Introduction

In the transformative period beginning in 1995, the rapid development of information technology (IT) was mirrored by the emergence of e-commerce platforms, with Amazon at the forefront. As a consequence, Amazon has amassed a vast dataset of user reviews on PC products, which is represented by the 'amazon_reviews_us_PC_v1_00' dataset. This dataset's mere size represents the quintessential challenge posed by big data. The principal issue we seek to resolve is twofold:

Sentiment Analysis

Analyse this vast dataset to glean meaningful insights regarding consumer sentiment during this time period. We seek to comprehend how consumers perceive PC products, assess overall satisfaction, identify prevalent trends, and identify the most frequently discussed aspects of PC products.

Methodology Comparison

Examine the outcomes and efficiencies of traditional data analysis techniques versus big data approaches. In terms of scalability and processing speed, traditional methods may fall short given the size and intricacy of the dataset. By comparing the two methodologies, we hope to emphasise their respective benefits and potential drawbacks.

This issue must be resolved because it not only provides a retrospective comprehension of IT's evolution through consumer sentiment, but also demonstrates the significance and potential of big data methodologies in extracting meaningful insights from massive datasets. The findings will inform the selection of data analysis methodologies for large-scale datasets and provide valuable insights for future product development and market strategies.

Dataset

Description

The "Amazon Customer Reviews" dataset is a comprehensive compilation of product reviews from Amazon's platform. It was obtained from Kaggle and is provided directly by Amazon. The dataset contains a vast array of data fields that capture various aspects of each review, including fundamental identifiers, review content, and associated metadata.

Size and Format

The dataset is presented in TSV (Tab Separated Values) format and is approximately 3.39 GB in size.

Sample Data

marketplace	customer_id	review_id	product_id
US	22873041	R3ARRMDEGED8RD	B00KJWQIIC
US	30088427	RQ28TSA020Y6J	B013ALA9LA

product_parent	product_title	...	review_date
335625766	Plemo 14-Inch Laptop Sleeve Case	...	31/08/2015
671157305	TP-Link OnHub AC1900 Wireless Wi-Fi Router	...	31/08/2015

Relevance

This dataset is highly pertinent to our objective of understanding the sentiment behind PC product evaluations during a period of IT transformation. It provides an exhaustive view of client ratings, opinions, and experiences spanning twenty years. This dataset will provide insights into consumer perceptions, prevalent trends, and the most-discussed aspects of PC products during this era through the analysis of this data.

Contents

Field Name	Description
marketplace	2-letter country code of the marketplace where the review was written.
customer_id	Random identifier for aggregating reviews by the same author.
review_id	Unique identifier for each review.
product_id	Unique identifier for the reviewed product.
product_parent	Identifier for aggregating reviews of the same product.
product_title	Title of the product.
product_category	Broad category for grouping reviews.
star_rating	Review rating (1-5 stars).
helpful_votes	Count of votes deeming the review helpful.
total_votes	Total votes received by the review.
vine	Indicates if the review was part of the Vine program.
verified_purchase	Denotes if the review is for a verified purchase.
review_headline	Title of the review.
review_body	Main text content of the review.
review_date	Date the review was written.

MapReduce Approach

Algorithm Choice

To process and analyse massive datasets, the MapReduce paradigm was selected as the backbone of our strategy due to its demonstrated efficacy with large-scale data. Our methodology was based on three fundamental algorithms, each with a distinct but interconnected function.

Data Extraction

The Data Extraction algorithm was implemented initially. Given the size of our dataset, it was essential that we refine our focus to the most pertinent information. This algorithm was created to systematically comb through the dataset and derive relevant review data. By isolating the'review_body' from the abundance of other data, we ensured that subsequent processes were streamlined and solely focused on the relevant data.

Data Cleaning

After extraction, the Data Cleaning algorithm assumed the spotlight. Raw data, particularly from user-generated content such as reviews, is frequently inconsistent and noisy. This algorithm was designed to remove any anomalies, redundancies, or extraneous information from the extracted reviews. By ensuring the integrity of the data, we established a firm foundation for the subsequent phases of analysis.

Tokenizing and Word Counting

The Tokenizing and Word Counting algorithm was finally implemented. With the cleansed evaluations in hand, the goal shifted to dividing this data into more granular, analyzable units. The evaluations were tokenized into individual terms, and their frequency was tallied. This process not only provided a clear image of the most prevalent words and phrases in the reviews, but also paved the way for additional analyses, such as sentiment analysis and trend identification.

By sequentially employing these three algorithms, we ensured a systematic and exhaustive approach to data processing and analysis, thereby maximising the potential insights derived from the dataset.

Algorithm 1: Data Extraction

The primary objective of the first algorithm is to extract the'review_body' from the dataset. This phase is crucial because it eliminates irrelevant data, ensuring that subsequent steps only process relevant data.

Mapper

In this algorithm, the Mapper is responsible for reading the input dataset and extracting the'review_body' column. The Mapper, for each record in the dataset:

- Splits the record using the tab character as a delimiter.
- Checks if the record has more than 13 fields to ensure the 'review_body' column exists.
- Writes the 'review_body' to the context.

```
import java.io.IOException;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class ReviewMapper extends Mapper<LongWritable, Text, Text, Text> {
   @Override
   public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
      String line = value.toString();
      String[] fields = line.split("\t");

      if (fields.length > 13) {
         String reviewBody = fields[13];
         context.write(new Text(reviewBody), new Text(""));
      }
   }
}
```

Input:

```
US 22873041 R3ARRMDEGED8RD B00KJWQIIC 335625766 Plemo 14-Inch Laptop Sleeve Case Waterproof Fabric Bag for MacBook Air / Laptops / Notebook, Gray PC 5 0 0 N Y Pleasantly surprised I was very surprised at the high quality of the stitching, the sturdiness of the handles and the padding for my laptop. The price is amazingly low and the look is very good. I am quite happy with this purchase. It fit my MacBook Pro perfectly, with a little bit of room to spare. 2015-08-31
```

Output:

```
I was very surprised at the high quality of the stitching, the sturdiness of the handles and the padding for my laptop. The price is amazingly low and the look is very good. I am quite happy with this purchase. It fit my MacBook Pro perfectly, with a little bit of room to spare.
```

Reducer

The Reducer in this algorithm is straightforward. It acts as a pass-through, ensuring that all the 'review_body' values from the Mapper are written to the output.

```
import java.io.IOException;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class ReviewReducer extends Reducer<Text, Text, Text, Text> {

    @Override
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        for (Text value : values) {
            context.write(key, value);
        }
    }
}
```

Input:

```
I was very surprised at the high quality of the stitching, the sturdiness of the handles and the padding for my laptop. The price is amazingly low and the look is very good. I am quite happy with this purchase. It fit my MacBook Pro perfectly, with a little bit of room to spare.
```

Output:

```
I was very surprised at the high quality of the stitching, the sturdiness of the handles and the padding for my laptop. The price is amazingly low and the look is very good. I am quite happy with this purchase. It fit my MacBook Pro perfectly, with a little bit of room to spare.
```

Algorithm 2: Data Cleaning

Before conducting any analysis, it is necessary to preprocess and cleanse the data to ensure its accuracy. This algorithm eliminates stop words, non-printable characters, and other noise from the'review_body' to prepare the data for analysis.

Mapper

The Mapper reads the 'review_body' and performs several cleaning operations:

- Converts the text to lowercase.
- Removes HTML tags, symbols, numbers, and extra spaces.
- Filters out stop words and words with repeated characters.

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

public class CleaningMapper extends Mapper<LongWritable, Text, Text, Text> {

    private Text outputKey = new Text();
    private Text outputValue = new Text();
    private Set<String> stopWords = new HashSet<String>(Arrays.asList(
        "a", "about", "actually", "almost", "also", "although", "always", "am", "an", "and", "any", "are", "as", "at",
        "be", "became", "become", "but", "by", "can", "could", "did", "do", "does", "each", "either", "else", "for",
        "from", "had", "has", "have", "hence", "how", "i", "if", "in", "is", "it", "its", "just", "may", "maybe", "me",
        "might", "mine", "must", "my", "neither", "nor", "not", "of", "oh", "ok", "when", "where", "whereas", "wherever",
        "whenever", "whether", "which", "while", "who", "whom", "whoever", "whose", "why", "will", "with", "within",
        "without", "would", "yes", "yet", "you", "your"
    ));

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        line = line.toLowerCase(); // Convert to lowercase
        line = line.replaceAll("<br />", " "); // Remove HTML tags
        line = line.replaceAll("\\p{Punct}", " "); // Remove symbols
        line = line.replaceAll("\\s+", " "); // Remove extra spaces
        line = line.replaceAll("\\d", ""); // Remove all numbers


        // Remove non-printable characters
        line = line.replaceAll("[^\\x20-\\x7E]", "");
        // Remove stop words and filter out unwanted entries
        StringBuilder cleanedLine = new StringBuilder();
        for (String word : line.split(" ")) {
            if (!stopWords.contains(word)
                && !word.trim().isEmpty()
                && word.trim().length() > 1
                && !word.equals("t")
                && !isRepeatedChars(word)) {
                cleanedLine.append(word).append(" ");
            }
        }

        String finalCleanedLine = cleanedLine.toString().trim();
        if (!finalCleanedLine.isEmpty()) {
            outputKey.set(finalCleanedLine);
            outputValue.set(""); // You can set this to any value or even the same cleaned line if needed
            context.write(outputKey, outputValue);
        }
    }

    private boolean isRepeatedChars(String word) {
        char firstChar = word.charAt(0);
        for (int i = 1; i < word.length(); i++) {
            if (word.charAt(i) != firstChar) {
                return false;
            }
        }
        return true;
    }
}
```

Input

```
I was very surprised at the high quality of the stitching, the sturdiness of the handles and the padding for my laptop. The price is amazingly low and the look is very good. I am quite happy with this purchase. It fit my MacBook Pro perfectly, with a little bit of room to spare.
```

Output

```
was very surprised at the high quality of the stitching the sturdiness of the handles and the padding for my laptop the price is amazingly low and the look is very good am quite happy with this purchase it fit my macbook pro perfectly with little bit of room to spare
```

Reducer

The Reducer in this algorithm is also a pass-through, ensuring that the cleaned 'review_body' values from the Mapper are written to the output.

```
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class CleaningReducer extends Reducer<Text, Text, Text, Text> {
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        for (Text value : values) {
            context.write(key, value);
        }}}
```

Input

```
was very surprised at the high quality of the stitching the sturdiness of the handles and the padding for my laptop the price is amazingly low and the look is very good am quite happy with this purchase it fit my macbook pro perfectly with little bit of room to spare
```

Output

```
was very surprised at the high quality of the stitching the sturdiness of the handles and the padding for my laptop the price is amazingly low and the look is very good am quite happy with this purchase it fit my macbook pro perfectly with little bit of room to spare
```

Algorithm 3: Tokenizing and Word Counting

This algorithm deconstructs the cleansed reviews into individual words and counts the occurrences of each word. This is the basis for numerous text analysis tasks, including sentiment analysis and topic modelling.

Mapper (TokenizerMapper)

The Mapper tokenizes the 'review_body' into individual words. For each word, it emits the word as a key and a count of one as the value.

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }   }}
```

Input

```
was very surprised at the high quality of the stitching the sturdiness of the handles and the padding for my laptop the price is amazingly low and the look is very good am quite happy with this purchase it fit my macbook pro perfectly with little bit of room to spare
```

Output

```
was 1
very 1
surprised 1
at 1
the 1
high 1
quality 1
of 1
the 1
stitching 1
the 1
sturdiness 1
of 1
the 1
handles 1
```

```
and 1
the 1
padding 1
for 1
my 1
laptop 1
the 1
price 1
is 1
amazingly 1
low 1
and 1
the 1
look 1
is 1
```

```
very 1
good 1
am 1
quite 1
happy 1
with 1
this 1
purchase 1
it 1
fit 1
my 1
macbook 1
pro 1
perfectly 1
```

```
with 1
little 1
bit 1
of 1
room 1
to 1
spare 1of 3
padding 1
perfectly 1
price 1
pro 1
purchase 1
quality 1
```

```
quite 1
room 1
spare 1
stitching 1
sturdiness 1
surprised 1
the 7
this 1
to 1
very 2
was 1
with 2
```

Reducer (TokenizerReducer)

The Reducer aggregates the counts for each word. For each word key, it sums up the counts and emits the total count for that word.

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class TokenizerReducer extends Reducer<Text,IntWritable,Text,IntWritable> {
   private IntWritable result = new IntWritable();

   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException
{   int sum = 0;
      for (IntWritable val : values) {
         sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
   }
}
```

Input

```
was    1
very    1
surprised    1
at    1
the    1
high    1
quality 1
of    1
the    1
stitching    1
the    1
sturdiness    1
of    1
the    1
handles 1
```

```
and    1
the    1
padding 1
for    1
my    1
laptop  1
the    1
price    1
is    1
amazingly    1
low    1
and    1
the    1
look    1
is    1
```

```
very    1
good    1
am    1
quite    1
happy    1
with    1
this    1
purchase    1
it    1
fit    1
my    1
macbook 1
pro    1
perfectly    1
```

```
with    1
little  1
bit    1
of    1
room    1
to    1
spare    1of    3
padding 1
perfectly    1
price    1
pro    1
purchase    1
quality 1
```

```
quite    1
room    1
spare    1
stitching    1
sturdiness    1
surprised    1
the    7
this    1
to    1
very    2
was    1
with    2
```

Output

```
am    1
amazingly    1
and    2
at    1
bit    1
fit    1
for    1
good    1
```

```
handles 1
happy    1
high    1
is    2
it    1
laptop  1
little  1
look    1
```

```
low    1
macbook 1
my    2
of    3
padding 1
perfectly    1
price    1
pro    1
```

```
purchase    1
quality 1
quite    1
room    1
spare    1
stitching    1
sturdiness    1
surprised    1
```

```
the    7
this    1
to    1
very    2
was    1
with    2
```

Python Approach

Imported libraries

```
import pandas as pd
import re
import nltk
from nltk.corpus import words, wordnet, names,stopwords
import numpy as np
from nltk.stem import PorterStemmer, WordNetLemmatizer
import unidecode
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

Data Importing and Preprocessing

To ensure the accuracy of the analysis, the data was cleaned by removing non-printable characters, emojis and other non-ASCII characters.

```
def clean_text(text):
    # Remove non-printable characters
    text = re.sub(r'[\x00-\x1F\x7F]', '', text)

    # Remove emojis and other non-ASCII characters
    text = re.sub(r'[^\x00-\x7F]', '', text)

    return text

with open('C:/Users/User/Downloads/archive/part-r-00000 PD WC.txt', 'r', encoding ='utf-8') as file:
    words1 = [line.strip().split('\t') for line in file if line.strip() != '' and len(line.strip()) > 1]

# Create dataframe from the imported file
df = pd.DataFrame(words1)

# Assigning Column Name for the dataframe
df.columns = ['Words','Frequency']
```

Text Normalization

The text was normalised to assure analysis consistency. This required converting to lowercase, removing non-alphanumeric characters, stemming, lemmatization, and reducing repeated character sequences.

```
# Download the necessary corpora if you haven't already
nltk.download('words')
nltk.download('wordnet')
nltk.download('names')
nltk.download('stopwords')

# Get words from the general word list
general_words = set(words.words())

# Get words from WordNet
wordnet_words = set(wordnet.words())

# Get names from the names corpus (both male and female names)
name_words = set(names.words())

stop_words = set(stopwords.words())
# Combine all the word sets
word_set = general_words.union(wordnet_words).union(name_words)

# Get the list of English stop words
stop_words = set(stopwords.words('english'))

# Initialize stemmer and lemmatizer
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()
def normalize_word(word):
    # Convert to lowercase
    word = word.lower()

    # Remove non-alphanumeric characters
    word = re.sub(r'[^a-zA-Z0-9]', '', word)
    # Handle accents and diacritics
    word = unidecode.unidecode(word)

    # Stemming
    word = stemmer.stem(word)

    # Lemmatization
    word = lemmatizer.lemmatize(word)

    # Reduce sequences of repeated characters
    normalized_double = re.sub(r'(.)\1{2,}', r'\1\1', word)
    if normalized_double in word_set:
        return normalized_double
    normalized_single = re.sub(r'(.)\1+', r'\1', word)
    return normalized_single if normalized_single in word_set else normalized_single

# Apply the normalization function to the 'Words' column
df['Words'] = df['Words'].apply(normalize_word)
# Convert 'Frequency' to integer
df['Frequency'] = df['Frequency'].astype(int)
# Filter out stop words
df = df[~df['Words'].isin(stop_words)]

print(df)
```

Sentiment Analysis

Using the TextBlob library, the sentiment of every word was analysed. Each term was categorised as positive, negative, or neutral based on the polarity of its sentiment.

```
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
from textblob import TextBlob
def get_sentiment(text):
    analysis = TextBlob(text)
    # Classify the polarity of the text
    if analysis.sentiment.polarity > 0:
        return 'positive'
    elif analysis.sentiment.polarity == 0:
        return 'neutral'
    else:
        return 'negative'

# Apply the sentiment analysis function to the 'Words' column
df['Sentiment'] = df['Words'].apply(get_sentiment)

print(df)
```

Grouped data

The data was grouped by sentiment and the frequency of each word was summed up.

```
# Group by the 'Words' column and sum the 'Frequency'
df_grouped = df.groupby('Words', as_index=False).agg({'Frequency': 'sum'})
```

Word Cloud Generation

The most prevalent words for each sentiment (positive, negative, neutral) were compiled into a word cloud.

```
# Sort the DataFrame by 'Frequency' in descending order
df_grouped = df_grouped.sort_values(by='Frequency', ascending=False)

for sentiment in ['positive', 'negative', 'neutral']:
    # Select the top 50 most frequent words for each sentiment
    top_words = df_grouped[df_grouped['Sentiment'] == sentiment]

    words = ' '.join(top_words['Words'])
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate(words)

    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title(f'{sentiment.capitalize()} Words')
    plt.show()
```

Output

Word Cloud

Positive Words

Negative Words

Neutral Words

Top 10 Words in all 3 Sentiment Categories

Positive

	Words	Frequency	Sentiment
0	great	2119338	positive
1	good	1516542	positive
2	love	1021799	positive
3	fit	847985	positive
4	much	724791	positive
5	nice	724598	positive
6	new	653396	positive
7	first	559534	positive
8	better	542913	positive
9	light	478419	positive

Negative

	Words	Frequency	Sentiment
0	hard	480487	negative
1	game	424946	negative
2	small	371131	negative
3	long	338219	negative
4	bad	231882	negative
5	black	144147	negative
6	complaint	141277	negative
7	least	126216	negative
8	wrong	121826	negative
9	fail	120718	negative

Neutral

	Words	Frequency	Sentiment
0	thi	6521650	neutral
1	wa	3338908	neutral
2	work	3071313	neutral
3	use	3024896	neutral
4	one	2079623	neutral
5	veri	2035170	neutral
6	case	1655271	neutral
7	like	1633638	neutral
8	would	1495495	neutral
9	get	1494529	neutral

Overall Sentiment Score

The aggregate sentiment score was calculated by multiplying each word's sentiment score by its frequency and then averaging the results.

Top 100 Words from each Group

```
# Split the DataFrame based on the 'Sentiment' column
df_positive = df_grouped[df_grouped['Sentiment'] == 'positive'].sort_values(by='Frequency', ascending=False).reset_index(drop=True)
df_negative = df_grouped[df_grouped['Sentiment'] == 'negative'].sort_values(by='Frequency', ascending=False).reset_index(drop=True)
df_neutral = df_grouped[df_grouped['Sentiment'] == 'neutral'].sort_values(by='Frequency', ascending=False).reset_index(drop=True)

# Map sentiments to numerical values
sentiment_mapping = {'positive': 1, 'negative': -1, 'neutral': 0}
df_grouped['Sentiment_Score'] = df_grouped['Sentiment'].map(sentiment_mapping)


# Filter top 100 words for each sentiment
top_positive = df_grouped[df_grouped['Sentiment'] == 'positive'].head(100)
top_negative = df_grouped[df_grouped['Sentiment'] == 'negative'].head(100)
top_neutral = df_grouped[df_grouped['Sentiment'] == 'neutral'].head(100)

# Compute the sum of sentiment scores for top 100 words of each sentiment
positive_score_sum = (top_positive['Sentiment_Score'] * top_positive['Frequency']).sum()
negative_score_sum = (top_negative['Sentiment_Score'] * top_negative['Frequency']).sum()
neutral_score_sum = (top_neutral['Sentiment_Score'] * top_neutral['Frequency']).sum()

# Compute the average sentiment score
overall_sentiment_score = (positive_score_sum + negative_score_sum + neutral_score_sum)/((top_positive['Frequency']).sum()+(top_negative['Frequency']).sum()+(top_neutral['Frequency']).sum())

print(f"Overall Sentiment Score: {overall_sentiment_score:.2f}")

if overall_sentiment_score > 0:
    print("The overall sentiment is POSITIVE.")
elif overall_sentiment_score < 0:
    print("The overall sentiment is NEGATIVE.")
else:
    print("The overall sentiment is NEUTRAL.")
```

Output

```
Overall Sentiment Score: 0.12
The overall sentiment is POSITIVE.
```

Overall Sentiment Score

```
# Map sentiments to numerical values
sentiment_mapping = {'positive': 1, 'negative': -1, 'neutral': 0}
df_grouped['Sentiment_Score'] = df_grouped['Sentiment'].map(sentiment_mapping)


# Filter dataframe for each sentiment
top_positive = df_grouped[df_grouped['Sentiment'] == 'positive']
top_negative = df_grouped[df_grouped['Sentiment'] == 'negative']
top_neutral = df_grouped[df_grouped['Sentiment'] == 'neutral']

# Compute the sum of sentiment scores for top 100 words of each sentiment
positive_score_sum = (top_positive['Sentiment_Score'] * top_positive['Frequency']).sum()
negative_score_sum = (top_negative['Sentiment_Score'] * top_negative['Frequency']).sum()
neutral_score_sum = (top_neutral['Sentiment_Score'] * top_neutral['Frequency']).sum()

# Compute the average sentiment score
overall_sentiment_score = (positive_score_sum + negative_score_sum + neutral_score_sum)/((top_positive['Frequency']).sum()+(top_negative['Frequency']).sum()+(top_neutral['Frequency']).sum())

print(f"Overall Sentiment Score: {overall_sentiment_score:.2f}")
if overall_sentiment_score > 0:
    print("The overall sentiment is POSITIVE.")
elif overall_sentiment_score < 0:
    print("The overall sentiment is NEGATIVE.")
else:
    print("The overall sentiment is NEUTRAL.")
```

Output

```
Overall Sentiment Score: 0.06
The overall sentiment is POSITIVE.
```

Challenges & Solutions

Large Dataset

Given the large size of the dataset (3.39 GB), it would be inefficient to process it all at once using conventional methods. Using MapReduce allowed for distributed processing, which simplified the endeavour.

Data Cleaning

The dataset may include records with missing or improperly formatted fields. Before processing, the Mapper verifies that each record has more than 13 fields.

Two-Stage Processing

Instead of conducting a word count on the entire dataset in a single step, a two-step process was utilised. This ensured that only pertinent data ('review_body') was processed during the word count phase, thereby increasing efficiency.

Results

The output of the MapReduce job was a tokenized file exported from HDFS, which was then further processed using Python for additional data cleansing, sentimental analysis, word cloud visualisation, and overall sentimental score calculation.

Hadoop MapReduce

Hadoop MapReduce is accountable for data extraction, cleansing, and tokenization on a large (3.39 GB) datafile. The data file that fed Python was only about 5 MB in size.

Python

Additional data cleansing (text normalisation, text deduplication (for example **aaaaaaawwwwwwwwwwsssome** -> awesome), lemmatization, stop word filtration using NLTK), sentimental analysis using TextBlob, word cloud visualisation, and sentimental score calculation.

Reason for Combining Both Strategies

The decision to combine the capabilities of Hadoop's MapReduce and Python's flexible data processing tools was motivated by the unique advantages of each platform, particularly when faced with the challenges of managing massive datasets. The specifics will be discussed in the section devoted to reflection.

Interpretation

The results provide valuable insights into the sentiment and predominant themes of the reviews. The tokenized and cleansed data enable a more targeted analysis of the content, illuminating patterns and sentiments that may have been concealed in the raw data.

Top 10 Words in all 3 Sentiment Categories

The analysis of the most frequently occurring words in each sentiment category provides a thorough comprehension of the sentiments expressed in the reviews.

Positive

With over 2.1 million occurrences, the frequency of the term "great" suggests a predominant positive sentiment among the reviews. It indicates that a considerable number of consumers enjoyed the products or services. Following closely behind is the word "good" with over 1.5 million mentions, further emphasising the positive user experiences of a large number of individuals. The prevalence of more than one million mentions of the word "love" indicates a strong emotional positive response, suggesting not only satisfaction but also a profound appreciation for the products or services. Additional notable mentions such as "fit", "nice", "new", and "better" further emphasise the positive user experiences.

Negative

On the negative spectrum, "hard" stands out with nearly 500,000 occurrences. This may indicate that some products or services were perceived to be difficult to use or comprehend. Intriguingly, "game" arises as the most prevalent negative term, suggesting potential problems

or dissatisfaction with games or gaming products. The term "small" suggests that size discrepancies were a recurrent concern among consumers, possibly implying that certain products did not meet size expectations. Words such as "bad" and "black" as well as "complaint" and "wrong" reveal additional user concerns.

Neutral

In the neutral category, derived versions of "this" and "was" such as "thi" and "wa" dominate the list. Their high frequency indicates that they are frequently used in assessments, although they do not necessarily convey a particular sentiment. With its numerous occurrences, the expression "work" is particularly intriguing. Although neutral, its connotation can vary depending on context; "does not work" has a negative connotation, whereas "works great" is positive. Other neutral terms such as "use", "one", and "like" indicate that consumers are describing their experiences or making comparisons. The term "case" may refer to product cases or particular circumstances, indicating its use as a descriptor in the reviews.

While the evaluations are overwhelmingly positive, the negative comments highlight specific issues that businesses or service providers may wish to address. The neutral words play a crucial role in comprehending the context and nuances of the assessments, despite their lack of emotional connotation.

Sentimental Scores

The sentiment scores quantify the predominant sentiments expressed in the evaluations. These scores are derived from the frequency of each sentiment category's associated terms.

Top 100 Words from each Group

Derived from the top 100 words in each sentiment category, the sentiment score of 0.12 indicates a predominantly positive sentiment. A score above 0 indicates a positive bias, and a score of 0.12, while not excessively high, is indicative of an overall positive sentiment among reviews. This indicates that the majority of the top 100 words in the reviews were positive, indicating a high level of user satisfaction.

Overall

When the entire dataset is considered, the sentiment score is marginally lower, at 0.06, but remains positive. This indicates that when all words, not just the top 100, are considered, the sentiment is somewhat diluted but still positive. This is a positive indicator, indicating that even when contemplating the entire spectrum of reviews, positive sentiments outweigh negative sentiments.

In summary, both the top 100 words and the entire corpus demonstrate a positive sentiment. This is evidence of the users' general contentment. Companies and service providers should not, however, rely on their laurels. They should delve deeply into the negative and neutral sentiments to identify improvement opportunities and ensure future satisfaction levels are even higher.

Visualization

WordCloud

To visually depict the frequency of words in the reviews, a word cloud was created. This visualisation facilitates the identification of the most prevalent words and themes within a dataset.

Positive Word Cloud Interpretation

The central and largest word in the positive word cloud is "great," which indicates its high frequency and significance in the positive reviews. Other terms such as "good," "love," "fit," and "much" are also quite prevalent, indicating that they are frequently used in positive reviews. The presence of these words indicates that a significant number of customers found the products or services to be of high quality, to fit well, and to evoke emotions of love or satisfaction.

Negative Word Cloud Interpretation

The phrases "hard game" and "long fail" are conspicuously displayed in the negative word cloud, suggesting difficulties or dissatisfaction associated with games or prolonged failures. Words such as "black," "poor," "low," "complaint," "haphazard," "distraught," and "weird" are also present, indicating that consumers experienced a variety of negative emotions or problems. The presence of these terms suggests that some consumers found certain aspects of the products or services to be lacking, difficult, or unsatisfactory.

Neutral Word Cloud Interpretation

Work time dominates the neutral word cloud, implying discussions regarding the functionality or duration of products or services. Other terms such as "device," "battery," "year," "headphone," "value," "waste," "day," and "Christmas" are scattered throughout the text, indicating a variety of topics that do not inherently convey strong positive or negative connotations. These terms may refer to general observations, product descriptions, or neutral feedback.

To sum up, the word clouds provide a graphical representation of the most frequently used terms in various emotion categories. The magnitude of each word indicates its frequency, and its position can aid comprehension of the context or terms' relationships. By analysing these word clouds, it is possible to rapidly comprehend the predominant themes and sentiments expressed by customers in their reviews.

Performance

MapReduce

On an AWS EC2 instance, the execution of all three algorithms was completed in 15 minutes. The main bottleneck was the difficulty of becoming conversant with the Hadoop environment and Java.

Python

The Python analysis and visualisation that followed took roughly 20 minutes. When attempting to process the raw TSV data directly, a significant barrier was encountered, resulting in nearly full RAM usage and a 20-minute non-responsive window. This issue was resolved by first tokenizing the data in MapReduce, then analysing and visualising it in Python.

The combination of Hadoop MapReduce for pre-processing and Python for analysis and visualisation proved to be an efficient method for dealing with large datasets. The MapReduce tokenization and cleaning stages made the data manageable for further analysis in Python, yielding insights into the sentiments and themes of the reviews. The performance was satisfactory, with environmental setup and hardware limitations being the primary obstacles that were successfully surmounted.

Reflection

Comparison

Hadoop and Python stand out in the domain of large-scale data processing due to their respective strengths. Together, they offer a comprehensive solution for managing and analysing massive datasets.

MapReduce Approach

Hadoop's MapReduce framework is a beacon of hope for those attempting to deal with massive data volumes. Its architecture is painstakingly designed to distribute data across multiple nodes, ensuring that each chunk is simultaneously processed. This layout is indispensable for navigating enormous datasets, such as the 3,4 GB file in this undertaking. Thanks to Hadoop's distributed architecture, no single machine is overburdened.

In addition, Hadoop's scalability is beyond praiseworthy. As data requirements increase, the system can easily scale up by adding additional nodes to the cluster. This flexibility guarantees that Hadoop is a robust solution regardless of the magnitude of the dataset. Hadoop is distinguished by its defect tolerance. With data replicated across multiple sites, the system is robust. Even if a node fails, processing continues without interruption, ensuring that extensive duties are not interrupted by unanticipated hardware issues. The fact that Hadoop's Distributed File System (HDFS) is adept at storing and managing massive files makes it an ideal choice for the initial phases of data processing.

Python's Non-MapReduce Approach

Python, on the other hand, is a versatile programming language renowned for its extensive collection of libraries and tools designed for data analysis, cleansing, and visualisation. Python is distinguished by its adaptability. The multitude of functions provided by libraries like pandas

and TextBlob simplifies data manipulation and analysis. This extensive arsenal, coupled with Python's intuitive syntax, makes data processing a breeze.

The user-centric design of Python is evident in its usability. Its concise code structure provides solace to developers because it simplifies writing, debugging, and maintenance. In addition, Python's visualisation capabilities, enabled by libraries such as matplotlib and wordcloud, facilitate the construction of compelling data visualisations, thereby enhancing data interpretation and presentation. Python's brilliance, however, has its limits. Directly processing large datasets can be resource-intensive, particularly in terms of RAM, as demonstrated by the 3,4 GB file.

Synergy of Hadoop and Python

By combining the capabilities of Hadoop and Python, this project accomplished a harmonious equilibrium. Hadoop's MapReduce was responsible for the initial data processing, filtering, and tokenization. Python took charge of further data cleansing, in-depth analysis, and visualisation once the data had been refined and condensed. This collaborative approach assured efficient data processing while capitalising on Python's analytical and visualisation strengths, providing a comprehensive solution to the challenges posed by big data.

Learnings

This assignment taught me the significance of choosing the right tool for the task. Python is a powerful and versatile programming language, but it has limitations, particularly when working with large datasets. On the other hand, although MapReduce offers scalability and performance advantages, it comes with its own set of complications. Before selecting an approach in the future, I would devote more time to comprehending the magnitude and requirements of the dataset. Moreover, I realised the significance of iterative testing and troubleshooting, particularly in a distributed environment such as Hadoop.

Future Work

As we reflect on the work accomplished, several avenues for improvement and expansion become apparent. These potential enhancements aim not only to refine the current methodology, but also to introduce new capabilities that can yield more insightful and actionable outcomes.

Improved Data Cleaning

While our current data cleaning process is robust, there is always room for improvement, particularly in light of the rapid advances in natural language processing (NLP). By incorporating sophisticated NLP techniques, we can perform more nuanced data cleansing and preprocessing. This could involve a greater command of idiomatic expressions, slang, and domain-specific jargon that may be prevalent in reviews. Such enhancements can result in more precise tokenization and, consequently, analyses.Sentiment Analysis Enhancements

Use more sophisticated models or deep learning techniques to improve sentiment analysis accuracy.

Sentiment Analysis Enhancements

Enhancements to Sentiment Analysis Our current approach to sentiment analysis provides a foundational understanding of the reviews' tone. However, sentiment analysis is an extensive field, and more advanced models and deep learning techniques are available. By utilising neural networks or transformer-based models such as BERT, we can delve deeper into the context and nuances of evaluations, resulting in a more nuanced and accurate prediction of sentiment.

Integration with Other Big Data Tools

While Hadoop and MapReduce have served us well, the big data ecosystem provides a multitude of other performance-enhancing tools. Integration with Apache Spark, which is renowned for its in-memory processing capabilities, can substantially accelerate data processing tasks. Such integrations can make our system more agile and able to effortlessly manage even larger datasets.

Interactive Dashboard

When data is represented graphically, it speaks volumes. We plan to develop an interactive interface to make our analysis more accessible and actionable for our stakeholders. This platform would offer insights into reviews, sentiments, and trends in real time. With intuitive visualisations and drill-down capabilities, stakeholders can obtain an all-encompassing view of the data and make informed decisions.

Feedback Loop

Since no model is flawless, and there is always an error margin. To continuously improve our sentiment analysis model, we propose implementing a mechanism for feedback. Users or analysts can mark incorrect sentiment predictions and receive feedback with the correct sentiment. This feedback can then be used to retrain and refine the model, enabling it to learn from its errors and increase its accuracy over time.

While our current strategy has yielded valuable insights, the potential outcomes are extensive. By embracing these prospective enhancements, we hope to develop a solution that is more refined, accurate, and actionable in order to meet the evolving requirements for data analysis.

Conclusion

In this report, we set out to address the difficulties associated with analysing a large dataset, specifically the Amazon US Customer Reviews dataset. The primary objective was to identify the distinctions between traditional data processing methods and the big data approach, especially when utilising the MapReduce paradigm.

The significance of the issue cannot be overstated. With the exponential development of data in the digital age, data processing techniques must be efficient and scalable. While traditional methods are effective for lesser datasets, they fail when confronted with the sheer volume and complexity of big data. Using a two-stage MapReduce process, our solution not only effectively filtered out extraneous data, but also accelerated and simplified subsequent processing.

By contrasting the MapReduce methodology with Python, we were able to emphasise the merits and limitations of each. Python offers flexibility and a rich library ecosystem, particularly for data analysis and visualisation, whereas MapReduce excels at handling large datasets by distributing the workload. However, its efficacy is constrained by local hardware limitations, which became evident when the 3.4 GB dataset was processed.

After processing with MapReduce and Python, the outcomes were incisive data visualisations and sentiment analysis. These preliminary results pave the way for additional analyses and potential applications in business intelligence, consumer feedback processing, and market research.

This assignment highlighted the significance of selecting the appropriate tools and methodologies for big data challenges. Combining MapReduce for data processing with Python for analysis and visualisation proved to be a potent combination, offering both scalability and depth. The problem of processing and analysing large datasets efficiently is urgent, and our solution demonstrates a viable strategy for addressing it.

References

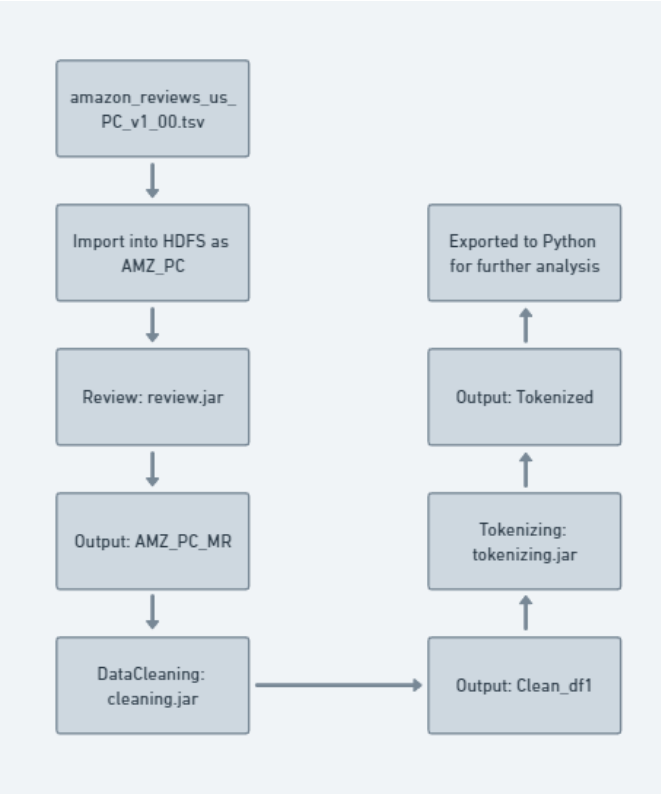
Joshi, A. (n.d.). Amazon-Reviews-Dataset-Analysis-MapReduce. GitHub. Retrieved from https://github.com/joshi-aditya/Amazon-Reviews-Dataset-Analysis-MapReduce

Kaggle. (2021). Amazon US Customer Reviews Dataset. Retrieved from https://www.kaggle.com/datasets/cynthiarempel/amazon-us-customer-reviews-dataset

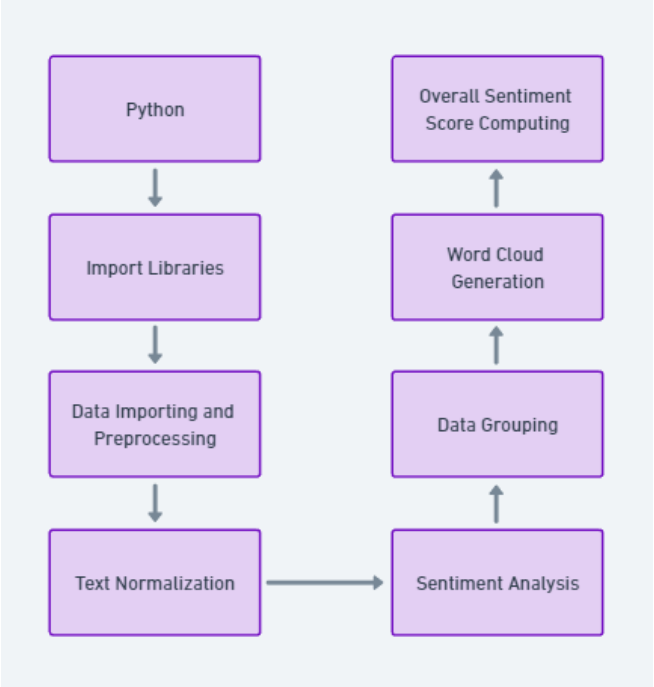
Hadoop Apache Project. (2021). Hadoop MapReduce Tutorial. Retrieved from https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

Appendix

Sample workflow in Hadoop before entering Python



Python workflow



Link to access the files on Github:

https://github.com/chiaazj/IST3134