are highly skewed that has about 62% and 68% of zeros. `star_rating` is not balanced with value of 5 dominated more than half of the rows. Using `df.describe()`, we can understand the scope of our data by viewing the summary statistics such as min value, max value, average, std. deviation etc and we can know that `helpful_vote` and `total_votes` are strongly skewed with most of the values being 0. Also, there is no missing values. I also used the `ProfileReport` from `pandas_profiling`, which is very useful for exploratory data analysis since it generated reports that summarize each column and the correlation between the features from the data. The label `star_rating` is not balanced so we may need to consider this in the future steps when training the model. There is weak correlation between `verified_purchase` and `star_rating`, so we may include this feature as well when training our model.

Question 3 (6pts): In Part C (Feature Construction), which feature construction method(s) did you employ? Specify the associated parameters (for example, if you used Sklearn's TfIdf, what are your preprocessor, tokenizer and analyzer?).

Grading rubric:

Rubric item	Score
List all of the feature construction steps, along with the state of the dataset before and after each step.	3pts
In each step, specify the input parameters in sufficient details so that the original code can be reconstructed from your descriptions.	3pts

Your response: I used `sklearn` 's `TfidfVectorizer` to construct the `review_body` after the data preprocessing step. After vectorizer, the string type column `review_body` from the dataframe became a 99909x5000 sparse matrix with 2545192 stored elements in Compressed Sparse Row format. `TfidfVectorizer` used analyzer=<method 'split' of 'str' objects>, max_features=5000, tokenizer=<method 'split' of 'str' objects>), preprocess=None. Then, I used the `toarray()` and `tfidf_vectorizer.get_feature_names()` functions to convert the sparse matrix to a pandas dataframe. After that, I concatenate the `tfidf` dataframe with the original dataframe to train the models. Then, I extracted all the tf-idf features along with `verified_purchase` to train the model. I also used the `LabelEncoder()` to transformed the `star_rating` column to transform it from 1-5 to 0-4.

Question 4 (5pts): In Part D (Model Training and Evaluation), list all the models that you explored. How did you select the best model, or the best combination of models (if you used an ensemble method)? If you used train-test split or cross validation, describe them in detail (e.g., the training portion, the type of cross validation, etc.).

Note: For model selection procedure, you should report your selection criteria and hyperparameters considered, among other details. Imagine someone has to work off your description and find the exact same model you report as the best.

Grading rubric:

Rubric item	Score
Name at least two explored models.	2pts
Provide sufficiently detailed descriptions of the model selection procedure that can be used to reconstruct the code.	3pts

Your response: To identify the best model, I used `train_test_split` to split the dataframe with `test_size=0.2` and `random_state=0` . I used `LogisticRegression` with `max_iter=1000` and `RandomForestClassifier` with `n_estimtors=100` , and `svm.SVC` with `kernel='linear'` , and `KNeighborsClassifier` with `n_neighbors=3` . I chose these modesl as the potential models as these are all supervised learning models for classification problem. For every method, I fit the model on the training data and then predict the labels for the testing data. Then, I used the `metrics.accuracy_score` to viewed the training and testing scores of each model. The `LogisticRegression` and `RandomForestClassifier` give two best scores but `RandomForestClassifier` requires longer time to train, so I chose the `LogisticRegression` as the model to train on the whole dataset.

Question 5 (3pts): Did you complete Parts A-E in one pass, as an ordered sequence of steps, or was it necessary to revisit some earlier parts of your solution in order to achieve the desired outcome when working on a later part? If you proceeded sequentially, did you perform any data planning to identify the best steps for each part? If you had to revisit some parts of your solution, which part did you revisit most, and why?

Grading rubric:

Rubric item	Score
Mention whether you went through the parts A-E sequentially.	1pt
Describe your data planning if you proceeded sequentially, OR	2pts
Specify the part that you revisited most recently and the reason for doing so.	2pts

Your response: I went through the parts A-E seuqentially at first by mainly focus on the `review_body` only. After training the model to achieve the desired outcome, I would like to try to incorporate more features. So, I went back to part A to perform the same data cleaning step for `review_headline` and used the `CountVectorizer` on this feature. However, including this does not improve the results to many but only taking more time to process the extra feature. The part I visited the most is the feature extraction as I would like to improve the accuracy results by using more features. I modified the `max_features` for the `TfidfVectorizer` as the `toarray()` function takes a long time if include all the features prodcued by the transformation, but set few features would impact the accuracy scores a lot. In order to find the optimal number of features, I had to experiment with different number of features. I also tried to include `total_votes` or `vine` or `useful_votes` to see the impact of these features, but none of them impact the model training. So, in the end I only used the `TfidfVectorizer` features from the `reivew_body` . .....

Question 6 (4pts): Did you identify any imbalance in the dataset? If you did, describe it here. Which method did you apply to address this imbalance? Describe the state of the dataset after you applied this method; was the identified imbalance mitigated or completely resolved?

Note: even if your final model did not address the data imbalance issue, you should still implement a method to address it for evaluation purpose, so that you can respond to this question.

Grading rubric:

Rubric item	Score
Describe the imbalance in the dataset quantitatively.	2pts
Describe the steps used to mitigate the imbalance and the state of the dataset after applying these steps.	2pts

Your response: The dataset is imbalanced due to the label `star_rating` is imbalanced with 53,558 of value 5, 17,349 of value 4, 14,146 value of 1, 8,483 of value 3 and 6,373 of value 2. In order to mitigate the imbalance, oversampling is implemented for the dataset. By counting each class of the `star_rating` , value 5 has more entries then any of the others, so I increased the

sample size of the other 4 classes to have the same number as the value of 5 using `sample()` function. This results in expanding the dataset to 257,078 number of samples with each class of `star_rating` has 53,558 counts.

Question 7 (5pts): Name two features that are not present in the given dataset but may help improve the accuracy of your models. For each feature, briefly describe how you would go about collecting data for it (e.g., by scraping a particular website, or using a particular API / database). Also explain why you think having that feature would improve your models.

Note: Here you can assume that all public websites can be scraped.

Grading rubric:

Rubric item	Score
Name the two proposed features that are not present in the dataset.	1pt
For each of the two features, name a specific data source (e.g., Amazon listing page) that may contain the data for it.	2pts
Explain how the proposed features would improve your models.	2pts

Your response: One feature can be the number of returns for the product in a month and the other feature can be the number of reviews of a single product. We can get the first feature from the company of the product and the other feature on the Amazon listing page of each product. These two features would improve our models as I believe they are a good indicator to the rating of the product. For the number of return of the product in a month, it can indicate whether people dislike the product or not. If many customers choose to return, as the return rate is high for a product, it might indicate this product would not get a high star rating. For the number of reviews of the single product, we might know whether this product is very good or very bad. Since people do not usually leave reviews on amazon with the review rate around 12%, I assume people would only leave a review if the product is so good that it exceeds the customers' expectations or is so bad that the customers cannot be accepted. From the statistics of these features incorporate in our model, the prediction results might be improved.

Question 8 (2pts): Which parts in the entire pipeline took the most development time? If you were to do a similar assignment in the future, how would you reduce the time taken for these parts of the solution?

Grading rubric:

Rubric item	Score
Describe the time-consuming portions of the pipeline and the reasons for them being time-consuming.	1pt
Describe methods to reduce time in the future assignments.	1pt

Your response: The most time-consuming portions of the pipeline is first the data preprocessing, because it is a very large dataset and it requires a long time to preprocess every `review_body` from the dataset. Second, the most time-consuming is training the model, as many models require very long time for training. In the future assignments, I might discard some

useless rows before data preprocessing so as not to waste time in preprocessing useless contents. For the model training, I should subsample the datasets for training and testing so as to decrease the time for training the model.

Question 9 (4pts): Are there differences in the language used in reviews with high ratings and those with low ratings? You can decide your own threshold for high vs. low and compare the review texts between the two groups. For example, did reviews with high ratings tend to contain more positive words, or have greater lengths?

Grading rubric:

Rubric item	Score
Clearly define “low” and ‘high” reviews.	1pt
Clearly characterize one difference between the language used in low and high reviews in layman’s terms.	2pts
Provide some numerical evidence to back up your observation (e.g., count, mean, std). No detailed tables or plots are required.	1pt

Your response: I defined the "high" reviews as those with 3 to 5 `star_rating` and "low" reviews with 1 and 2 `star_rating` . There are 79,390 total high reviews and total 20,519 low reviews. There are more positive words in the high reviews compared to the low reviews. For the high reviews, there are 31,710 reviews include word "great", 244,27 with word ”well", 23,495 with word "good". The average of each good review is around 308 words and 315 words for bad review. The word "not" include "didnt", "doesnt" appeared in the bad reviews only with about 0.2% frequencies in all the words used in the bad reviews.