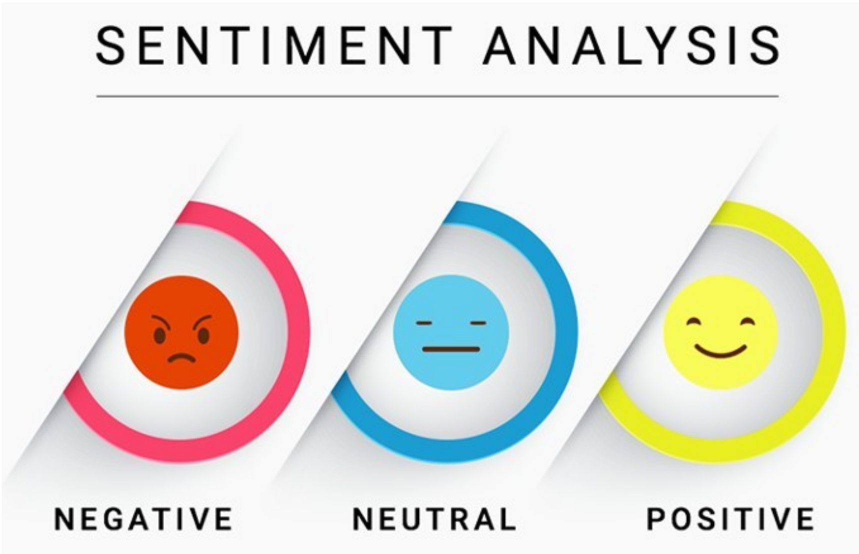

SENTIMENT ANALYSIS
NEGATIVE
NEUTRAL
POSITIVE

Out[1]:

In depth series 1: SENTIMENT ANALYSIS, why and how, EDA and solutions with Transformers

In this study, I explained Sentiment Analysis in detail.

I chose a sample dataset for Sentiment Analysis and embodied the subject I explained on a real example.

Then I made a detailed analysis on the dataset and visualized it.

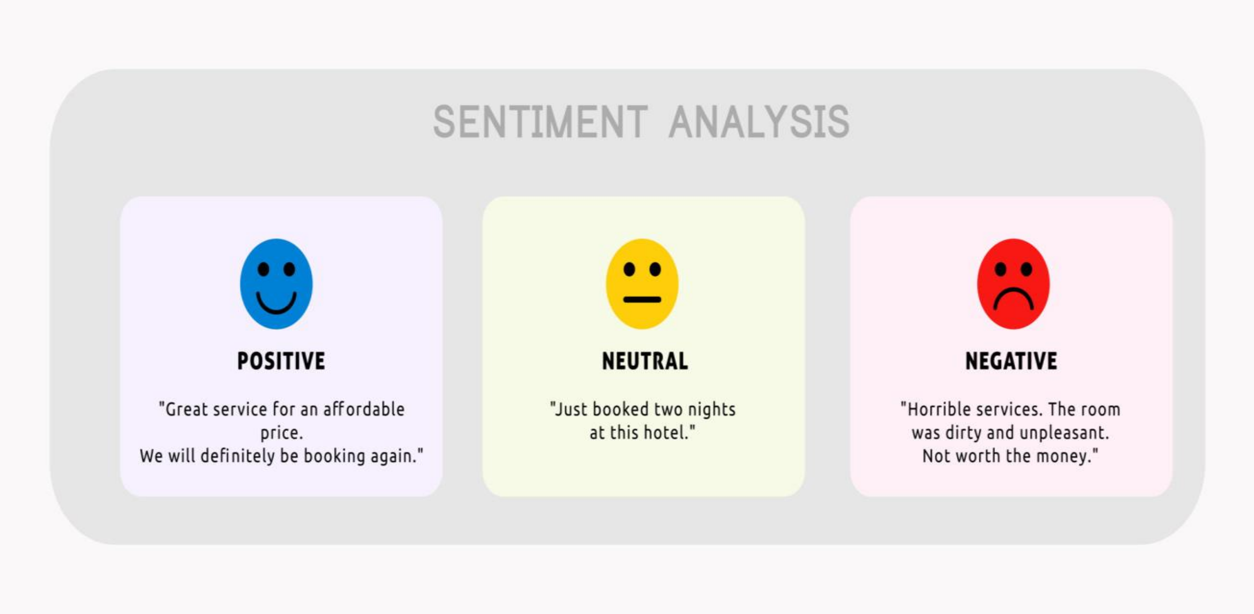
After preprocessing the data, I tried to complete the Sentimet Analysis task with state-of-the-art models.

I analyzed the results of this model and interpreted its outputs.

I have indicated the sources I used while doing this study at the end of the notebook. Thank you to everyone who contributed to this field :).

Table of Contents

1. SENTIMENT ANALYSIS

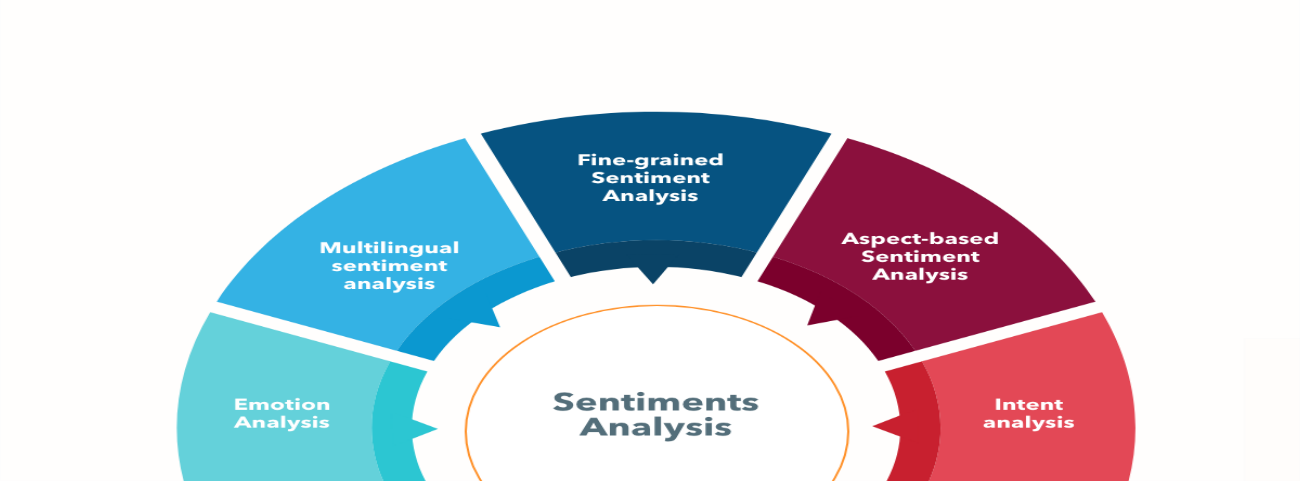


source = https://d3caycb064h6u1.cloudfront.net/wp-content/uploads/2021/06/sentimentanalysishotelgeneric-2048x803-1.jpg

Sentiment analysis (or opinion mining) is a natural language processing (NLP) technique used to determine whether data is positive, negative or neutral. Sentiment analysis is often performed on textual data to help businesses monitor brand and product sentiment in customer feedback, and understand customer needs.

Sentiment analysis helps data analysts within large enterprises gauge public opinion, conduct nuanced market research, monitor brand and product reputation, and understand customer experiences. In addition, companies often develop sentiment analysis systems for customer experience management, social media monitoring, or workforce analytics platform to about their own customers.

Types of Sentiment Analysis

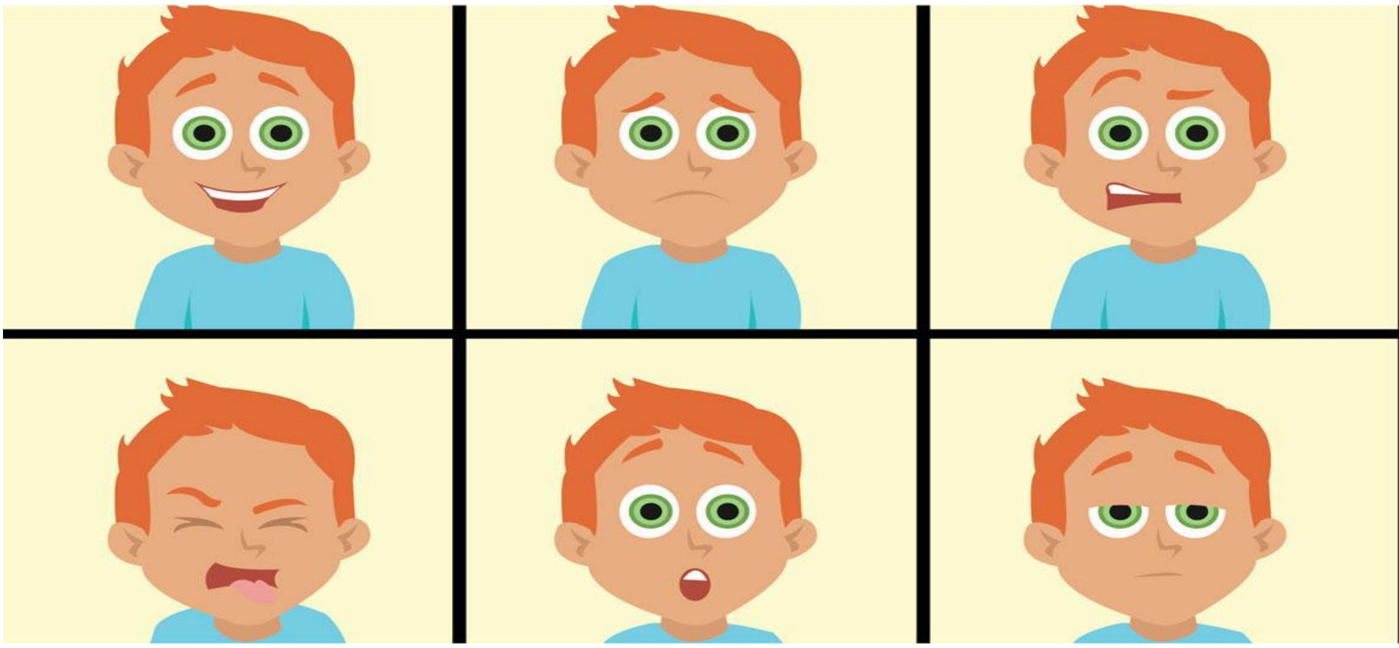


source = https://mobcoder.com/blog/sentimental-analysis-how-the-phenomenon-changing-the-dynamics-of-brand-monitoring/

Sentiment analysis is aimed at determining the general emotional state of a text. One of these cases focuses on the polarity of a text (positive, negative, neutral) but it also goes beyond polarity to detect specific feelings and emotions (angry, happy, sad, etc), urgency (urgent, not urgent) and even intentions (interested v. not interested).

Let's explain them in more detail

Emotion Analysis



source = https://kids.frontiersin.org/articles/10.3389/frym.2018.00015

The type of emotion analysis in which emotion types(happiness, frustration, anger, and sadness) are classified is called **emotion detection.**

There are some difficulties with this classification. Users can express their feelings with many different words. They can use a word with a bad meaning for happiness. The most difficult examples of classification models here are; For example, the sentence "I connect to customer service too late, it's killing me" is a negative sentence, while the sentence "you are killing me" is positive.

Multilingual Sentiment Analysis

It is the version of Sentiment Analysis systems that provides multi-language support. What is mentioned here is to do sentiment analysis in more than one language.

I usually have two suggestions for this:

My first suggestion is to detect the language of the text with the language classifier and run a sentimen analysis model suitable for this language. The second method is to develop a Multilingual language model and finetune this model and make the model work in many languages.

Graded Sentiment Analysis

source = https://i.pinimg.com/originals/5b/7d/62/5b7d62fb62b03b8142b402cb85644865.png

If the precision of the mood is important, the categories can be further elaborated. A broader classification can be made, not just positive and negative:

- Very positive
- Positive
- Neutral
- Negative
- Very negative

This classification is often used in reviews and reviews where 5 stars are awarded.

- Very Positive = 5 stars
- Very Negative = 1 star

Aspect-based Sentiment Analysis

source = https://www.surveysensum.com/wp-content/uploads/2020/02/SENTIMENT-09-1.png

Generally, when analyzing the emotions of the texts, the focus is on determining whether the comment/opinion is positive or negative. But we do not focus on what is positive or negative in this text.

To put it more clearly, in the expression "I did not like the product at all, the size is too small", the user is not satisfied with the product and complains about its dimensions. In a normal sentiment analysis, this sentence is classified as negative, but in **aspect-based sentiment analysis**, the "the size is too small" part is also focused on.

Intent Analysis

Intent analysis focuses on what the user wants to do. Understanding what the user wants to do will allow us to better guide him.

For example, being able to understand that a customer browsing an e-commerce site has a shopping intention also allows us to offer him the right products.

One of the most used areas is the smart assistant systems in the applications. It allows us to direct users to the right places within the application in line with their requests and we can offer a better application experience to the user.

Why Is Sentiment Analysis Important?

source = https://brand24.com/

People now share their comments/emotions on social media, e-commerce sites and many other sites. A lot of data is created on these platforms.

Often brands want to know what they are talking about. Brands/companies make great efforts to quickly identify their customers' expectations and provide them with the right service.It allows their customers to learn what makes them happy or disappointed so they can tailor products and services to their customers' needs. In addition, brands want to observe the impact of their advertisements on users.

For these reasons, Sentiment analysis is becoming more important every day.

The overall benefits of sentiment analysis include:

Sorting Data at Scale

Users make a lot of comments about brands, it is almost impossible to process them manually. Sentiment analysis enables businesses to automatically classify large amounts of raw data.

Real-Time Analysis

Companies can learn the wishes of their customers by analyzing the social media comments about you in real time. They can identify the angry customer and ensure his satisfaction.

Discovering New Marketing Strategies

With more data and information gathered through sentiment analysis, the organizations could develop an effective marketing strategy.

The outcome from the strategies can be measured from the customers’ positive or negative key messages.

By observing the customers' conversations on their social media and detect the specific key messages related to your brand, specific marketing campaigns can be designed for the target consumers.

How Does Sentiment Analysis Work?

source = https://monkeylearn.com/sentiment-analysis/

Sentiment analysis works to automatically determine emotional tone thanks to natural language processing (NLP), rule-based methods, and machine learning algorithms.

There are different ways we can do sentiment analysis, depending on how much data you need to analyze, how accurate your model needs to be, and how many resources you have.

We will talk about some of them below.

Sentiment analysis algorithms fall into one of three buckets:

- **Rule-based:** these systems automatically perform sentiment analysis based on a set of manually crafted rules.
- **Automatic:** systems rely on machine learning techniques to learn from data.

Rule-based Approaches

Usually, a rule-based system tries to help determine the subjectivity of the sentence, the polarity, or the subject matter of an idea. The most used tool here is "regex".

These rules usually include the following two NLP techniques:

- Stemming, tokenization, part-of-speech tagging and parsing.
- Lexicons (i.e. lists of words and expressions).

The working mechanism of these systems is briefly as follows;

1. Build a list of polarized words (e.g. bad-good, worst-best, ugly-beautiful etc). You can find them as open source
2. The ratio of positive and positive words in a sentence

Rule-based approaches are now obsolete, not used as much as they used to be. Rule-based approaches fail to detect ironies, not exactly how users are feeling. For this reason, automated approaches are gaining more importance now.

Automatic Approaches

These systems don't rely on manually crafted rules, but on machine learning techniques, such as classification. Classification, which is used for sentiment analysis, is an automatic system that needs to be fed sample text before returning a category, e.g. positive, negative, or neutral.

Here's how a machine learning classifier can be implemented:

Classification Algorithms

The classification step usually involves a statistical model like Naïve Bayes, Logistic Regression, Support Vector Machines, or Neural Networks:

- **Naïve Bayes:** are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features (see Bayes classifier).
- **Linear Regression:** is a linear approach for modelling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables).
- **Support Vector Machines(SVM):** is a supervised machine learning algorithm that can be used for classification or regression problems. However, it is mostly used in classification problems. Support Vector Machine is a boundary that best separates two classes (hyperplane/line)
- **Deep Learning:** (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.

We can explain the sentiment analysis in general like this. Now we have determined a data for how we will apply it next, and we will spread visualizations on that data and train models.

2. EDA

Information of the Data

Hotels play a crucial role in traveling and with the increased access to information new pathways of selecting the best ones emerged. With this dataset, consisting of 20k reviews crawled from Tripadvisor, you can explore what makes a great hotel and maybe even use this model in your travels!

How to use

- Predict Review Rating
- Topic Modeling on Reviews
- Explore key aspects that make hotels good or bad

Information of the Problem

Customer satisfaction is very important for the service industry. For this reason, it is necessary to determine the emotional state of the customer's thoughts. We need to classify the user's emotion in our hotel reviews data.

Imports

n [2]:

```
import pandas as pd

from wordcloud import WordCloud
```

```
import seaborn as sns
import re
import string
from collections import Counter, defaultdict

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objects as go
from plotly.offline import plot

import matplotlib.gridspec as gridspec
from matplotlib.ticker import MaxNLocator
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
```

In [3]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [4]:

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stopWords_nltk = set(stopwords.words('english'))
```

unfold_lessHide output

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Helper Functions

In [5]:

```
import re
from typing import Union, List

class CleanText():
    """ clearing text except digits () . , word character """

    def __init__(self, clean_pattern = r"[^A-ZĞÜŞİÖÇIa-zğüı'şöç0-9.\"',()]"):
        self.clean_pattern =clean_pattern

    def __call__(self, text: Union[str, list]) -> List[List[str]]:

        if isinstance(text, str):
            docs = [[text]]

        if isinstance(text, list):
            docs = text

        text = [[re.sub(self.clean_pattern, " ", sent) for sent in sents] for sents in docs]

        return text
```

```
def remove_emoji(data):
    emoj = re.compile("["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
        u"\U00002500-\U00002BEF"
        u"\U00002702-\U000027B0"
        u"\U00002702-\U000027B0"
        u"\U000024C2-\U0001F251"
        u"\U0001f926-\U0001f937"
        u"\U00010000-\U0010ffff"
        u"\u2640-\u2642"
        u"\u2600-\u2B55"
        u"\u200d"
        u"\u23cf"
        u"\u23e9"
        u"\u231a"
        u"\ufe0f"  # dingbats
        u"\u3030"
                      "]+", re.UNICODE)
    return re.sub(emoj, '', data)


def tokenize(text):
    """ basic tokenize method with word character, non word character and digits """
    text = re.sub(r" +", " ", str(text))
```

```
    text = re.split(r"(\d+|[a-zA-Zğüşıöç ĞÜŞİÖÇ]+|\W)", text)
    text = list(filter(lambda x: x != '' and x != ' ', text))
    sent_tokenized = ' '.join(text)
    return sent_tokenized

regex = re.compile('[%s]' % re.escape(string.punctuation))

def remove_punct(text):
    text = regex.sub(" ", text)
    return text

clean = CleanText()
```

In [6]:

```
# label encode
def label_encode(x):
    if x == 1 or x == 2:
        return 0
    if x == 3:
        return 1
    if x == 5 or x == 4:
        return 2

# label to name
def label2name(x):
    if x == 0:
        return "Negative"
```

```
    if x == 1:
        return "Neutral"
    if x == 2:
        return "Positive"
```

Read Data

In [7]:

```
df = pd.read_csv("../input/trip-advisor-hotel-reviews/tripadvisor_hotel_reviews.csv")
```

In [8]:

```
# show column names
print("df.columns: ", df.columns)
```

```
df.columns:  Index(['Review', 'Rating'], dtype='object')
```

In [9]:

```
# head of df
df.head()
```

Out[9]:

	Review	Rating
0	nice hotel expensive parking got good deal sta...	4
1	ok nothing special charge diamond member hilto...	2
2	nice rooms not 4* experience hotel monaco seat...	3

3	unique, great stay, wonderful time hotel monac...	5
4	great stay great stay, went seahawk game aweso...	5

In [10]:

```
# count of ratings
fig = px.histogram(df,
             x = 'Rating',
             title = 'Histogram of Review Rating',
             template = 'ggplot2',
             color = 'Rating',
             color_discrete_sequence= px.colors.sequential.Blues_r,
             opacity = 0.8,
             height = 525,
             width = 835,
            )

fig.update_yaxes(title='Count')
fig.show()
```

12345010002000300040005000600070008009000

Rating42351Histogram of Review RatingRatingCount

In [11]:

```
# basic info
df.info()
```

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 20491 entries, 0 to 20490

Data columns (total 2 columns):

```
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Review  20491 non-null  object
 1   Rating  20491 non-null  int64
dtypes: int64(1), object(1)
memory usage: 320.3+ KB
```

In [12]:

```python
# encode label and mapping label name
df["label"] = df["Rating"].apply(lambda x: label_encode(x))
df["label_name"] = df["label"].apply(lambda x: label2name(x))
```

In [13]:

```python
# clean text, lowercase and remove punk
df["Review"] = df["Review"].apply(lambda x: remove_punct(clean(remove_emoji(x).lower())[0][0]))
```

In [14]:

```python
df.head()
```

Out[14]:

	Review	Rating	label	label_name
0	nice hotel expensive parking got good deal sta...	4	2	Positive
1	ok nothing special charge diamond member hilto...	2	0	Negative
2	nice rooms not 4 experience hotel monaco seat...	3	1	Neutral
3	unique great stay wonderful time hotel monac...	5	2	Positive
4	great stay great stay went seahawk game aweso...	5	2	Positive

Visualizations

Word Cloud

Word clouds generators work by breaking the text down into component words and counting how frequently they appear in the body of text. We can quickly obtain preliminary information about the data. We can understand what a dataset we don't know is talking about.

In [15]:

```
def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color='black',
        max_words=200,
        max_font_size=40,
        scale=1,
        random_state=1
).generate(" ".join(data))

    fig = plt.figure(1, figsize=(15, 15))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()
```

In [16]:

```
show_wordcloud(df["Review"].values)
```

Target Count

How many targets do we have? Learning this information will give us an idea about the model we will build. It will also provide guidance on our methods of analyzing data.

In [17]:

```
fig = make_subplots(rows=1, cols=2, specs=[[{"type": "pie"}, {"type": "bar"}]])

colors = ['gold', 'mediumturquoise', 'lightgreen'] # darkorange

fig.add_trace(go.Pie(labels=df.label_name.value_counts().index,

                     values=df.label.value_counts().values), 1, 1)



fig.update_traces(hoverinfo='label+percent', textfont_size=20,

              marker=dict(colors=colors, line=dict(color='#000000', width=2)))



fig.add_trace(go.Bar(x=df.label_name.value_counts().index, y=df.label.value_counts().values,
marker_color = colors), 1,2)



fig.show()
```

PositiveNegativeNeutral02k4k6k8k10k12k14k73.7%15.7%10.7%

PositiveNegativeNeutraltrace 1

Token Counts with simple tokenizer

Finding out the number of tokens available for each sample will give us information about the length of our data. The classification algorithm we will use for a long text will not be the same as the algorithm used for a short text.

In [18]:

```
# tokenize data

df["tokenized_review"] = df.Review.apply(lambda x: tokenize(x))

# calculate token count for any sent

df["sent_token_length"] = df["tokenized_review"].apply(lambda x: len(x.split()))
```

In [19]:

```
fig = px.histogram(df, x="sent_token_length", nbins=20,
color_discrete_sequence=px.colors.cmocean.algae, barmode='group', histnorm="percent")

fig.show()
```

050010001500200001020304050607080090

sent_token_lengthpercent

In [20]:

```
(df.sent_token_length < 512).mean()
```

Out[20]:

0.989117173393197

Token Counts with BERT tokenizer

Since we will create a Transformers-based model, the value that BERT tokinezer will give us is very important. With the information here, the value of the seq_len parameter that we will use while encoding the data will be decided.

In [21]:

```
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',
```

```
                    do_lower_case=True)
```

unfold_moreShow hidden output

In [22]:

```
# data tokenize with bert tokenizer
df["sent_bert_token_length"] = df["Review"].apply(lambda x: len(tokenizer(x, add_special_tokens=False)["input_ids"]))
```

unfold_moreShow hidden output

In [23]:

```
fig = px.histogram(df, x="sent_token_length", nbins=20, color_discrete_sequence=px.colors.cmocean.algae, barmode='group', histnorm="percent")
fig.show()
```

050010001500200001020304050607080 90

sent_token_lengthpercent

In [24]:

```
# Less than 512 covers how many of the data
(df.sent_bert_token_length < 512).mean()
```

Out[24]:

0.9853106241764678

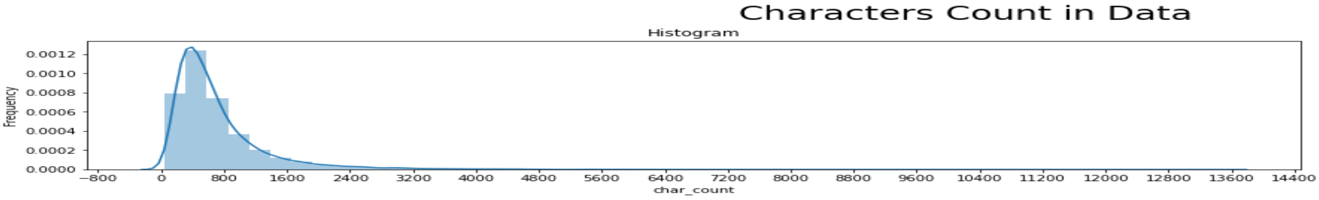
Characters Count in the Data

Let's look at the frequency of the number of characters. It will give us information about the overall size of our data

unfold_moreShow hidden code

In [26]:

```
plot_dist3(df, 'char_count',
           'Characters Count in Data')
```



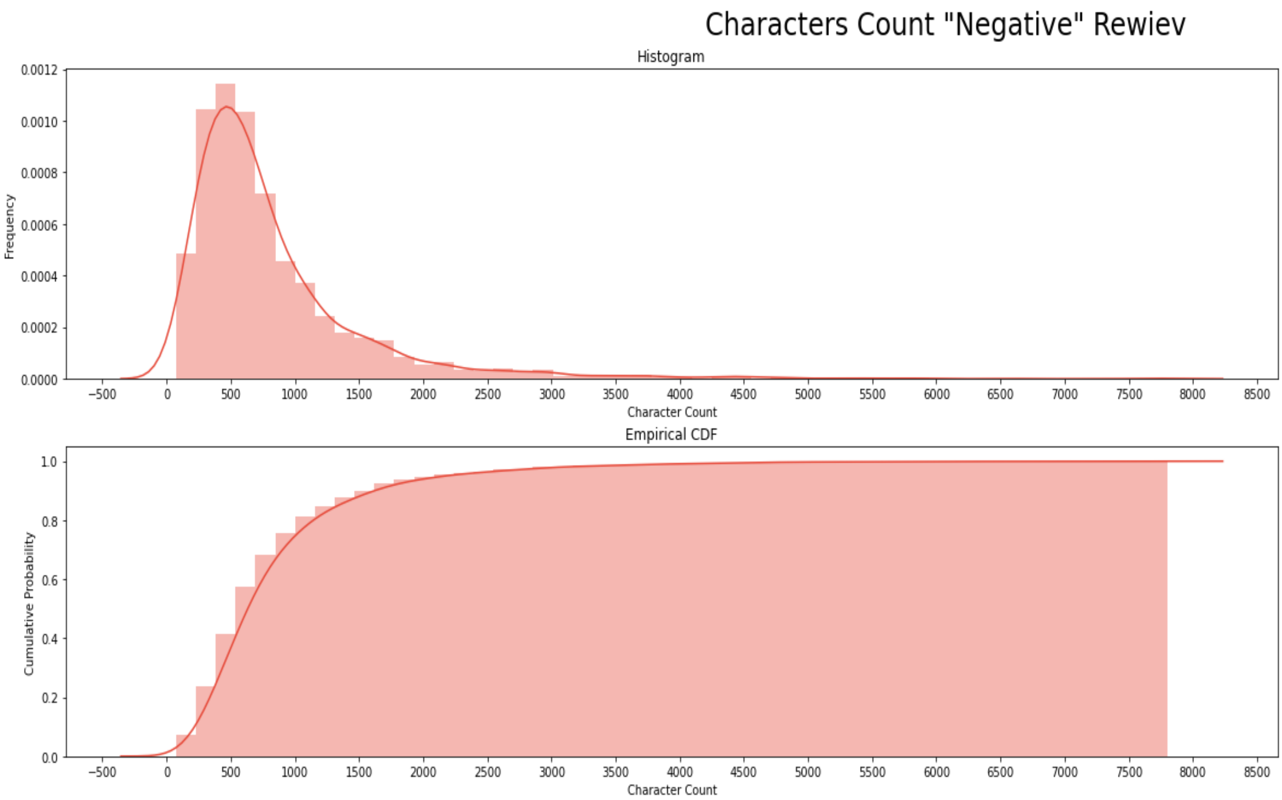
Reviews Lengths

When we look at the number of characters per comment, it can give us very striking information about the data. Here, when we look at the length of the comments made by people according to their feelings, negative comments are shorter than neutral and positive comments. We can come to the notion that people simply express negative things :).

unfold_moreShow hidden code

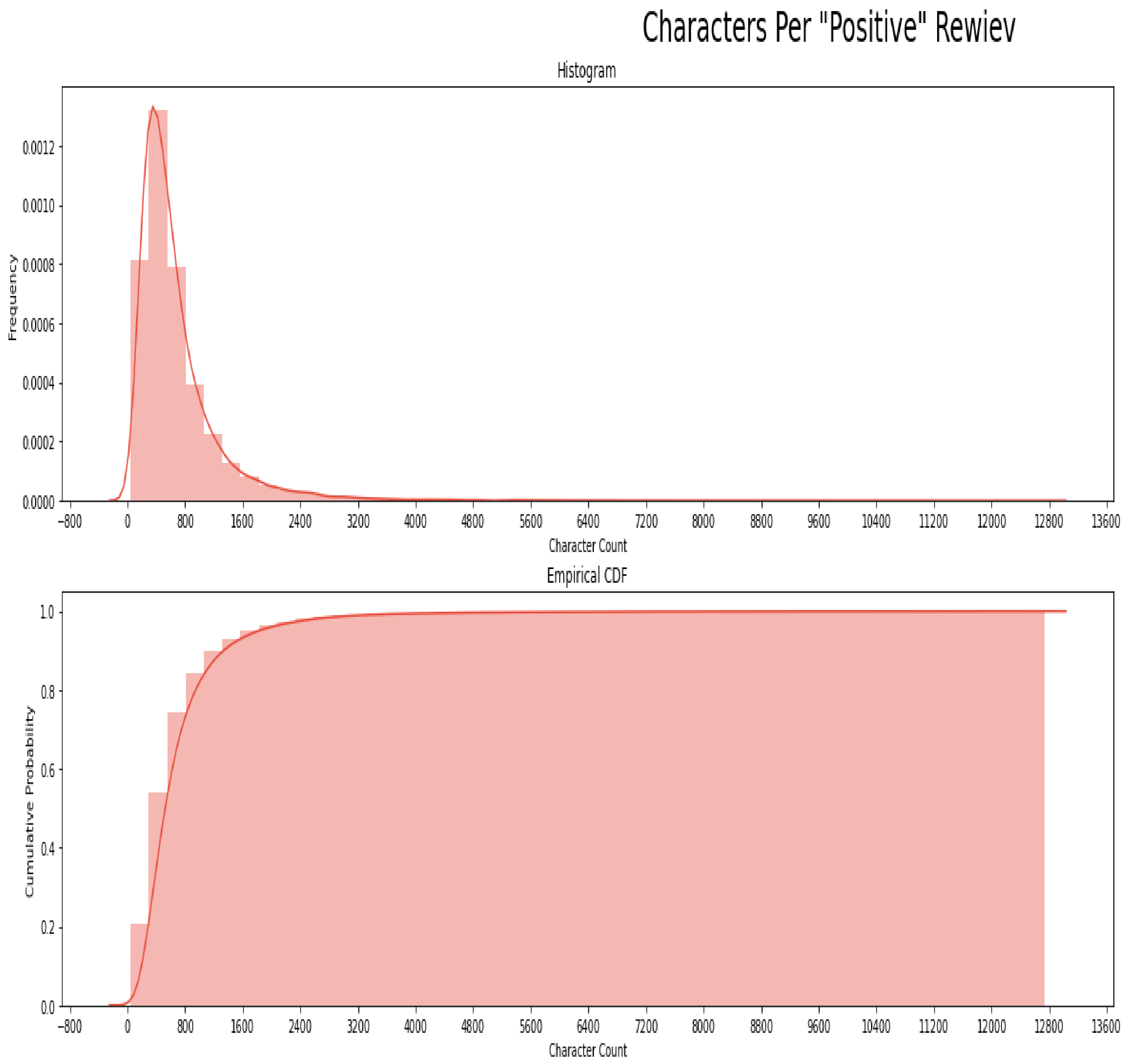
In [28]:

```
plot_dist3(df[df['label'] == 0], 'Character Count',
           'Characters Count "Negative" Rewiev')
```



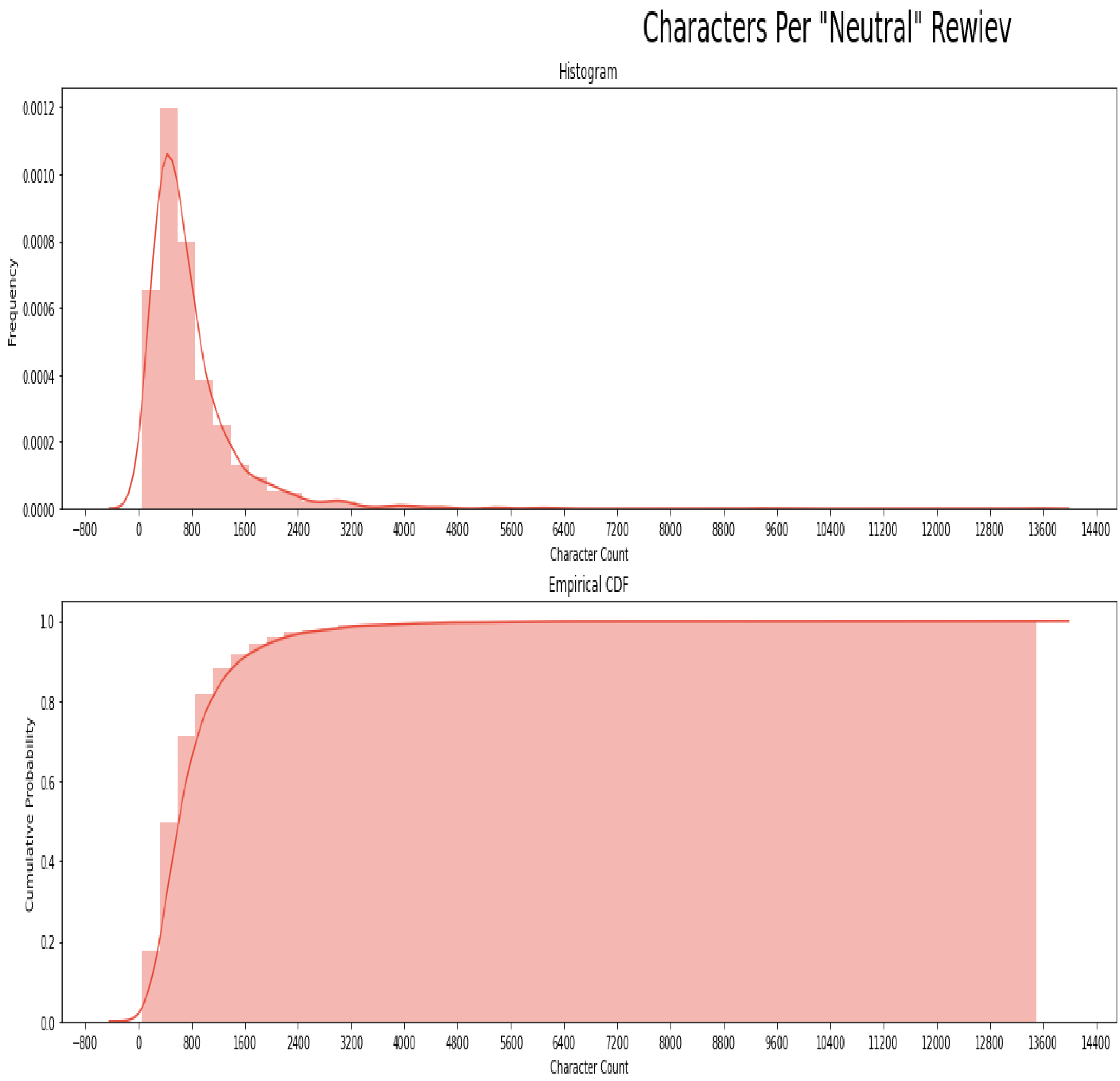
In [29]:

```
plot_dist3(df[df['label'] == 2], 'Character Count',

        'Characters Per "Positive" Rewiev')
```



In [30]:

```
plot_dist3(df[df['label'] == 1], 'Character Count',

        'Characters Per "Neutral" Rewiev')
```



Word Counts

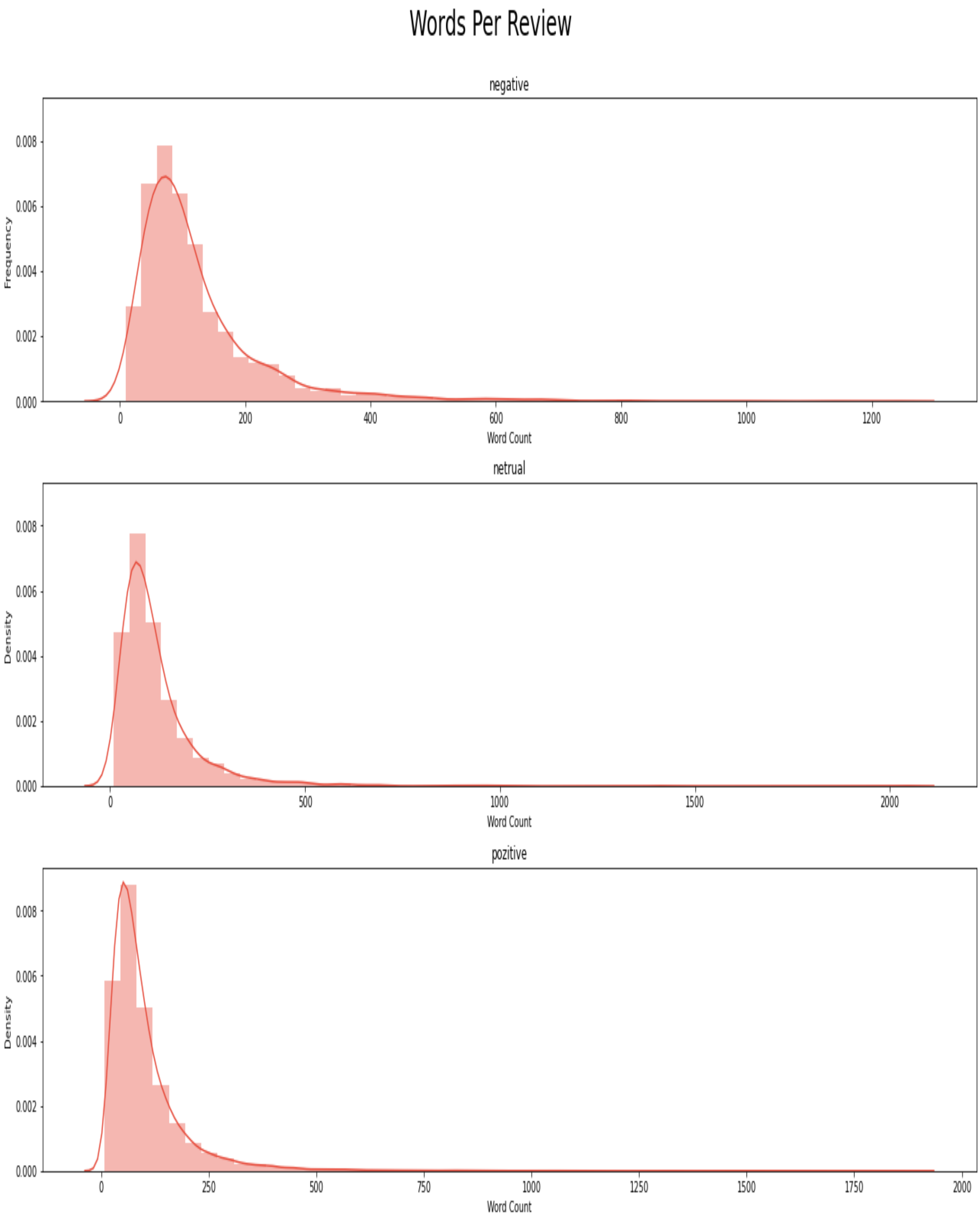
We see that the situation in the number of characters and the situation in the number of words are the same. We have seen that people use less word count when expressing negative things.

unfold_moreShow hidden code

In [32]:

```
plot_word_number_histogram(df[df['label'] == 0]['Review'],
                           df[df['label'] == 1]['Review'],
                           df[df['label'] == 2]['Review'],
```

)



In [33]:

 # remove punk

```
df['tokenized_review'] = df['tokenized_review'].apply(lambda x: remove_punct(x))
```

Most Common Words

In [34]:

```
texts = df['tokenized_review']
new = texts.str.split()
new = new.values.tolist()
corpus = [word for i in new for word in i]
counter = Counter(corpus)
most = counter.most_common()
x, y = [], []
for word, count in most[:30]:
    if word not in stopWords_nltk:
        x.append(word)
        y.append(count)


fig = go.Figure(go.Bar(
            x=y,
            y=x,
            orientation='h',  marker=dict(
        color='rgba(50, 171, 96, 0.6)',
        line=dict(
            color='rgba(50, 171, 96, 1.0)',
            width=1),
    ),
    name='Most common Word',))
```

```
fig.update_layout( title={

        'text': "Most Common Words",

        'y':0.9,

        'x':0.5,

        'xanchor': 'center',

        'yanchor': 'top'}, font=dict(

        family="Courier New, monospace",

        size=18,

        color="RebeccaPurple"

    ))


fig.show()
```

010k20k30k40k50khotelgreatgoodstayroomsstayednightbeachbreakfastfoodresortplace

Most Common Words

Most Common ngrams

In [35]:

```
fig = make_subplots(rows=1, cols=3)

title_ = ["negative", "neutral", "positive"]


for i in range(3):

    texts = df[df["label"] == i]['tokenized_review']


    new = texts.str.split()

    new = new.values.tolist()

    corpus = [word for i in new for word in i]

    counter = Counter(corpus)
```

```
    most = counter.most_common()
    x, y = [], []

    for word, count in most[:30]:
        if word not in stopWords_nltk:
            x.append(word)
            y.append(count)

    fig.add_trace(go.Bar(
                x=y,
                y=x,
                orientation='h', type="bar",
        name=title_[i], marker=dict(color=colors[i])), 1, i+1)

fig.update_layout(
    autosize=False,
    width=2000,
    height=600,title=dict(
        text='<b>Most Common ngrams per Classes</b>',
        x=0.5,
        y=0.95,
        font=dict(
        family="Courier New, monospace",
        size=24,
        color="RebeccaPurple"
        )
    ),)
```

```
fig.show()
```

02000400060008000hotelroomnstayroomsstaffnightgoodservicedaytimefoodlikeresort2beachstayedgotnice3toldpeopledeskplacegreat010002000300040005000hotelroomngoodnicegreatroomsstafflocationstaybeachnightfoodcleanservicedaytimelikestayedresortbreakfastpool2small010k20k30khotelroomgreatstaffgoodnstaynicelocationroomsstayedbreakfastcleantimebeachservicedaynightfoodfriendlyreallyplaceexcellentpool

negativeneutralpositiveMost Common ngrams per Classes

In [36]:

```
def _get_top_ngram(corpus, n=None):
    #getting top ngrams
    vec = CountVectorizer(ngram_range=(n, n),
                          max_df=0.9,
                          ).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx])
                  for word, idx in vec.vocabulary_.items()]
    words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
    return words_freq[:15]
```

In [37]:

```
# unigram
fig = make_subplots(rows=1, cols=3)

title_ = ["negative", "neutral", "positive"]

for i in range(3):
```

```
    texts = df[df["label"] == i]['tokenized_review']

    new = texts.str.split()
    new = new.values.tolist()
    corpus = [word for i in new for word in i]
    top_n_bigrams = _get_top_ngram(texts, 2)[:15]
    x, y = map(list, zip(*top_n_bigrams))


    fig.add_trace(go.Bar(
            x=y,
            y=x,
            orientation='h', type="bar",
      name=title_[i], marker=dict(color=colors[i])), 1, i+1)


fig.update_layout(
    autosize=False,
    width=2000,
    height=600,title=dict(
      text='<b>Most Common unigrams per Classes</b>',
      x=0.5,
      y=0.95,
      font=dict(
      family="Courier New, monospace",
      size=24,
      color="RebeccaPurple"
```

```
    )
  ))
fig.show()
```

02004006008001000did notpunta canaroom notstar hotelhotel notnot stayroom servicenot goodcheck inair conditioningstay hotelnot worthnot recommendcustomer servicecredit card0100200300400500did notgreat locationstaff friendlypunta cananot badgood locationnot goodroom cleanroom servicecheck inwalking distancehotel notsan juanstar hotelstayed hotel0500100015002000did notgreat locationstaff friendlygreat hotelfriendly helpfulhotel greatwalking distancerecommend hotelpunta canahighly recommendhotel staffth floorjust returnedminute walkstayed hotel

negativeneutralpositiveMost Common unigrams per Classes

In [38]:

```
#trigram

fig = make_subplots(rows=1, cols=3)
title_ = ["negative", "neutral", "positive"]

for i in range(3):
  texts = df[df["label"] == i]['tokenized_review']

  new = texts.str.split()
  new = new.values.tolist()
  corpus = [word for i in new for word in i]

  top_n_bigrams = _get_top_ngram(texts, 3)[:15]
  x, y = map(list, zip(*top_n_bigrams))

  fig.add_trace(go.Bar(
        x=y,
```

```
            y=x,
            orientation='h', type="bar",
        name=title_[i], marker=dict(color=colors[i])), 1, i+1),

fig.update_layout(
    autosize=False,
    width=2000,
    height=600,title=dict(
        text='<b>Most Common trigrams per Classes</b>',
        x=0.5,
        y=0.95,
        font=dict(
        family="Courier New, monospace",
        size=24,
        color="RebeccaPurple"
        )
    ))

fig.show()
```

020406080100120did not worknot recommend hotelold san juannon smoking roomroom not readyroom did notnot star hotelno air conditioningnot worth moneyking size bedno hot waternot stay hotelhotel did notworst hotel stayeddid not want01020304050old san juanstaff friendly helpfulhotel great locationstayed hotel nightsking size bedgood value moneyhotel good location10 minute walkflat screen tvel san juandid not likenon smoking roomdid not workla carte restaurantsjust returned week0200400600staff friendly helpfulhotel great locationhighly recommend hotelgreat place stayold san juanflat screen tvgreat hotel great10 minute walkking size bedgood value moneyeasy walking distancehotel staff friendlyfree internet accessstaff helpful friendlyjust returned night

negativeneutralpositiveMost Common trigrams per Classes

We examined and visualized the data, now we can move on to the model building part.

3. MODELS

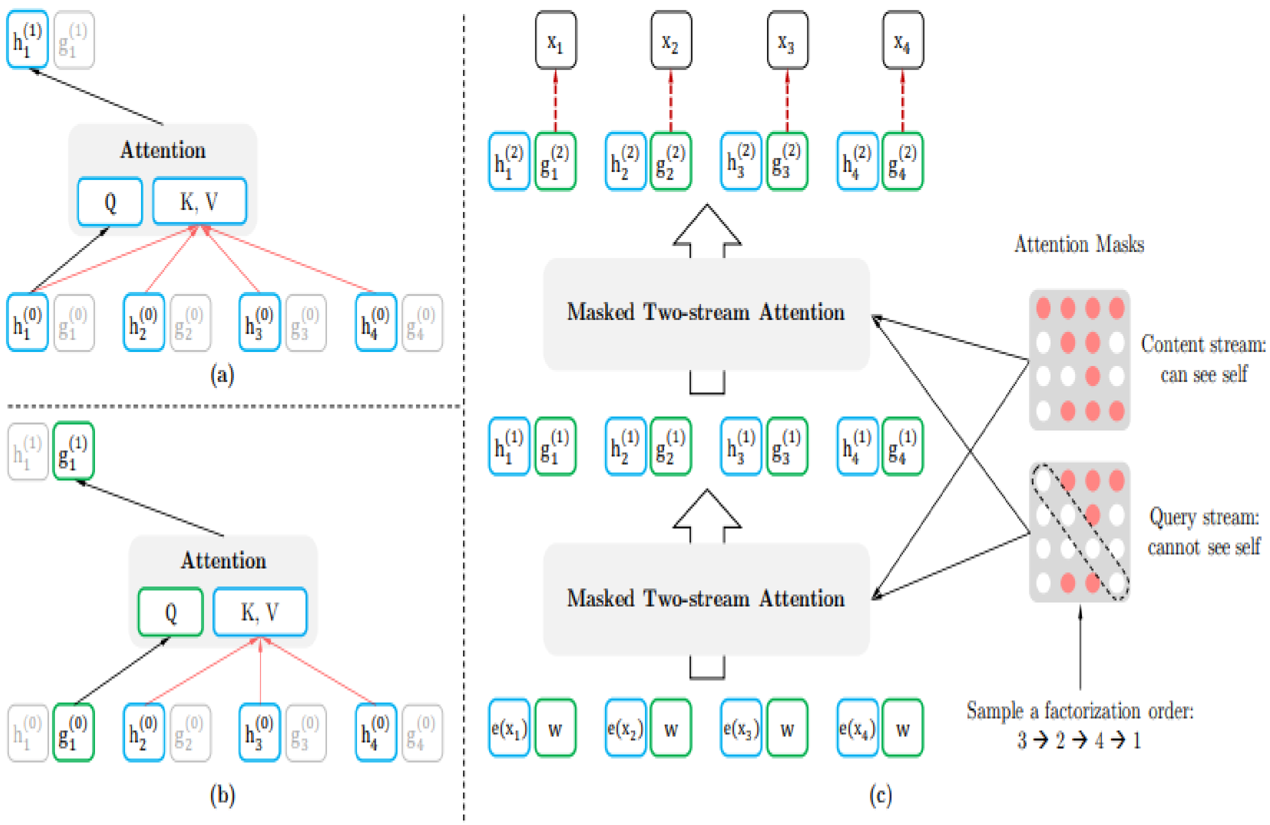
A brief information about BERT

BERT makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. In its vanilla form, Transformer includes two separate mechanisms — an encoder that reads the text input and a decoder that produces a prediction for the task. Since BERT's goal is to generate a language model, only the encoder mechanism is necessary.

BERT is a bi-directional transformer for pre-training over a lot of unlabeled textual data to learn a language representation that can be used to fine-tune for specific machine learning tasks. While BERT outperformed the NLP state-of-the-art on several challenging tasks, its performance improvement could be attributed to the bidirectional transformer, novel pre-training tasks of Masked Language Model and Next Structure Prediction along with a lot of data and Google's compute power.

The detailed workings of Transformer are described in a paper by Google.

A brief information about XLNET



XLNet is a large bidirectional transformer that uses improved training methodology, larger data and more computational power to achieve better than BERT prediction metrics on 20 language tasks.

To improve the training, XLNet introduces permutation language modeling, where all tokens are predicted but in random order. This is in contrast to BERT’s masked language model where only the masked (15%) tokens are predicted. This is also in contrast to the traditional language models, where all tokens were predicted in sequential order instead of random order. This helps the model to learn bidirectional relationships and therefore better handles dependencies and relations between words. In addition, Transformer XL was used as the base architecture, which showed good performance even in the absence of permutation-based training.

XLNet was trained with over 130 GB of textual data and 512 TPU chips running for 2.5 days, both of which ar e much larger than BERT.

A brief information about RoBERTa

RoBERTa. Introduced at Facebook, Robustly optimized BERT approach RoBERTa, is a retraining of BERT with improved training methodology, 1000% more data and compute power.

To improve the training procedure, RoBERTa removes the Next Sentence Prediction (NSP) task from BERT’s pre-training and introduces dynamic masking so that the masked token changes during the training epochs. Larger batch-training sizes were also found to be more useful in the training procedure.

Importantly, RoBERTa uses 160 GB of text for pre-training, including 16GB of Books Corpus and English Wikipedia used in BERT. The additional data included CommonCrawl News dataset (63 million articles, 76 GB), Web text corpus (38 GB) and Stories from Common Crawl (31 GB). This coupled with whopping 1024 V100 Tesla GPU’s running for a day, led to pre-training of RoBERTa.

Comparison of Transformer Models

	BERT	RoBERTa	DistilBERT	XLNet
Size (millions)	**Base**: 110 **Large**: 340	**Base**: 110 **Large**: 340	**Base**: 66	**Base**: ~110 **Large**: ~340
Training Time	**Base**: 8 x V100 x 12 days* **Large**: 64 TPU Chips x 4 days (or 280 x V100 x 1 days*)	**Large**: 1024 x V100 x 1 day; 4-5 times more than BERT.	**Base**: 8 x V100 x 3.5 days; 4 times less than BERT.	**Large**: 512 TPU Chips x 2.5 days; 5 times more than BERT.
Performance	Outperforms state-of-the-art in Oct 2018	2-20% improvement over BERT	3% degradation from BERT	2-15% improvement over BERT
Data	16 GB BERT data (Books Corpus + Wikipedia). 3.3 Billion words.	160 GB (16 GB BERT data + 144 GB additional)	16 GB BERT data. 3.3 Billion words.	**Base**: 16 GB BERT data **Large**: 113 GB (16 GB BERT data + 97 GB additional). 33 Billion words.
Method	BERT (Bidirectional Transformer with MLM and NSP)	BERT without NSP**	BERT Distillation	Bidirectional Transformer with Permutation based modeling

source = https://towardsdatascience.com/bert-roberta-distilbert-xlnet-which-one-to-use-3d5ab82ba5f8

In this table, the models are compared under 5 headings, let's take them all one by one.

1. When we look at the sizes of the models, BERT, RoBERTa and XLNet have the same values, while the size of the DistillBERT is smaller.

2. The biggest factor that determines Training Times is the size of the models and the data they have. As you can imagine, the time increases as the size increases :).

3. When we look at the performance, BERT considers the model as the base model. RoBERTa offers 2-20% better performance than BERT. A similar performance applies to XLNet. XLNet performs 2-15% better than BERT model. DisltiBERT, despite its small size, is not equally poor in performance. It performs only 3% worse.

4. When we look at its data, the model with the largest corpus is ROBERTa. It is followed by XLNET, then BERT and DistilBERT have the same data. One of the reasons for the higher performance of RoBERTa and XLNet is that the datasets are so high.

1. As it is known, there are MLM and NSP tasks in the BERT model. The RoBERTa model is the trained version of the BERT model without the NSP task. DiltilBERT is a reduced number of parameters of BERT, it maintains 97% performance, but uses only half the number of parameters (paper). To enhance the training, XLNet offers permutation language modeling where all tokens are predicted but in random order.

I recommend you to read the articles for more detailed information.

Preprocess for BERT Train

In [39]:

```
import pandas as pd
import numpy as np
import os
import random
from pathlib import Path
import json
```

In [40]:

```
import torch
from tqdm.notebook import tqdm

from transformers import BertTokenizer
from torch.utils.data import TensorDataset

from transformers import BertForSequenceClassification
```

In [41]:

```
class Config():
    seed_val = 17
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    epochs = 5
    batch_size = 6
    seq_length = 512
    lr = 2e-5
    eps = 1e-8
    pretrained_model = 'bert-base-uncased'
```

```
    test_size=0.15
    random_state=42
    add_special_tokens=True
    return_attention_mask=True
    pad_to_max_length=True
    do_lower_case=False
    return_tensors='pt'

config = Config()
```

In [42]:

```
# params will be saved after training
params = {"seed_val": config.seed_val,
    "device":str(config.device),
    "epochs":config.epochs,
    "batch_size":config.batch_size,
    "seq_length":config.seq_length,
    "lr":config.lr,
    "eps":config.eps,
    "pretrained_model": config.pretrained_model,
    "test_size":config.test_size,
    "random_state":config.random_state,
    "add_special_tokens":config.add_special_tokens,
    "return_attention_mask":config.return_attention_mask,
    "pad_to_max_length":config.pad_to_max_length,
    "do_lower_case":config.do_lower_case,
    "return_tensors":config.return_tensors,
```

```
}
```

In [43]:

```
# set random seed and device
import random

device = config.device

random.seed(config.seed_val)
np.random.seed(config.seed_val)
torch.manual_seed(config.seed_val)
torch.cuda.manual_seed_all(config.seed_val)
```

In [44]:

```
df.head()
```

Out[44]:

	Review	Rating	label	label_name	tokenized_review	sent_token_length	sent_bert_token_length	char_count	Character Count
0	nice hotel expensive parking got good deal sta...	4	2	Positive	nice hotel expensive parking got good deal sta...	88	91	593	593
1	ok nothing special charge diamond member hilto...	2	0	Negative	ok nothing special charge diamond member hilto...	258	268	1689	1689

2	nice rooms not 4 experience hotel monaco seat...	3	1	Neutral	nice rooms not 4 experience hotel monaco seatt...	237	273	1427	1427
3	unique great stay wonderful time hotel monac...	5	2	Positive	unique great stay wonderful time hotel monaco ...	92	102	600	600
4	great stay great stay went seahawk game aweso...	5	2	Positive	great stay great stay went seahawk game awesom...	197	213	1281	1281

Train and Validation Split

In [45]:

```
#split train test
from sklearn.model_selection import train_test_split


train_df_, val_df = train_test_split(df,
                                     test_size=0.10,
                                     random_state=config.random_state,
                             stratify=df.label.values)
```

In [46]:

```
train_df_.head()
```

Out[46]:

	Review	Rating	label	label_name	tokenized_review	sent_token_length	sent_bert_token_length	char_count	Character Count

8159	central simple 4 nights bbvery small room no a...	3	1	Neutral	central simple 4 nights bbvery small room no a...	27	37	208	208
15738	stay stayed flight cancelled stranded 3 days ...	5	2	Positive	stay stayed flight cancelled stranded 3 days a...	75	87	487	487
9972	n t want stay picked hotel du candran excellen...	5	2	Positive	n t want stay picked hotel du candran excellen...	142	162	902	902
7265	best deal town reserved internet months advanc...	5	2	Positive	best deal town reserved internet months advanc...	48	48	353	353
8747	nice place wife arrived usa 10am offered choic...	4	2	Positive	nice place wife arrived usa 10 am offered choi...	86	91	579	579

In [47]:

```
train_df, test_df = train_test_split(train_df_,
                                     test_size=0.10,
                                     random_state=42,
                             stratify=train_df_.label.values)
```

In [48]:

```
# count of unique label  control
```

```
print(len(train_df['label'].unique()))
print(train_df.shape)
```

```
3
(16596, 9)
```

In [49]:

```
# count of unique label  control
print(len(val_df['label'].unique()))
print(val_df.shape)
```

```
3
(2050, 9)
```

In [50]:

```
print(len(test_df['label'].unique()))
print(test_df.shape)
```

```
3
(1845, 9)
```

BertTokenizer and Encoding the Data

In [51]:

```
# create tokenizer
tokenizer = BertTokenizer.from_pretrained(config.pretrained_model,
                                          do_lower_case=config.do_lower_case)
```

In [52]:

```
encoded_data_train = tokenizer.batch_encode_plus(
    train_df.Review.values,
    add_special_tokens=config.add_special_tokens,
    return_attention_mask=config.return_attention_mask,
    pad_to_max_length=config.pad_to_max_length,
    max_length=config.seq_length,
    return_tensors=config.return_tensors
)
encoded_data_val = tokenizer.batch_encode_plus(
    val_df.Review.values,
    add_special_tokens=config.add_special_tokens,
    return_attention_mask=config.return_attention_mask,
    pad_to_max_length=config.pad_to_max_length,
    max_length=config.seq_length,
    return_tensors=config.return_tensors
)
```

Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate examples to max length. Defaulting to 'longest_first' truncation strategy. If you encode pairs of sequences (GLUE-style) with the tokenizer you can select this strategy more precisely by providing a specific strategy to `truncation`.

In [53]:

```
input_ids_train = encoded_data_train['input_ids']
attention_masks_train = encoded_data_train['attention_mask']
labels_train = torch.tensor(train_df.label.values)

input_ids_val = encoded_data_val['input_ids']
```

```
attention_masks_val = encoded_data_val['attention_mask']

labels_val = torch.tensor(val_df.label.values)
```

In [54]:

```
dataset_train = TensorDataset(input_ids_train, attention_masks_train, labels_train)

dataset_val = TensorDataset(input_ids_val, attention_masks_val, labels_val)
```

Creating the Model

- bert-base-uncased is a smaller pre-trained model.
- Using num_labels to indicate the number of output labels.

In [55]:

```
model = BertForSequenceClassification.from_pretrained(config.pretrained_model,
                                                      num_labels=3,
                                                      output_attentions=False,
                                                      output_hidden_states=False)
```

Downloading: 100%

420M/420M [00:10<00:00, 42.8MB/s]

Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertForSequenceClassification: ['cls.predictions.bias', 'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.transform.dense.bias', 'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.dense.weight', 'cls.seq_relationship.bias', 'cls.predictions.decoder.weight', 'cls.seq_relationship.weight']

- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).

- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

Data Loaders

- DataLoader combines a dataset and a sampler, and provides an iterable over the given dataset.
- We use RandomSampler for training and SequentialSampler for validation.
- Given the limited memory in my environment, I set batch_size=64.

In [56]:

```
from torch.utils.data import DataLoader, RandomSampler, SequentialSampler

dataloader_train = DataLoader(dataset_train,
                              sampler=RandomSampler(dataset_train),
                              batch_size=config.batch_size)

dataloader_validation = DataLoader(dataset_val,
                                   sampler=SequentialSampler(dataset_val),
                                   batch_size=config.batch_size)
```

Optimizer & Scheduler

In [57]:

```
from transformers import AdamW, get_linear_schedule_with_warmup

optimizer = AdamW(model.parameters(),
                  lr=config.lr,
                  eps=config.eps)
```

```
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps=0,
                                            num_training_steps=len(dataloader_train)*config.epochs)
```

Performance Metrics

We will use f1 score as performance metrics.

In [58]:

```
from sklearn.metrics import f1_score


def f1_score_func(preds, labels):
    preds_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return f1_score(labels_flat, preds_flat, average='weighted')


def accuracy_per_class(preds, labels, label_dict):
    label_dict_inverse = {v: k for k, v in label_dict.items()}

    preds_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()

    for label in np.unique(labels_flat):
        y_preds = preds_flat[labels_flat==label]
        y_true = labels_flat[labels_flat==label]
        print(f'Class: {label_dict_inverse[label]}')
        print(f'Accuracy: {len(y_preds[y_preds==label])}/{len(y_true)}\n')
```

Training Loop

In [59]:

```
def evaluate(dataloader_val):

    model.eval()

    loss_val_total = 0
    predictions, true_vals = [], []

    for batch in dataloader_val:

        batch = tuple(b.to(config.device) for b in batch)

        inputs = {'input_ids':      batch[0],
                  'attention_mask': batch[1],
                  'labels':         batch[2],
                 }

        with torch.no_grad():
            outputs = model(**inputs)

        loss = outputs[0]
        logits = outputs[1]
        loss_val_total += loss.item()

        logits = logits.detach().cpu().numpy()
        label_ids = inputs['labels'].cpu().numpy()
        predictions.append(logits)
```

```
        true_vals.append(label_ids)

    # calculate avareage val loss
    loss_val_avg = loss_val_total/len(dataloader_val)

    predictions = np.concatenate(predictions, axis=0)
    true_vals = np.concatenate(true_vals, axis=0)

    return loss_val_avg, predictions, true_vals
```

In [60]:

```
config.device
```

Out[60]:

```
device(type='cuda', index=0)
```

In [61]:

```
model.to(config.device)

for epoch in tqdm(range(1, config.epochs+1)):

    model.train()

    loss_train_total = 0
    # allows you to see the progress of the training
    progress_bar = tqdm(dataloader_train, desc='Epoch {:1d}'.format(epoch), leave=False, disable=False)

    for batch in progress_bar:
```

```python
        model.zero_grad()

        batch = tuple(b.to(config.device) for b in batch)


        inputs = {'input_ids':      batch[0],
                  'attention_mask': batch[1],
                  'labels':         batch[2],
                 }

        outputs = model(**inputs)

        loss = outputs[0]
        loss_train_total += loss.item()
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

        optimizer.step()
        scheduler.step()

        progress_bar.set_postfix({'training_loss': '{:.3f}'.format(loss.item()/len(batch))})


    torch.save(model.state_dict(), f'_BERT_epoch_{epoch}.model')
```

```python
    tqdm.write(f'\nEpoch {epoch}')

    loss_train_avg = loss_train_total/len(dataloader_train)
    tqdm.write(f'Training loss: {loss_train_avg}')

    val_loss, predictions, true_vals = evaluate(dataloader_validation)
    val_f1 = f1_score_func(predictions, true_vals)
    tqdm.write(f'Validation loss: {val_loss}')

    tqdm.write(f'F1 Score (Weighted): {val_f1}');
# save model params and other configs
with Path('params.json').open("w") as f:
    json.dump(params, f, ensure_ascii=False, indent=4)
```

```
100%
5/5 [1:26:39<00:00, 1038.70s/it]
Epoch 1: 100%
2766/2766 [16:42<00:00, 2.76it/s, training_loss=0.113]
Epoch 1
Training loss: 0.44685599791267866
Validation loss: 0.30867522299747197
F1 Score (Weighted): 0.8787187536388859

Epoch 2: 100%
2766/2766 [16:40<00:00, 2.60it/s, training_loss=0.078]
Epoch 2
Training loss: 0.33569076879218773
```

Validation loss: 0.443880146502O9234

F1 Score (Weighted): 0.8733283050365404

Epoch 3: 100%

2766/2766 [16:40<00:00, 2.78it/s, training_loss=0.113]

Epoch 3

Training loss: 0.26331509235532197

Validation loss: 0.4841138020460596

F1 Score (Weighted): 0.8839202492823627

Epoch 4: 100%

2766/2766 [16:38<00:00, 2.70it/s, training_loss=0.000]

Epoch 4

Training loss: 0.174491831848849

Validation loss: 0.6204505104885426

F1 Score (Weighted): 0.8782044542022744

Epoch 5: 100%

2766/2766 [16:35<00:00, 2.80it/s, training_loss=0.000]

Epoch 5

Training loss: 0.10495032141427339

Validation loss: 0.7065923309128053

F1 Score (Weighted): 0.8772936330208443

Test on validation set

In [62]:

```
model.load_state_dict(torch.load(f'./_BERT_epoch_3.model', map_location=torch.device('cpu')))
```

Out[62]:

<All keys matched successfully>

In [63]:

```
from sklearn.metrics import classification_report

preds_flat = np.argmax(predictions, axis=1).flatten()
print(classification_report(preds_flat, true_vals))
```

```
              precision    recall  f1-score   support

           0       0.82      0.85      0.83       310
           1       0.48      0.46      0.47       227
           2       0.95      0.94      0.95      1513

    accuracy                           0.88      2050
   macro avg       0.75      0.75      0.75      2050
weighted avg       0.88      0.88      0.88      2050
```

4. ERROR ANALYSIS

In [64]:

```
# step by step predictions on dataframe
# We do this to view predictions in the pandas dataframe and easily filter them and perform error analysis.

pred_final = []
```

```
for i, row in tqdm(val_df.iterrows(), total=val_df.shape[0]):
    predictions = []

    review = row["Review"]
    encoded_data_test_single = tokenizer.batch_encode_plus(
    [review],
    add_special_tokens=config.add_special_tokens,
    return_attention_mask=config.return_attention_mask,
    pad_to_max_length=config.pad_to_max_length,
    max_length=config.seq_length,
    return_tensors=config.return_tensors
    )
    input_ids_test = encoded_data_test_single['input_ids']
    attention_masks_test = encoded_data_test_single['attention_mask']


    inputs = {'input_ids':      input_ids_test.to(device),
              'attention_mask':attention_masks_test.to(device),
             }

    with torch.no_grad():
        outputs = model(**inputs)

    logits = outputs[0]
    logits = logits.detach().cpu().numpy()
    predictions.append(logits)
```

```
    predictions = np.concatenate(predictions, axis=0)

    pred_final.append(np.argmax(predictions, axis=1).flatten()[0])
```

100%

2050/2050 [00:52<00:00, 41.06it/s]

In [65]:

```
# add pred into val_df

val_df["pred"] = pred_final
```

In [66]:

```
#  Add control column for easier wrong and right predictions

control = val_df.pred.values == val_df.label.values

val_df["control"] = control
```

In [67]:

```
# filtering false predictions

val_df = val_df[val_df.control == False]
```

In [68]:

```
# buraları düzenle bbaaaabbaaaaa

# label to intent mapping

name2label = {"Negative":0,

              "Neutral":1,

             "Positive":2

             }

label2name = {v: k for k, v in name2label.items()}
```

```
val_df["pred_name"] = val_df.pred.apply(lambda x: label2name.get(x))
```

In [69]:

```
from sklearn.metrics import confusion_matrix

# We create a confusion matrix to better observe the classes that the model confuses.
pred_name_values = val_df.pred_name.values
label_values = val_df.label_name.values
confmat = confusion_matrix(label_values, pred_name_values, labels=list(name2label.keys()))
```

In [70]:

```
confmat
```

Out[70]:

```
array([[ 0, 66,  4],
       [27,  0, 68],
       [ 9, 71,  0]])
```

In [71]:

```
df_confusion_val = pd.crosstab(label_values, pred_name_values)
df_confusion_val
```

Out[71]:

col_0	Negative	Neutral	Positive
row_0			
Negative	0	66	4
Neutral	27	0	68

Positive	9	71	0

In [72]:

```
# save confissuan matrix df
df_confusion_val.to_csv("val_df_confusion.csv")
```

5. INFERENCE

In [73]:

```
test_df.head()
```

Out[73]:

	Review	Rating	label	label_name	tokenized_review	sent_token_length	sent_bert_token_length	char_count	Character Count
2298	great location nice hotel family 5 stayed june...	4	2	Positive	great location nice hotel family 5 stayed june...	38	39	260	260
9503	welcoming spotless just returned 2nd visit bar...	5	2	Positive	welcoming spotless just returned 2 nd visit ba...	68	77	470	470
14742	beautiful resort beautiful gardens friendly st...	3	1	Neutral	beautiful resort beautiful gardens friendly st...	81	86	506	506
4140	cheaply renovated wo n t going aside	2	0	Negative	cheaply renovated wo n t going aside beautiful...	104	113	684	684

	beautiful...								
3552	nothing spectacular time dr time doing inclusi...	3	1	Neutral	nothing spectacular time dr time doing inclusi...	110	128	719	719

In [74]:

```
encoded_data_test = tokenizer.batch_encode_plus(
    test_df.Review.values,
    add_special_tokens=config.add_special_tokens,
    return_attention_mask=config.return_attention_mask,
    pad_to_max_length=config.pad_to_max_length,
    max_length=config.seq_length,
    return_tensors=config.return_tensors
)
```

In [75]:

```
input_ids_test = encoded_data_test['input_ids']
attention_masks_test = encoded_data_test['attention_mask']
labels_test = torch.tensor(test_df.label.values)
```

In [76]:

```
model = BertForSequenceClassification.from_pretrained(config.pretrained_model,
                                                      num_labels=3,
                                                      output_attentions=False,
                                                      output_hidden_states=False)
```

```
model.to(config.device)


model.load_state_dict(torch.load(f'./_BERT_epoch_3.model', map_location=torch.device('cpu')))


_, predictions_test, true_vals_test = evaluate(dataloader_validation)

# accuracy_per_class(predictions, true_vals, intent2label)
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertForSequenceClassification: ['cls.predictions.bias', 'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.transform.dense.bias', 'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.dense.weight', 'cls.seq_relationship.bias', 'cls.predictions.decoder.weight', 'cls.seq_relationship.weight']

- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).

- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

In [77]:

```
from sklearn.metrics import classification_report


preds_flat_test = np.argmax(predictions_test, axis=1).flatten()

print(classification_report(preds_flat_test, true_vals_test))
```

```
         precision    recall  f1-score   support


       0       0.78      0.88      0.83       288
```

```
           1       0.56      0.47      0.51       260
           2       0.95      0.95      0.95      1502

    accuracy                           0.88      2050
   macro avg       0.76      0.77      0.76      2050
weighted avg       0.88      0.88      0.88      2050
```

In [78]:

```
pred_final = []

for i, row in tqdm(test_df.iterrows(), total=test_df.shape[0]):
    predictions = []

    review = row["Review"]
    encoded_data_test_single = tokenizer.batch_encode_plus(
    [review],
    add_special_tokens=config.add_special_tokens,
    return_attention_mask=config.return_attention_mask,
    pad_to_max_length=config.pad_to_max_length,
    max_length=config.seq_length,
    return_tensors=config.return_tensors
    )
    input_ids_test = encoded_data_test_single['input_ids']
    attention_masks_test = encoded_data_test_single['attention_mask']
```

```
    inputs = {'input_ids':      input_ids_test.to(device),
              'attention_mask':attention_masks_test.to(device),
             }

    with torch.no_grad():
        outputs = model(**inputs)

    logits = outputs[0]
    logits = logits.detach().cpu().numpy()
    predictions.append(logits)
    predictions = np.concatenate(predictions, axis=0)
    pred_final.append(np.argmax(predictions, axis=1).flatten()[0])
```

100%

1845/1845 [00:47<00:00, 39.87it/s]

In [79]:

```
# add pred into test
test_df["pred"] = pred_final
```

In [80]:

```
#  Add control column for easier wrong and right predictions
control = test_df.pred.values == test_df.label.values
test_df["control"] = control
```

In [81]:

```
# filtering false predictions
test_df = test_df[test_df.control == False]
```

In [82]:

```
test_df["pred_name"] = test_df.pred.apply(lambda x: label2name.get(x))
```

In [83]:

```
from sklearn.metrics import confusion_matrix


# We create a confusion matrix to better observe the classes that the model confuses.
pred_name_values = test_df.pred_name.values
label_values = test_df.label_name.values
confmat = confusion_matrix(label_values, pred_name_values, labels=list(name2label.keys()))
```

In [84]:

```
confmat
```

Out[84]:

```
array([[ 0, 53, 19],
       [34,  0, 66],
       [ 6, 61,  0]])
```

In [85]:

```
df_confusion_test = pd.crosstab(label_values, pred_name_values)
df_confusion_test
```

Out[85]:

col_0	Negative	Neutral	Positive
row_0			

Negative	0	53	19
Neutral	34	0	66
Positive	6	61	0

6. References

1. Hugging Face
2. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
3. RoBERTa: A Robustly Optimized BERT Pretraining Approach
4. XLNet: Generalized Autoregressive Pretraining for Language Understanding
5. Coursera
6. Brand24
7. MonkeyLearn

If you like the notebook, Please don't forget to UPVOTE and comment :) :)