1. DATASET GENERATION

In [1]:
```python
#DATASET GENERATION

import pandas as pd
from sklearn.model_selection import train_test_split

#  file path
file_path = r"C:\Users\adity\Desktop\NLP\amazon_reviews_us_Office_Products_v1_00.tsv"

# Load the dataset
df = pd.read_csv(file_path, sep='\t', header=0, on_bad_lines='skip', usecols=['star_rating', 'review_body'])
df.columns = ['rating', 'review']  # Rename columns for consistency

df = df.dropna()  # Drop rows with missing values
df['rating'] = df['rating'].astype(int)

# Balance dataset to have 50K instances per rating score
frames = []  # List to hold data frames to concatenate
for rating in range(1, 6):
    subset = df[df['rating'] == rating]
    if len(subset) >= 50000:
        frames.append(subset.sample(n=50000, random_state=42))
    else:
        frames.append(subset)

# Concatenate all frames to form the balanced dataset
balanced_df = pd.concat(frames)

# ternary labels created
def label_sentiment(row):
    if row['rating'] > 3:
        return 1  # Positive
    elif row['rating'] < 3:
        return 2  # Negative
    else:
        return 3  # Neutral

#labeling function
balanced_df['sentiment'] = balanced_df.apply(label_sentiment, axis=1)

# Splitting the dataset into training and testing sets (80%/20%)
train_df, test_df = train_test_split(balanced_df, test_size=0.2, random_state=42)

#  target variable for training and testing data
X_train = train_df['review']
Y_train = train_df['sentiment']
X_test = test_df['review']
Y_test = test_df['sentiment']

# save for later use
train_df.to_csv('train_dataset.csv', index=False)
test_df.to_csv('test_dataset.csv', index=False)

print("Training Dataset:")
print(train_df.head())

print("\nTesting Dataset:")
print(test_df.head())

# Print shapes of X_train, Y_train, X_test, and Y_test
# to confirm the size and structure of the data splits
print("X_train shape:", X_train.shape)
print("Y_train shape:", Y_train.shape)
print("X_test shape:", X_test.shape)
print("Y_test shape:", Y_test.shape)
```

```
C:\Users\adity\AppData\Local\Temp\ipykernel_11396\2823314214.py:8: DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv(file_path, sep='\t', header=0, on_bad_lines='skip', usecols=['star_rating', 'review_body'])
Training Dataset:
         rating                                             review  sentiment
1735859       2  I'm not sure the pre-owned pen is not a knock-...          2
1924721       2  I ordered two cartridges and only one worked. ...          2
1037272       4  Works well for shipping comic books but is con...          1
710208        2  Does not work with www.stamps.com. Which I und...          2
811349        3  It's good, but the adhesives need to be stickier.          3

Testing Dataset:
         rating                                             review  sentiment
2054240       1  This order was for two color ink cartridges. W...          2
1530392       2  I love Avery products but these sheet protecto...          2
2590754       1  Imagine pressing a key (like Enter or the deci...          2
1539774       3  I like the color but I can't keep it flowing i...          3
1174910       4  So far I like it - it's fast and crisp and was...          1
X_train shape: (200000,)
Y_train shape: (200000,)
X_test shape: (50000,)
Y_test shape: (50000,)
```

2. WORD EMBEDDING

In [2]:
```python
#  2. WORD EMBEDDING

from gensim.models import KeyedVectors

# path to googlecWord2Vec binary file
model_path = r"C:\Users\adity\Desktop\NLP\GoogleNews-vectors-negative300.bin.gz"

# Load pre-trained Word2Vec model
# The model is loaded in a binary format to optimize memory usage
model = KeyedVectors.load_word2vec_format(model_path, binary=True)

# Example words
words = ['king', 'man', 'woman', 'queen', 'excellent', 'outstanding']

# Check similarity between 'king' and 'queen' and between 'excellent' and 'outstanding'
similarity_king_queen = model.similarity('king', 'queen')
similarity_excellent_outstanding = model.similarity('excellent', 'outstanding')

print(f"Similarity between 'king' and 'queen': {similarity_king_queen}")
print(f"Similarity between 'excellent' and 'outstanding': {similarity_excellent_outstanding}")

# Performing a vector algebra operation to find a word that best fits the relationship: King - Man + Woman.
# Example: King - Man + Woman -> ?
result = model.most_similar(positive=['woman', 'king'], negative=['man'], topn=1)
print("King - Man + Woman =", result)
```

```
Similarity between 'king' and 'queen': 0.6510956883430481
Similarity between 'excellent' and 'outstanding': 0.556748628616333
King - Man + Woman = [('queen', 0.7118193507194519)]
```

In [3]:
```python
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess

# Preprocess the reviews from your balanced dataset.
# The dataset 'balanced_df' was prepared in Part 1, ensuring an equal distribution of ratings.
# Each review is converted to a list of tokens (words), with simple preprocessing applied to each.
# 'balanced_df' is my dataset from Part 1
reviews = balanced_df['review'].astype(str).tolist()
tokenized_reviews = [simple_preprocess(review) for review in reviews]

# 'vector_size=300' sets the size of the word vectors.
# 'window=11' defines the maximum distance between the current and predicted word within a sentence.
# 'min_count=10' ignores all words with total frequency lower than this.
# 'workers=4' sets the number of worker threads to use for training.
# TraiNING MY Word2Vec model
my_model = Word2Vec(sentences=tokenized_reviews, vector_size=300, window=11, min_count=10, workers=4)

# semantic similarities with examples
try:
    similarity_king_queen_my_model = my_model.wv.similarity('king', 'queen')
    similarity_excellent_outstanding_my_model = my_model.wv.similarity('excellent', 'outstanding')
    print(f"Similarity between 'king' and 'queen' in my model: {similarity_king_queen_my_model}")
    print(f"Similarity between 'excellent' and 'outstanding' in my model: {similarity_excellent_outstanding_my_model}")
except KeyError as e:
    # This block catches the case where the words 'king', 'queen', 'excellent', or 'outstanding' are not in the vocabulary.
    # This could happen if the words were not frequent enough in the dataset or were removed during preprocessing.
    print(f"Word not in vocabulary: {e}")

# Example: King - Man + Woman in MY model
try:
    result_my_model = my_model.wv.most_similar(positive=['woman', 'king'], negative=['man'], topn=1)
    print("King - Man + Woman in my model =", result_my_model)
except KeyError as e:
    print(f"Word not in vocabulary for my model: {e}")
```

```
Similarity between 'king' and 'queen' in my model: 0.45475128293037415
Similarity between 'excellent' and 'outstanding' in my model: 0.8282253742218018
King - Man + Woman in my model = [('magnum', 0.4322156310081482)]
```

3. Simple models

In [4]:
```python
import numpy as np
from sklearn.linear_model import Perceptron
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from gensim.utils import simple_preprocess

from gensim.models import KeyedVectors

# Loading pre-trained Google News Word2Vec model
google_news_path = r"C:\Users\adity\Desktop\NLP\GoogleNews-vectors-negative300.bin.gz"
# The 'binary=True' parameter indicates that the model file is in a binary format.
google_model = KeyedVectors.load_word2vec_format(google_news_path, binary=True)

# using google_model to compute average Word2Vec vectors


def average_word2vec(model, reviews, vector_size):
    vectors = []
    for review in reviews:
        words = simple_preprocess(review)
        word_vectors = [model[word] for word in words if word in model]
        if word_vectors:
            vectors.append(np.mean(word_vectors, axis=0))
        else:
            vectors.append(np.zeros(vector_size))
    return np.array(vectors)
```

In [5]:
```python
def train_perceptron(X_train, Y_train, X_test, Y_test):
    perceptron = Perceptron()
    # X_train contains the feature vectors for the training set,
    # and Y_train contains the corresponding labels.
    perceptron.fit(X_train, Y_train)
     # X_test contains the feature vectors for the testing set.
    Y_pred_perceptron = perceptron.predict(X_test)
     # It compares the predicted labels (Y_pred_perceptron) against the actual labels (Y_test)
    # and returns the proportion of correctly predicted labels.
    accuracy = accuracy_score(Y_test, Y_pred_perceptron)
    return accuracy
```

In [6]:
```python
def train_svm(X_train, Y_train, X_test, Y_test):
    svm = SVC()
    svm.fit(X_train, Y_train)
    Y_pred_svm = svm.predict(X_test)
    accuracy = accuracy_score(Y_test, Y_pred_svm)
    return accuracy
```

In [ ]:
```python
# 'google_model' is  pre-trained Word2Vec model loaded from word2vec-google-news-300
X_train_google_w2v = average_word2vec(google_model, X_train, 300)
X_test_google_w2v = average_word2vec(google_model, X_test, 300)

accuracy_perceptron_google = train_perceptron(X_train_google_w2v, Y_train, X_test_google_w2v, Y_test)
print("Perceptron Accuracy with Google News Word2Vec:", accuracy_perceptron_google)

accuracy_svm_google = train_svm(X_train_google_w2v, Y_train, X_test_google_w2v, Y_test)
print("SVM Accuracy with Google News Word2Vec:", accuracy_svm_google)



# 'my_model'
X_train_my_w2v = average_word2vec(my_model, X_train, 300)
X_test_my_w2v = average_word2vec(my_model, X_test, 300)

accuracy_perceptron_my = train_perceptron(X_train_my_w2v, Y_train, X_test_my_w2v, Y_test)
print("Perceptron Accuracy with My Word2Vec:", accuracy_perceptron_my)

accuracy_svm_my = train_svm(X_train_my_w2v, Y_train, X_test_my_w2v, Y_test)
print("SVM Accuracy with My Word2Vec:", accuracy_svm_my)
```

4. FEEDFORWARD NEURAL NETWORKS

In [11]:
```python
# 4. FEEDFORWARD NEURAL NETWORKS

!pip install torch torchvision

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
```

```
Requirement already satisfied: torch in c:\users\adity\anaconda3\lib\site-packages (2.2.0)
Requirement already satisfied: torchvision in c:\users\adity\anaconda3\lib\site-packages (0.17.0)
Requirement already satisfied: filelock in c:\users\adity\anaconda3\lib\site-packages (from torch) (3.9.0)
Requirement already satisfied: typing-extensions>=4.8.0 in c:\users\adity\anaconda3\lib\site-packages (from torch) (4.9.0)
Requirement already satisfied: sympy in c:\users\adity\anaconda3\lib\site-packages (from torch) (1.11.1)
Requirement already satisfied: networkx in c:\users\adity\anaconda3\lib\site-packages (from torch) (3.1)
Requirement already satisfied: jinja2 in c:\users\adity\anaconda3\lib\site-packages (from torch) (3.1.2)
Requirement already satisfied: fsspec in c:\users\adity\anaconda3\lib\site-packages (from torch) (2023.4.0)
Requirement already satisfied: numpy in c:\users\adity\anaconda3\lib\site-packages (from torchvision) (1.24.3)
Requirement already satisfied: requests in c:\users\adity\anaconda3\lib\site-packages (from torchvision) (2.31.0)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in c:\users\adity\anaconda3\lib\site-packages (from torchvision) (9.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\adity\anaconda3\lib\site-packages (from jinja2->torch) (2.1.1)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\adity\anaconda3\lib\site-packages (from requests->torchvision) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\adity\anaconda3\lib\site-packages (from requests->torchvision) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\adity\anaconda3\lib\site-packages (from requests->torchvision) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\adity\anaconda3\lib\site-packages (from requests->torchvision) (2023.7.22)
Requirement already satisfied: mpmath>=0.19 in c:\users\adity\anaconda3\lib\site-packages (from sympy->torch) (1.3.0)
```

In [14]:
```python
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess
```

In [34]:
```python
class MLP(nn.Module):
    def __init__(self, input_size, output_size):
        super(MLP, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(input_size, 50),
            nn.ReLU(),
            nn.Linear(50, 10),
            nn.ReLU(),
            nn.Linear(10, output_size)
        )

    def forward(self, x):
        return self.layers(x)
```

In [35]:
```python
# function to calculate the average Word2Vec vectors
def average_word2vec(model, reviews, vector_size):
    vectors = []
    for review in reviews:
        words = simple_preprocess(review)
        word_vectors = [model.wv[word] for word in words if word in model.wv.key_to_index]
        if word_vectors:
            vectors.append(np.mean(word_vectors, axis=0))
        else:
            vectors.append(np.zeros(vector_size))
    return np.array(vectors)
```

In [36]:
```python
print(balanced_df.columns)
```

```
Index(['rating', 'review'], dtype='object')
```

In [37]:
```python
# Splitting the dataset into training and testing sets
train_df, test_df = train_test_split(balanced_df, test_size=0.2, random_state=42)

# Extracting the reviews and ratings
X_train = train_df['review'].tolist()
y_train = train_df['rating']
X_test = test_df['review'].tolist()
y_test = test_df['rating']

train_embeddings = average_word2vec(my_model, X_train, vector_size=300)
test_embeddings = average_word2vec(my_model, X_test, vector_size=300)

y_train_zero_indexed = y_train - 1
y_test_zero_indexed = y_test - 1

# Converting the embeddings and labels into PyTorch tensors
X_train_tensor = torch.tensor(train_embeddings, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.long)
X_test_tensor = torch.tensor(test_embeddings, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.long)

# Creating TensorDatasets and DataLoaders
train_data = TensorDataset(X_train_tensor, y_train_tensor)
test_data = TensorDataset(X_test_tensor, y_test_tensor)

train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64)
```

In [38]:
```python
def train_model(model, optimizer, criterion, train_loader, num_epochs):
    model.train()
    for epoch in range(num_epochs):
        for inputs, labels in train_loader:
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
        print(f'Epoch {epoch+1}/{num_epochs} finished')

def evaluate_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = correct / total
    return accuracy
```

In [39]:
```python
# Preprocess labels
y_train = y_train - 1
y_test = y_test - 1

# DataLoaders
batch_size = 64
train_loader_binary = DataLoader(TensorDataset(X_train_tensor, y_train_binary_tensor), batch_size=batch_size, shuffle=True)
test_loader_binary = DataLoader(TensorDataset(X_test_tensor, y_test_binary_tensor), batch_size=batch_size)
train_loader_ternary = DataLoader(TensorDataset(X_train_tensor, y_train_ternary_tensor), batch_size=batch_size, shuffle=True)
test_loader_ternary = DataLoader(TensorDataset(X_test_tensor, y_test_ternary_tensor), batch_size=batch_size)

# Initializing models, loss function, and optimizers
binary_model = MLP(input_size=300, output_size=2)
ternary_model = MLP(input_size=300, output_size=3)
criterion = nn.CrossEntropyLoss()
optimizer_binary = optim.Adam(binary_model.parameters())
optimizer_ternary = optim.Adam(ternary_model.parameters())

# Train binary model and evaluate
num_epochs = 10  # You can adjust this
train_model(binary_model, optimizer_binary, criterion, train_loader_binary, num_epochs)
binary_accuracy = evaluate_model(binary_model, test_loader_binary)
print(f'Binary classification accuracy: {binary_accuracy:.2f}')

# ternary model
train_model(ternary_model, optimizer_ternary, criterion, train_loader_ternary, num_epochs)
ternary_accuracy = evaluate_model(ternary_model, test_loader_ternary)
print(f'Ternary classification accuracy: {ternary_accuracy:.2f}')
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[39], line 21
     19 # Train and evaluate binary model
     20 num_epochs = 10  # You can adjust this
---> 21 train_model(binary_model, optimizer_binary, criterion, train_loader_binary, num_epochs)
     22 binary_accuracy = evaluate_model(binary_model, test_loader_binary)
     23 print(f'Binary classification accuracy: {binary_accuracy:.2f}')

Cell In[38], line 7, in train_model(model, optimizer, criterion, train_loader, num_epochs)
      5 optimizer.zero_grad()  # Reset gradients to zero for each batch
      6 outputs = model(inputs)  # Forward pass
----> 7 loss = criterion(outputs, labels)  # Compute loss
      8 loss.backward()  # Backward pass
      9 optimizer.step()  # Update weights

File ~\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1511, in Module._wrapped_call_impl(self, *args, **kwargs)
   1509     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1510 else:
-> 1511     return self._call_impl(*args, **kwargs)

File ~\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1520, in Module._call_impl(self, *args, **kwargs)
   1515 # If we don't have any hooks, we want to skip the rest of the logic in
   1516 # this function, and just call forward.
   1517 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hooks or self._forward_pre_hooks
   1518         or _global_backward_pre_hooks or _global_backward_hooks
   1519         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1520     return forward_call(*args, **kwargs)
   1522 try:
   1523     result = None

File ~\anaconda3\Lib\site-packages\torch\nn\modules\loss.py:1179, in CrossEntropyLoss.forward(self, input, target)
   1178 def forward(self, input: Tensor, target: Tensor) -> Tensor:
-> 1179     return F.cross_entropy(input, target, weight=self.weight,
   1180                            ignore_index=self.ignore_index, reduction=self.reduction,
   1181                            label_smoothing=self.label_smoothing)

File ~\anaconda3\Lib\site-packages\torch\nn\functional.py:3059, in cross_entropy(input, target, weight, size_average, ignore_index, reduce, reduction, label_smoothing)
   3057 if size_average is not None or reduce is not None:
   3058     reduction = _Reduction.legacy_get_string(size_average, reduce)
-> 3059 return torch._C._nn.cross_entropy_loss(input, target, weight, _Reduction.get_enum(reduction), ignore_index, label_smoothing)

IndexError: Target 2 is out of bounds.
```

In [27]:
```python
print('Unique y_train labels:', y_train.unique())
print('Unique y_test labels:', y_test.unique())
```

```
Unique y_train labels: [1 3 2 4 0]
Unique y_test labels: [0 1 2 3 4]
```

5. CONVOLUTIONAL NEURAL NETWORKS

In [40]:
```python
#5. Convolutional Neural Networks

# Preparing the data

def prepare_cnn_data(reviews, model, sequence_length=50, vector_size=300):
    data = np.zeros((len(reviews), sequence_length, vector_size))

    for i, review in enumerate(reviews):
        words = review.split()[:sequence_length]
        for j, word in enumerate(words):
            if word in model.wv.key_to_index:
                data[i, j, :] = model.wv[word]

       return data

X_train_cnn = prepare_cnn_data(X_train, my_model, sequence_length=50, vector_size=300)
X_test_cnn = prepare_cnn_data(X_test, my_model, sequence_length=50, vector_size=300)
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
Cell In[40], line 18
     15     return data
     17 # Assuming X_train, X_test, and my_model are already defined
---> 18 X_train_cnn = prepare_cnn_data(X_train, my_model, sequence_length=50, vector_size=300)
     19 X_test_cnn = prepare_cnn_data(X_test, my_model, sequence_length=50, vector_size=300)

Cell In[40], line 6, in prepare_cnn_data(reviews, model, sequence_length, vector_size)
      5 def prepare_cnn_data(reviews, model, sequence_length=50, vector_size=300):
----> 6     data = np.zeros((len(reviews), sequence_length, vector_size))
      8     for i, review in enumerate(reviews):
      9         words = review.split()[:sequence_length]

MemoryError: Unable to allocate 22.4 GiB for an array with shape (200000, 50, 300) and data type float64
```

In [ ]:
```python
# Creating a Simple CNN Model

import torch.nn.functional as F

class SimpleCNN(nn.Module):
    def __init__(self, input_channels, sequence_length, output_size):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv1d(input_channels, 50, kernel_size=5, padding=2)  # Output: 50 channels
        self.conv2 = nn.Conv1d(50, 10, kernel_size=5, padding=2)  # Output: 10 channels
        self.fc = nn.Linear(10 * sequence_length, output_size)

    def forward(self, x):
        x = x.permute(0, 2, 1)
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x
```

In [ ]:
```python
# Training and Evaluating the CNN

# Convert the embeddings and labels into PyTorch tensors for CNN
X_train_tensor_cnn = torch.tensor(X_train_cnn, dtype=torch.float32)
X_test_tensor_cnn = torch.tensor(X_test_cnn, dtype=torch.float32)

train_data_cnn = TensorDataset(X_train_tensor_cnn, y_train_tensor)
test_data_cnn = TensorDataset(X_test_tensor_cnn, y_test_tensor)

train_loader_cnn = DataLoader(train_data_cnn, batch_size=64, shuffle=True)
test_loader_cnn = DataLoader(test_data_cnn, batch_size=64)

# Initialize the CNN
cnn_model = SimpleCNN(input_channels=300, sequence_length=50, output_size=2)  # Use output_size=3 for ternary classification

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(cnn_model.parameters())

# Train
train_model(cnn_model, optimizer, criterion, train_loader_cnn, num_epochs=10)  # Adjust num_epochs as needed

# Evaluate
accuracy = evaluate_model(cnn_model, test_loader_cnn)
print(f'CNN classification accuracy: {accuracy:.2f}')
```