SENTIMENT ANALYSIS FOR

MARKETING

SATHYA.K: 510521104041

PHASE 5: FINAL PROJRCT

INTRODUCTION:

Sentiment analysis is the area which deals with judgments, responses as well as feelings, which is generated from texts, being extensively used in fields like data mining, web mining, and social media analytics because sentiments are the most essential characteristics to judge the human behavior. This particular field is creating ripples in both research and industrial societies. Sentiments can be positive, negative, or neutral or it can contain an arithmetical score articulating the effectiveness of the sentiment. Sentiments can be expressed by calculating the judgment of people on a certain topic, approach, and sensation toward a unit [1], where a unit can be an occurrence, a theme, or even a character. Sentiment analysis and opinion mining are used interchangeably in several cases though there are occurrences where they hold minute dissimilarities among themselves [2]. Sentiment analysis works on discovering opinions, classify the attitude they convey, and ultimately categorize them division-wise.

What is sentiment analysis?

Sentiment analysis, also referred to as opinion mining, is a method to identify and assess sentiments expressed within a text. The primary purpose is to gauge whether the attitude towards a specific topic, product, or service is positive, negative, or neutral. This process utilizes AI and natural language processing

(NLP) to interpret human language and its intricacies, allowing machines to understand and respond to our emotions.

Consider this process akin to mining a wealth of data, identifying hidden nuggets of sentiments in them that can provide actionable business insights. Rather than merely scratching the surface with rudimentary sentiment analysis, businesses can delve deeper using advanced AI techniques.

To illustrate, let's consider three examples:

1. Suppose a customer writes an online review saying, "I love the friendly staff and quick service at this restaurant." Sentiment analysis would categorize this feedback as positive due to the presence of words like 'love,' 'friendly,' and 'quick' in the feedback.
2. On the other hand, a comment such as "The laptop is good, but it overheats too quickly" contains mixed sentiments. While the customer appreciates some aspect of the product ('good'), the negative sentiment is expressed in 'overheats too quickly.'
3. A neutral sentiment could be something like "The book was received on time," which neither praises nor criticizes the product or service.

Through sentiment analysis, businesses can go beyond simple count-based metrics, capturing high-value insights from customer conversations and social media streams. As a result, they can comprehensively understand customer sentiments, improve customer experiences, and ultimately enhance their product or service offerings.

Types of sentiment analysis

Sentiment analysis encompasses a variety of analytical methods that help understand the sentiment or emotional tone behind textual data.

Here are four key types:

1. **Aspect-based sentiment analysis**:

This approach allows us to understand the sentiment related to specific product or service aspects. Instead of merely determining whether the sentiment is positive or negative, it drills down to discover the sentiment about individual

features or aspects. For example, in the case of a restaurant, aspects might include "food quality," "ambiance," or "service speed."

2. **Fine-grained sentiment analysis**:

This technique refines sentiment polarity by distinguishing degrees of sentiment. Instead of merely classifying sentiments as positive, negative, or neutral, it breaks them down into more specific categories such as "very positive," "somewhat positive," "neutral," "somewhat negative," or "very negative." This is particularly useful for detailed analysis of reviews or ratings, offering a granular view of customers' sentiments.

3. **Emotion detection**:

This form of sentiment analysis identifies specific emotions within textual data. Instead of determining whether the sentiment is positive, neutral, or negative, it categorizes the emotions expressed in the text, like joy, surprise, anger, sadness, fear, etc. This offers a deeper understanding of the user's emotional state.

4. **Intent analysis**:

This type aims to understand the intention or goal behind a particular text. By identifying the underlying intent, organizations can gain insights into customer behavior, predict future actions, and adapt their strategies accordingly. It's especially useful in scenarios like customer service, where predicting a customer's behavior can help plan effective responses.

How does AI-based sentiment analysis work?

Sentiment analysis, a subset of AI, employs the techniques of natural language processing and machine learning to automatically categorize a given text as positive, negative, or neutral based on the embedded emotions and opinions. The NLP element converts human language into a form that machines can interpret. This is achieved through syntactic and semantic techniques. Syntactic techniques focus on understanding the structure of a text using methods such as tokenization, lemmatization, and part-of-speech tagging, while semantic techniques aim to comprehend the text's meaning.

After the text is processed using NLP techniques, it's then prepared for categorization using machine learning algorithms. Machine learning algorithms enable machines to identify patterns within data and subsequently predict outcomes. Instead of depending on explicit programming, these algorithms learn from a variety of similar examples known as training data.

To construct a model that classifies text based on sentiment, you need to train it with examples of emotional expressions in the text. Each of these instances needs to be marked with an appropriate category. Many samples for each category are required to enhance your model's accuracy. After being exposed to numerous examples, the model begins to link specific inputs (text) with particular categories. These learned criteria are then used to predict new, unseen content tags.

One noteworthy aspect of sentiment analysis with AI is its ability to tag large data sets simultaneously and in real-time. As the machine learning algorithms grow more sophisticated with experience, sentiment analysis models will improve their proficiency over time as they categorize new data.

Traditionally, companies used surveys and focus groups to understand customers' sentiments toward their products. However, the advent of big data analytics has allowed organizations to delve into larger data volumes, like social media data, to gain a more accurate depiction of customer opinions.

AI-based sentiment analysis predominantly employs two techniques:

1. **Rule-based sentiment analysis:**

   This method uses a set of predefined rules along with a sentiment-labeled dictionary. The system uses NLP techniques to identify words before comparing them against the dictionary. The sentiment score derived at the end considers other factors, such as negations and dependencies. However, this method has limitations, such as its inability to understand sarcasm or the context of words in a sentence, and it requires regular updates and maintenance.

2. **Machine learning-based sentiment analysis:**

   Trains a machine learning model on labeled datasets. After enough training, the algorithm can discern sentiments from new texts without depending on predefined rules, enabling it to detect complexities like sarcasm and synonyms. According to some sources, there's also a hybrid approach that merges the rule-based and machine-learning methods, often producing more accurate results.

Other key AI technologies that are employed in sentiment analysis are as follows. And here is how they work:

- **Machine Learning with manual feature selection:**

  NLP shifted from manually coded rules to machine learning models in the early years. These early models often combined hand-selected features and weights. An engineer could create a series of features – for instance, whether the word "not" appears within a five-word window – and the machine learning model (like logistic regression) would then determine that feature's significance in determining the text's sentiment.

- **Deep learning and word embeddings:**

  The advent of word vector embedding models like Word2Vec and GloVe in 2013, primarily developed by Google researcher Tomáš Mikolov and his team, transformed sentiment analysis. Word embeddings assign a unique vector to every word in a given lexicon. Words with similar meanings or functions are situated close to each other in this vector space, while dissimilar words are located farther apart. These embeddings transform a sentence from plain text into a numerical matrix analogous to an image. This matrix can then be input into a deep neural network, such as a Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), or Long Short-term Memory (LSTM) network. Combining word embeddings with neural networks enables the model to discern context within a sentence, enhancing the robustness of sentiment analysis algorithms to linguistic nuances and variations.

- **Transformers:**

  The transformer-based models represent the current state of the art in sentiment analysis. The most notable transformer model is BERT (Bidirectional Encoder Representations from Transformers). Similar to word vector embeddings, transformer models transform words in a sentence into vectors within a high-dimensional space. However, a key advantage of transformers is their ability to represent a word with different vectors based on its context within the sentence, enabling a more nuanced understanding of sentence sentiment.

Key applications of AI-based sentiment analysis:

Sentiment analysis finds a broad range of applications across various domains. Here is how it is put to use in diverse areas:

- **Unveiling and predicting market trends:**

  Sentiment analysis can dissect vast amounts of market research data, identifying emerging trends and insights into consumer purchasing behaviors. This ability is particularly useful in stock market trading, where decisions often rely on market sentiment. It can help traders understand the sentiments towards a specific stock or sector and make informed trading decisions.

- **Brand perception monitoring**:

  Utilizing sentiment analysis allows businesses to delve into how consumers perceive their products or services. Businesses can provide development teams with substantial data by analyzing product-related discussions and sentiments, driving informed decisions about product improvements or modifications.

- **Scrutinizing public and political sentiments**:

  Sentiment analysis can be a potent tool in political science, allowing analysts to predict election outcomes by analyzing public sentiment towards candidates. A more accurate prediction can be made about the election's outcome by sifting through news articles, social media posts, public opinions, and suggestions.

- **Interpreting customer feedback**:

  Customer feedback holds a wealth of insights. With sentiment analysis, businesses can interpret this data, identifying improvement areas. By extracting sentiment from customer feedback, businesses can shape strategies that enhance customer satisfaction and loyalty.

- **Social media conversations analysis**:

  Social media is a treasure trove of information about brand perception. Sentiment analysis allows businesses to decode these conversations, deriving

meaningful insights about what consumers think about their brand. This can be crucial in shaping future business strategies, marketing campaigns, and customer service improvements.

- **Mitigating employee turnover**:

  By applying sentiment analysis to employee feedback, organizations can understand employee satisfaction levels and uncover potential areas of discontent. It can guide measures to boost employee morale and productivity, leading to a reduction in employee turnover and a happier, more productive workforce.

- **Brand monitoring across platforms**:

  Beyond social media, brand-related discussions occur on numerous platforms, such as blogs, news sites, forums, and product reviews. Sentiment analysis adds valuable context to these discussions, understanding the nuances of customer opinions. This can help assess the impact of a PR crisis on your brand and evaluate the effectiveness of measures taken to manage the situation.

- **Market research and competitor analysis:**

  Sentiment analysis can be a vital asset in market research. Analyzing sentiments around your competition can reveal their strengths and weaknesses, helping you strategize effectively. Also, tracking keywords or hash tags relevant to your industry can help detect market trends and gauge interest around specific topics, providing a competitive advantage.

- **Understanding employee feedback (Voice of employee):**

  Organizations can gain actionable insights by employing sentiment analysis on employee feedback from various sources. It helps uncover how employees feel about different aspects of their job, like work-life balance, compensation, benefits, etc. Such insights can drive initiatives to boost employee engagement, improve communication, and attract new talent, ultimately creating a more positive and productive work environment.

Benefits of AI-based sentiment analysis:

Understanding customer sentiments

Harnessing the power of AI for sentiment analysis can bring businesses closer to understanding their customers' emotions during interactions. By evaluating the tone, vocabulary, and intensity of conversations, companies can detect if customers are delighted, vexed, or disappointed. Equipped with this knowledge, customer service representatives can tailor their responses to reflect customer emotions, cultivating a more empathetic and positive experience.

Boosting response efficiency

Sentiment analysis enhances the efficiency of responses to crucial customer interactions. By gauging the sentiment underlying customer communication, these tools can highlight urgent or significant concerns that demand immediate attention. This knowledge enables businesses to prioritize their responses and address the most pressing matters, such as a flaw in a product, thereby enhancing their response efficiency.

Tailoring customer experience

Sentiment analysis facilitates the personalization of customer experiences by comprehending customer emotions and inclinations, particularly during customer service interactions. Tools available today can discern a customer's interests, likes, and challenges by analyzing the sentiment of customer service interactions. This knowledge enables businesses to customize their responses and suggestions to meet the customer's unique needs, resulting in a highly tailored customer experience.

Increasing customer retention

Sentiment analysis can also be instrumental in bolstering customer retention by identifying and rectifying the root causes of customer dissatisfaction. By examining sentiments in customer communications, businesses can spot recurring

issues and trends that lead to customer attrition. This knowledge allows businesses to adopt proactive measures to enhance customer satisfaction and loyalty.

Use cases of AI-enabled sentiment analysis across industry verticals:

AI-driven sentiment analysis in retail

The retail sector can harness the power of AI-driven sentiment analysis to gain a deeper understanding of customer perceptions about their brand and pinpoint upcoming market trends.

- **Understanding customer sentiments**:

  AI sentiment analysis tools can evaluate customer reviews, social media posts, and other user-generated content to understand customers' feelings about a brand or its products. These insights can help retailers identify areas of success or areas that need improvement.

- **Personalized marketing:**

  By understanding individual customer sentiments, retailers can tailor their marketing and advertising strategies to better resonate with their target audience. This personalized approach can increase engagement and improve overall marketing effectiveness.

Customer service improvement:

AI for sentiment analysis can identify negative sentiments in real time, allowing customer service teams to address complaints or issues promptly. This proactive approach can help improve customer satisfaction and brand perception.

- **Product development**:

Customer sentiment analysis can reveal what features customers like or dislike about a product. This valuable feedback can guide product development teams to modify to meet customer expectations better.

- **Competitive analysis**:

By applying sentiment analysis to public opinions about competitors, retailers can identify strengths and weaknesses in their rivals' offerings. Such insights can help them position their products more effectively in the market.

- **Trend forecasting**:

AI for sentiment analysis can identify consumer attitudes and preferences shifts over time, providing early warnings of changing market trends. Retailers can use this information to stay ahead of the curve, adjust their strategies, and cater to evolving customer needs.

Overall, AI-based sentiment analysis offers a wealth of benefits to retailers, from enhancing customer satisfaction and personalizing marketing to improving products and anticipating market trends. By leveraging these insights, retailers can make more informed decisions, drive growth, and enhance their competitive edge .

AI based sentiment analysis in tourism and hospitality:

The hospitality sector can greatly benefit from using AI for sentiment analysis in many ways. Here's a snapshot of its use cases:

- **Understanding guest feedback**:

Hotels, restaurants, and other hospitality businesses receive massive amounts of feedback through online reviews, social media posts, and direct customer interactions. AI based sentiment analysis can help these businesses efficiently parse through these feedback data to understand customer sentiment, uncovering valuable insights about their services.

- **Enhancing guest experiences**:

Sentiment analysis can reveal specific aspects of the guest experience that are especially delightful or problematic. This information can help hospitality

businesses tailor their services to meet guest expectations better, leading to increased customer satisfaction and loyalty.

- **Real-time service recovery**:

  AI for sentiment analysis can identify negative sentiments in real time, allowing the hospitality business to intervene immediately and resolve the issue before it escalates. This can significantly improve the overall guest experience and prevent potential reputational damage.

- **Strategic decision-making**:

  By gauging public sentiment about different aspects of their offerings, businesses in the hospitality sector can make data-driven decisions regarding their services. Whether revamping a restaurant's menu or redesigning a hotel's rooms, sentiment analysis can provide actionable insights that help enhance business strategy.

- **Competitor analysis**:

  By applying sentiment analysis to reviews of competitor hotels or rest aurants, businesses can identify their own relative strengths and weaknesses. This information can help them position their services more effectively in the market.

- **Trend identification**:

  AI for sentiment analysis can help hospitality businesses identify emerging trends or changing guest preferences. This can be particularly useful in such a dynamic industry, where staying on top of trends can provide a significant competitive advantage.

AI for sentiment analysis in telecommunications:

I-driven sentiment analysis plays a critical role in the telecom industry, providing a range of use cases to improve customer experience, business operations, and strategic decision-making. Here's how it can be employed:

- **Customer experience management**:

The telecom sector often has to manage massive volumes of customer interactions across various channels, including call centers, social media, emails, and more. AI based sentiment analysis can process this data to understand customer sentiment and identify pain points, enabling proactive customer service and improved customer experience.

- **Churn prediction and prevention**:

  Telecom providers can identify dissatisfied customers and predict potential churn by analyzing customer sentiment over time. This information can be leveraged to implement targeted strategies and offers to retain at-risk customers.

- **Service improvement**:

  Sentiment analysis can reveal underlying issues with service quality, network coverage, or pricing causing customer dissatisfaction. Telecom providers can use these insights to improve these areas and enhance customer satisfaction

- **Competitive analysis**:

  Sentiment analysis can be applied to public discussions and reviews about competitors to understand their strengths and weaknesses from the customers' perspective. This can help telecom providers position their services more effectively and identify opportunities for differentiation.

- **Product development**:

  By understanding how customers feel about different features and services, telecom providers can align their product development efforts with customer needs and preferences.

- **Marketing and sales**:

  Sentiment analysis can provide insights into how marketing and sales messages are resonating with the audience. This can inform the development of more effective marketing strategies and sales pitches.

- **Real-time decision-making**:

  In an industry as dynamic as telecom, real-time sentiment analysis can help providers respond quickly to emerging issues or opportunities, leading to better decision-making and agility.

AI-driven sentiment analysis in healthcare:

In the healthcare sector, AI sentiment analysis can be harnessed in a variety of ways to improve patient care, service delivery, and operational efficiency. Here are some potential use cases:

- **Patient feedback analysis**:

  AI-driven sentiment analysis can be used to interpret patient feedback from various sources, such as social media, online reviews, and patient surveys. By identifying positive and negative sentiments, healthcare providers can better understand patient experiences, address concerns, and enhance service quality.

- **Improving patient care**:

  Sentiment analysis can aid in interpreting patient feelings and emotions conveyed during consultations or written in patient records or communication. This can assist healthcare professionals in better understanding a patient's mental and emotional state, potentially leading to more personalized and effective care.

- **Clinical trial monitoring**:

  AI sentiment analysis can track participants' experiences in clinical trials. Analyzing sentiments in patient reports can provide early indications of side effects, efficacy, or adherence issues.

- **Healthcare marketing**:

  Using sentiment analysis, healthcare organizations can understand public perceptions and sentiments about their brand, services, or specific marketing campaigns. This information can guide marketing strategy and communication.

- **Policy making**:

Governmental and regulatory bodies can use sentiment analysis to gauge public sentiment towards health policies or public health issues, aiding in policy formulation and public health interventions.

- **Mental health analysis**:

Sentiment analysis could potentially help monitor patients' mental health by analyzing their written or spoken communication for signs of negative sentiment or distress.

AI-powered sentiment analysis in banking:

AI sentiment analysis plays a significant role in banking by enhancing customer experience, risk management, and brand perception. Here are some potential use cases:

- **Customer service improvement**:

AI in sentiment analysis can analyze customer feedback from various sources such as social media, online reviews, and customer surveys. By identifying and understanding the sentiments, banks can improve their services and address customer complaints more effectively.

- **Risk management:**

Sentiment analysis can assist in early warning signal detection for credit risk management. For instance, negative sentiments from a business's customers can indicate potential financial distress, which can be factored into the bank's risk assessment process.

- **Product and service development**:

By understanding customer sentiments towards various banking products and services, banks can gain valuable insights into what customers appreciate and what they don't. This can guide the development of new products and the refinement of existing services.

- **Brand perception:**

  Sentiment analysis can help banks understand the overall perception of their brand in the marketplace. They can identify positive and negative sentiments, understand trends over time, and compare their sentiment score with competitors .

- **Marketing strategy**:

  By analyzing customer sentiment towards various marketing campaigns, banks can understand which messages resonate with customers and why. This can help guide future marketing strategies.

- **Customer segmentation**:

  Sentiment analysis can categorize customers based on their sentiments. This segmentation can tailor communication and offers to different customer segments, enhancing personalization and customer satisfaction.

- **Fraud detection**:

  While not a direct application, sentiment analysis can indirectly support fraud detection. Unusual negative sentiments or sudden sentiment changes can indicate potential fraudulent activity, triggering further investigation.

How to do sentiment analysis using LSTM?

We will show how to do sentiment analysis on movie reviews using LSTMs. LSTM networks are enhanced versions of RNNs specifically designed to learn sequential data and its long-range dependencies more effectively than traditional RNNs. They find wide applications in areas of deep learning, such as predicting stock trends, recognizing speech, and processing natural language.
We will be using the IMDB movie review dataset that can be downloaded from this link – http://ai.stanford.edu/~amaas/data/sentiment/
Follow the under mentioned steps.

Import and display the data:

Download the data from the given link and load it using the below code to view the data:

```
# read data from text files

with open('aclImdb/test/neg/10000_4.txt', 'r') as f:

reviews = f.read()

with open('aclImdb/test/neg/10000_4.txt', 'r') as f:

labels = f.read()

print(reviews[:50])

print()

print(labels[:26])

Output - This is an example of why the majority of action f

This is an example of why
```

Now convert the data into lowercase and remove the punctuation

```
reviews = reviews.lower()

from string import punctuation

print(punctuation)

all_text = ''.join([c for c in reviews if c not in punctuation])
```

Data processing: Generate a list of reviews

We have consolidated all the text into one large string. Now, our next step is to split this into individual reviews and store them as separate elements in a list, such as [review_1, review_2, review_3… review_n].

```
reviews_split = all_text.split('\n')

print ('Number of reviews :', len(reviews_split))

Output - Number of reviews : 1
```

Tokenization: Establish a dictionary mapping vocabulary to integers

For many Natural Language Processing (NLP) tasks, it's common to construct a dictionary that maps indexes in a way that words appearing more frequently are assigned lower indexes. A popular approach for accomplishing this involves the use of the Counter method from the Collections library.

```
from collections import Counter

all_text2 = ' '.join(reviews_split)

# create a list of words

words = all_text2.split()

# Count all the words using Counter Method

count_words = Counter(words)

total_words = len(words)

sorted_words = count_words.most_common(total_words)
```

To establish a dictionary mapping vocabulary to integers, you would proceed as follows:

```
vocab_to_int = {w:i for i, (w,c) in enumerate(sorted_words)}
```

Here's a minor detail to consider: In this mapping, indexing begins from 0, meaning 'the' would be mapped to 0. However, we will later need to add padding to shorter reviews, and the standard choice for padding is 0. Therefore, it's necessary to start this indexing from 1.

```
vocab_to_int = {w:i+1 for i, (w,c) in enumerate(sorted_words)}
```

Tokenization: Convert words into an encoded form

Up until now, we have crafted a) a list of reviews and b) a dictionary mapping indexes using the vocabulary derived from all our reviews. The objective of these steps was to encode the reviews, replacing the words in our reviews with corresponding integers.

```
reviews_int = []

for review in reviews_split:

r = [vocab_to_int[w] for w in review.split()]

reviews_int.append(r)
```

```
print (reviews_int[0:3])

Output - [[3, 8, 43, 44, 4, 20, 1, 45, 4, 21, 22, 9, 1, 10, 46, 2, 47, 48, 23, 49, 24, 25, 50, 5, 51, 52, 4, 1,
53, 54, 55, 4, 26, 2, 27, 28, 56, 57, 58, 59, 60, 61, 29, 62, 9, 63, 4, 11, 2, 11, 64, 65, 66, 30, 3, 31, 67,
6, 32, 68, 69, 70, 12, 33, 32, 71, 72, 13, 26, 12, 73, 74, 1, 75, 76, 77, 12, 78, 13, 27, 28, 2, 6, 1, 79,
80, 81, 82, 83, 84, 85, 86, 3, 7, 87, 14, 1, 88, 2, 89, 90, 91, 92, 1, 34, 35, 36, 15, 93, 37, 3, 7, 2, 20, 1,
34, 94, 95, 96, 97, 1, 38, 10, 16, 98, 39, 99, 100, 7, 101, 102, 30, 35, 36, 40, 103, 104, 1, 38, 10, 105,
16, 2, 14, 106, 37, 39, 107, 16, 108, 17, 41, 109, 110, 111, 112, 113, 3, 8, 114, 21, 115, 116, 9, 117,
18, 22, 19, 6, 2, 118, 119, 23, 120, 19, 6, 3, 31, 33, 121, 122, 17, 8, 123, 5, 124, 125, 42, 40, 18, 11,
2, 5, 18, 126, 1, 127, 128, 29, 41, 3, 14, 129, 24, 25, 15, 5, 130, 131, 132, 1, 133, 1, 134, 15, 135,
136, 17, 137, 138, 19, 139, 140, 13, 1, 141, 7, 142, 42, 143, 144, 145]]
```

Tokenization: Convert labels into encoded form

This step is straightforward as we have only two output labels. Accordingly, we will designate ‘positive’ with the number 1 and ‘negative’ with the number 0.

```
import numpy as np

labels_split = ['positive', 'negative', 'positive', 'positive', 'negative']

encoded_labels = [1 if label =='positive' else 0 for label in labels_split]

encoded_labels = np.array(encoded_labels)
```

Examine the lengths of reviews

```
import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

reviews_len = [len(x) for x in reviews_int]

pd.Series(reviews_len).hist()

plt.show()

pd.Series(reviews_len).describe()

Output
```

```
count 1.0

mean 232.0

std NaN

min 232.0

25% 232.0

50% 232.0

75% 232.0

max 232.0

dtype: float64
```

The generated plot looks like the following:

Eliminating outliers: Disposing of excessively long or short reviews

```
reviews_int = [ reviews_int[i] for i, l in enumerate(reviews_len) if l>0 ]
```

Padding or truncating the remaining data:

To manage both short and long reviews, we will adjust all our reviews to a specified length, which we will refer to as the Sequence Length. This length corresponds to the number of time steps for the LSTM layer.

For reviews shorter than the sequence length, we will add padding with 0s. Conversely, for reviews longer than the sequence length, we will truncate them to include only the first set of words up to the sequence length.

```
def pad_features(reviews_int, seq_length):

''' Return features of review_ints, where each review is padded with 0's or truncated to the input seq_length.

'''

features = np.zeros((len(reviews_int), seq_length), dtype = int)

for i, review in enumerate(reviews_int):

review_len = len(review)

if review_len <= seq_length: zeroes = list(np.zeros(seq_length-review_len)) new = zeroes+review elif review_len > seq_length:

new = review[0:seq_length]

features[i,:] = np.array(new)

return features
```

Splitting the dataset into training, validation, and test sets

After preprocessing our data into an appropriate format, we will divide it into training, validation, and test sets.
The distribution will be as follows: 80% for training, 10% for validation, and 10% for testing.

```
# Assuming you have defined the reviews_int variable with your data

# and you want to set the sequence length to 100

seq_length = 100

# Call the pad_features function with reviews_int and seq_length

features = pad_features(reviews_int, seq_length)

# The rest of the code remains the same
```

```
split_frac = 0.8

train_x = features[0:int(split_frac*len(reviews_int))]

train_y = encoded_labels[0:int(split_frac*len(reviews_int))]

remaining_x = features[int(split_frac*len(reviews_int)):]

remaining_y = encoded_labels[int(split_frac*len(reviews_int)):]

valid_x = remaining_x[0:int(len(remaining_x)*0.5)]

valid_y = remaining_y[0:int(len(remaining_y)*0.5)]

test_x = remaining_x[int(len(remaining_x)*0.5):]

test_y = remaining_y[int(len(remaining_y)*0.5):]
```

Loading data and organizing it into batches

Once we have separated our data into training, testing, and validation sets, the following step is to construct dataloaders for this data. Rather than using a generator function to batch our data, we will opt to use a TensorDataset.

```
import numpy as np

import torch

from torch.utils.data import DataLoader, TensorDataset

# Assuming you have some data in numpy arrays

train_x = np.random.rand(100, 10) # 100 samples, 10 features

train_y = np.random.randint(0, 2, size=(100,)) # 100 labels (binary)

# Create Tensor datasets

train_data = TensorDataset(torch.from_numpy(train_x), torch.from_numpy(train_y))

# DataLoader

batch_size = 50

train_loader = DataLoader(train_data, shuffle=True, batch_size=batch_size)
```

To procure a single batch of training data for visualization purposes, we will establish a data iterator.

```
# obtain one batch of training data

dataiter = iter(train_loader)

sample_x, sample_y = next(dataiter) # Use `next()` directly on the iter object

print('Sample input size: ', sample_x.size()) # batch_size, seq_length

print('Sample input: \n', sample_x)

print()

print('Sample label size: ', sample_y.size()) # batch_size

print('Sample label: \n', sample_y)
```

```
Sample input size: torch.Size([50, 10])

Sample input:

tensor([[0.7034, 0.4482, 0.8321, 0.6271, 0.6431, 0.6657, 0.8682, 0.8476, 0.3166,

0.4798],

[0.4986, 0.0200, 0.4595, 0.9254, 0.8145, 0.7975, 0.7027, 0.5315, 0.0633,

0.3708],

[0.2233, 0.8519, 0.2751, 0.1726, 0.0131, 0.9168, 0.4826, 0.2859, 0.8347,

0.3571],

[0.4834, 0.6181, 0.8814, 0.6987, 0.2538, 0.8455, 0.0361, 0.4932, 0.1716,

0.6979],

[0.3436, 0.2338, 0.9501, 0.2772, 0.7758, 0.3319, 0.4560, 0.0637, 0.3984,

0.8067],

[0.6528, 0.1932, 0.9580, 0.2130, 0.9449, 0.8794, 0.5506, 0.7223, 0.9259,
```

0.3914],
[0.1691, 0.2396, 0.1201, 0.0938, 0.6786, 0.4733, 0.2956, 0.0286, 0.2427,
0.1807],
[0.4079, 0.6814, 0.9060, 0.3632, 0.1305, 0.5444, 0.7909, 0.1413, 0.2312,
0.9908],
[0.0060, 0.9553, 0.6465, 0.8999, 0.4546, 0.5736, 0.0576, 0.6150, 0.7116,
0.3232],
[0.0087, 0.6969, 0.3452, 0.0889, 0.4847, 0.3626, 0.9049, 0.7747, 0.6826,
0.4671],
[0.3622, 0.2621, 0.0864, 0.4941, 0.5397, 0.5830, 0.5530, 0.5423, 0.5781,
0.7268],
[0.0919, 0.5907, 0.2638, 0.1189, 0.9791, 0.5678, 0.2447, 0.9174, 0.4527,
0.2479],
[0.1109, 0.1358, 0.4870, 0.2571, 0.0105, 0.8599, 0.5062, 0.3309, 0.1940,
0.8816],
[0.8237, 0.7502, 0.0300, 0.6941, 0.7874, 0.8147, 0.7814, 0.5930, 0.7264,
0.2516],
[0.4720, 0.8332, 0.7371, 0.5548, 0.3322, 0.1917, 0.2605, 0.6905, 0.0773,
0.0194],
[0.8260, 0.5284, 0.6566, 0.1853, 0.6731, 0.7663, 0.9669, 0.5206, 0.5350,
0.5815],
[0.0195, 0.9383, 0.4579, 0.4509, 0.5258, 0.4668, 0.0694, 0.7823, 0.5013,

0.0329],
[0.8956, 0.6812, 0.1840, 0.6256, 0.5740, 0.5031, 0.0395, 0.8363, 0.4525,
0.3345],
[0.5260, 0.8632, 0.0893, 0.1557, 0.1420, 0.2395, 0.2133, 0.7120, 0.4225,
0.9701],
[0.6108, 0.6718, 0.9251, 0.4928, 0.2130, 0.7672, 0.9414, 0.6192, 0.8826,
0.1397],
[0.1588, 0.8059, 0.1725, 0.1385, 0.8286, 0.5996, 0.9071, 0.1250, 0.0257,
0.1717],
[0.7239, 0.5421, 0.1405, 0.0718, 0.6385, 0.4447, 0.9896, 0.0021, 0.2930,
0.3223],
[0.2663, 0.8737, 0.4490, 0.8135, 0.3247, 0.9046, 0.9639, 0.8595, 0.9099,
0.2215],
[0.4407, 0.1478, 0.3036, 0.2252, 0.3201, 0.7197, 0.1164, 0.7817, 0.2695,
0.1697],
[0.1724, 0.6860, 0.8862, 0.1776, 0.5827, 0.7069, 0.9616, 0.6420, 0.4005,
0.2195],
[0.9335, 0.1254, 0.6823, 0.2451, 0.7582, 0.2520, 0.9394, 0.6173, 0.9073,
0.5894],
[0.7955, 0.4576, 0.9754, 0.7944, 0.4832, 0.2182, 0.1606, 0.4541, 0.7654,
0.0996],
[0.5673, 0.1580, 0.1877, 0.7124, 0.3510, 0.8210, 0.5903, 0.3984, 0.0172,

0.9169],
[0.8406, 0.3152, 0.4214, 0.8054, 0.3103, 0.8748, 0.5084, 0.7876, 0.2713,
0.9026],
[0.1041, 0.3078, 0.0334, 0.9537, 0.8232, 0.7124, 0.1294, 0.3954, 0.5099,
0.0601],
[0.9727, 0.0047, 0.3879, 0.5295, 0.8541, 0.5677, 0.8425, 0.9426, 0.8628,
0.9001],
[0.5560, 0.8635, 0.4567, 0.5668, 0.3350, 0.5839, 0.0938, 0.0450, 0.7188,
0.7714],
[0.9290, 0.2526, 0.5834, 0.6354, 0.0252, 0.6823, 0.6056, 0.7914, 0.7256,
0.9863],
[0.4679, 0.6593, 0.4330, 0.2957, 0.8998, 0.9524, 0.8147, 0.0711, 0.0199,
0.0072],
[0.6717, 0.0452, 0.5355, 0.0869, 0.9485, 0.9067, 0.9364, 0.6426, 0.4703,
0.1123],
[0.1588, 0.2142, 0.3126, 0.5052, 0.1097, 0.5796, 0.9278, 0.2355, 0.3715,
0.1877],
[0.9141, 0.9462, 0.3081, 0.6432, 0.7723, 0.9893, 0.5752, 0.8356, 0.8120,
0.8874],
[0.5130, 0.8690, 0.8283, 0.6539, 0.0724, 0.3174, 0.7202, 0.2355, 0.2533,
0.1136],
[0.7707, 0.9367, 0.5900, 0.0450, 0.0664, 0.9667, 0.4897, 0.2897, 0.5269,

0.0057],
[0.4037, 0.1897, 0.8003, 0.9956, 0.9417, 0.7785, 0.1825, 0.3463, 0.7151,
0.6947],
[0.6328, 0.0575, 0.2658, 0.9864, 0.7538, 0.7379, 0.8595, 0.4063, 0.2231,
0.8980],
[0.5761, 0.4596, 0.4098, 0.7361, 0.2366, 0.5256, 0.4925, 0.4653, 0.4855,
0.5675],
[0.3936, 0.6751, 0.6700, 0.1433, 0.0736, 0.5637, 0.7160, 0.8280, 0.7033,
0.9736],
[0.9990, 0.1377, 0.2928, 0.8615, 0.1747, 0.0418, 0.0603, 0.3587, 0.6106,
0.2169],
[0.2078, 0.0172, 0.5001, 0.5338, 0.8021, 0.6420, 0.1756, 0.9057, 0.9402,
0.9377],
[0.9235, 0.1980, 0.1743, 0.4629, 0.3510, 0.5150, 0.2793, 0.7308, 0.4083,
0.2104],
[0.2590, 0.6794, 0.2936, 0.8421, 0.0090, 0.7507, 0.0512, 0.6522, 0.8703,
0.3848],
[0.9626, 0.6640, 0.1349, 0.9443, 0.9004, 0.5377, 0.3468, 0.2036, 0.4505,
0.2096],
[0.8717, 0.1118, 0.1710, 0.2745, 0.0250, 0.3887, 0.0661, 0.2466, 0.9903,
0.8031],
[0.4419, 0.4513, 0.9796, 0.0894, 0.0739, 0.5816, 0.8260, 0.9002, 0.4198,

```
0.3433]], dtype=torch.float64)

Sample label size: torch.Size([50])

Sample label:

tensor([1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,

0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,

0, 1])
```

Next we need to feed the data into the LSTM network Establishing the model class

```
import torch.nn as nn

class SentimentLSTM(nn.Module):

"""

The RNN model that will be used to perform Sentiment analysis.

"""

def __init__(self, vocab_size, output_size, embedding_dim, hidden_dim, n_layers, drop_prob=0.5):

"""

Initialize the model by setting up the layers.

"""

super().__init__()

self.output_size = output_size

self.n_layers = n_layers

self.hidden_dim = hidden_dim

# embedding and LSTM layers

self.embedding = nn.Embedding(vocab_size, embedding_dim)
```

```
self.lstm = nn.LSTM(embedding_dim, hidden_dim, n_layers,

dropout=drop_prob, batch_first=True)

# dropout layer

self.dropout = nn.Dropout(0.3)

# linear and sigmoid layers

self.fc = nn.Linear(hidden_dim, output_size)

self.sig = nn.Sigmoid()

def forward(self, x, hidden):

"""

Perform a forward pass of our model on some input and hidden state.

"""

batch_size = x.size(0)

# embeddings and lstm_out

embeds = self.embedding(x)

lstm_out, hidden = self.lstm(embeds, hidden)

# stack up lstm outputs

lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)

# dropout and fully-connected layer

out = self.dropout(lstm_out)

out = self.fc(out)

# sigmoid function

sig_out = self.sig(out)
```

```
# reshape to be batch_size first

sig_out = sig_out.view(batch_size, -1)

sig_out = sig_out[:, -1] # get last batch of labels

# return last sigmoid output and hidden state

return sig_out, hidden

def init_hidden(self, batch_size):

''' Initializes hidden state '''

# Create two new tensors with sizes n_layers x batch_size x hidden_dim,

# initialized to zero, for hidden state and cell state of LSTM

weight = next(self.parameters()).data

if (train_on_gpu):

hidden = (weight.new(self.n_layers, batch_size, self.hidden_dim).zero_().cuda(),

weight.new(self.n_layers, batch_size, self.hidden_dim).zero_().cuda())

else:

hidden = (weight.new(self.n_layers, batch_size, self.hidden_dim).zero_(),

weight.new(self.n_layers, batch_size, self.hidden_dim).zero_())

return hidden
```

Train the network

Instantiate the network

```
# Instantiate the model w/ hyperparams

vocab_size = len(vocab_to_int) + 1 # +1 for the 0 padding

output_size = 1

embedding_dim = 400
```

```
hidden_dim = 256

n_layers = 2

net = SentimentLSTM(vocab_size, output_size, embedding_dim, hidden_dim, n_layers)

print(net)
```

Output

```
SentimentLSTM(

(embedding): Embedding(146, 400)

(lstm): LSTM(400, 256, num_layers=2, batch_first=True, dropout=0.5)

(dropout): Dropout(p=0.3, inplace=False)

(fc): Linear(in_features=256, out_features=1, bias=True)

(sig): Sigmoid()

)
```

Train the loop

```
# loss and optimization functions

lr=0.001

criterion = nn.BCELoss()

optimizer = torch.optim.Adam(net.parameters(), lr=lr)

train_on_gpu = torch.cuda.is_available()

# training params

epochs = 4 # 3-4 is approx where I noticed the validation loss stop decreasing

counter = 0

print_every = 100

clip=5 # gradient clipping
```

```
# move model to GPU, if available

if(train_on_gpu):

net.cuda()

net.train()

# train for some number of epochs

for e in range(epochs):

# initialize hidden state

h = net.init_hidden(batch_size)

# batch loop

for inputs, labels in train_loader:

counter += 1

if(train_on_gpu):

inputs, labels = inputs.cuda(), labels.cuda()

# Creating new variables for the hidden state, otherwise

# we'd backprop through the entire training history

h = tuple([each.data for each in h])

# zero accumulated gradients

net.zero_grad()

# get the output from the model

inputs = inputs.type(torch.LongTensor)

output, h = net(inputs, h)

# calculate the loss and perform backprop
```

```
loss = criterion(output.squeeze(), labels.float())

loss.backward()

# `clip_grad_norm` helps prevent the exploding gradient problem in RNNs / LSTMs.

nn.utils.clip_grad_norm_(net.parameters(), clip)

optimizer.step()

# loss stats

if counter % print_every == 0:

# Get validation loss

val_h = net.init_hidden(batch_size)

val_losses = []

net.eval()

for inputs, labels in valid_loader:

# Creating new variables for the hidden state, otherwise

# we'd backprop through the entire training history

val_h = tuple([each.data for each in val_h])

if(train_on_gpu):

inputs, labels = inputs.cuda(), labels.cuda()

inputs = inputs.type(torch.LongTensor)

output, val_h = net(inputs, val_h)

val_loss = criterion(output.squeeze(), labels.float())

val_losses.append(val_loss.item())

net.train()
```

```
print("Epoch: {}/{}...".format(e+1, epochs),

"Step: {}...".format(counter),

"Loss: {:.6f}...".format(loss.item()),

"Val Loss: {:.6f}".format(np.mean(val_losses)))
```

Testing

On test data

```
test_losses = [] # track loss

num_correct = 0

# init hidden state

h = net.init_hidden(batch_size)

net.eval()

# Convert lists to NumPy arrays

test_x = np.array(test_x)

test_y = np.array(test_y)

# Create Tensor datasets for test data

test_data = TensorDataset(torch.from_numpy(test_x), torch.from_numpy(test_y))

# DataLoader for test data

test_loader = DataLoader(test_data, shuffle=False, batch_size=batch_size)

# Move this line outside the loop to initialize the hidden state only once

val_h = net.init_hidden(1) # Initialize for batch size 1

# iterate over test data

for inputs, labels in test_loader:

# Move this line inside the loop to create a new hidden state for each batch
```

```
h = val_h

h = tuple([each.data for each in h])

if train_on_gpu:

inputs, labels = inputs.cuda(), labels.cuda()

# get predicted outputs

inputs = inputs.type(torch.LongTensor)

output, h = net(inputs, h)

# Calculate loss using BCEWithLogitsLoss

test_loss = criterion(output, labels.float())

test_losses.append(test_loss.item())

# convert output probabilities to predicted class (0 or 1)

pred = torch.round(output.squeeze()) # rounds to the nearest integer

# compare predictions to true label

correct_tensor = pred.eq(labels.float().view_as(pred))

correct = np.squeeze(correct_tensor.numpy()) if not train_on_gpu else
np.squeeze(correct_tensor.cpu().numpy())

num_correct += np.sum(correct)

# -- stats! -- ##

# avg test loss

print("Test loss: {:.3f}".format(np.mean(test_losses)))

# accuracy over all test data

test_acc = num_correct / len(test_loader.dataset)
```

```
print("Test accuracy: {:.3f}".format(test_acc))
```

Generated output

Test loss: 0.695 Test accuracy: 0.000

On user-generated data

Initially, we will define a function called ‘tokenize’ that handles the pre-processing tasks. Subsequently, we will create a ‘predict’ function that delivers the final output after analyzing the user-supplied review.

```
import numpy as np

import torch

from string import punctuation

# Define the SentimentLSTM class and other necessary variables here

# Define the negative review text you want to test

test_review_neg = "This movie was really disappointing. The acting was bad and the plot made no sense."

# Tokenization and preprocessing

def tokenize_review(test_review):

test_review = test_review.lower()

test_text = ''.join([c for c in test_review if c not in punctuation])

test_words = test_text.split()

test_ints = [vocab_to_int[word] for word in test_words if word in vocab_to_int]

return test_ints

# Define the sequence length

seq_length = 200

# Tokenize and preprocess the review

test_ints = tokenize_review(test_review_neg)

# Padding
```

```
features = pad_features([test_ints], seq_length)

# Convert to tensor

feature_tensor = torch.from_numpy(features)

# Model prediction

def predict(net, test_feature_tensor):

net.eval()

batch_size = test_feature_tensor.size(0)

h = net.init_hidden(batch_size)

if train_on_gpu:

test_feature_tensor = test_feature_tensor.cuda()

output, h = net(test_feature_tensor, h)

pred = torch.round(output.squeeze())

return pred.item()

# Predict the sentiment

prediction = predict(net, feature_tensor)

# Print the prediction

if prediction == 1:

print("Positive review detected!")

else:

print("Negative review detected.")
```

Generated output

Negative review detected.

Endnote:

The influence of sentiment analysis, propelled by advanced AI techniques, is already exerting a transformative effect on global customer interactions. Despite its current impact, we have merely scratched the surface of its potential benefits. The thriving landscape of sentiment analysis as a service offers diverse options for businesses seeking to integrate this powerful tool into their operations. While its advantages may vary across industries, there is a consensus that sentiment analysis will play a pivotal role in elevating customer experience and engagement for enterprises in the foreseeable future.

Nevertheless, it's important to recognize that the capabilities of AI do not replace the need for genuine customer comprehension. Rather, they equip us with more refined instruments for understanding customers more profoundly. These insights, facilitated by AI-driven sentiment analysis, empower businesses to not only decipher customer sentiments but also to translate those insights into tangible enhancements. By bridging the gap between technology and empathy, sentiment analysis stands as a cornerstone in the evolution of customer-centric strategies, guiding businesses toward more meaningful connections and enduring success.