Project Name: Sentiment Analysis of comments.

Goal: To predict sentiment of comments using Natural Language Processing

Dataset: IIT - KGP KshitiJ AI Hackathon dataset

About the Project: With the increasing reliance on online platforms for reviews, feedback, and social media interactions, understanding customer sentiment has become a critical task for businesses across industries. Sentiment analysis allows companies to gain deeper insights into customer opinions, which can help improve products, services, and customer satisfaction. However, interpreting the true sentiment behind vast amounts of text data, such as reviews, comments, and social media posts, can be challenging due to the nuances of language, tone, and context.

To address this, we are developing a Natural Language Processing (NLP) model that analyzes customer feedback and classifies the sentiment expressed in the text. By training the model on a variety of text data with labeled sentiments, it will be able to accurately predict whether a given review or comment is positive, negative, or neutral. This tool will help businesses quickly identify areas of concern and opportunities for improvement by analyzing large volumes of customer feedback. The ultimate goal is to provide actionable insights that can enhance decision-making, improve customer relationships, and drive business success.

Section 1: Data Collection

In [1]:
```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
url1 = r"C:\Users\user\Desktop\Code Fragments\Machine Learning\Datasets\AI-Hackathon-test-data-set.csv"
data = pd.read_csv(url1)
data
```

Out[2]:

	"Comments,Sentiment"
0	"I love the new features in the update.,Positive"
1	"Fast and efficient customer support.,Positive"
2	"The product quality is amazing!,Positive"
3	"Fast and efficient customer support.,Positive"
4	"I am disappointed with the delivery.,Neutral"
...	...
19998	"Fast and efficient customer support,Positive"
19999	"The room was clean but noisy.,Neutral"
20000	"The product quality is amazing!,Positive"
20001	"I am disappointed with the delivery.,Negative"
20002	"The food was cold and tasteless.,Negative"

20003 rows × 1 columns

We have ~ 20000 rows in our dataset, but the dataset hasn't be converted into our desired dataframe format.

Let's try to manipulate it manually.

In [3]:
```
data["Sentiment"] = data["Comments,Sentiment"].apply(lambda x: x[-9:-1])
data["Comments"] = data["Comments,Sentiment"].apply(lambda x: x[1:-10])
data
```

Out[3]:

	"Comments,Sentiment"	Sentiment	Comments
0	"I love the new features in the update.,Positive"	Positive	I love the new features in the update.
1	"Fast and efficient customer support.,Positive"	Positive	Fast and efficient customer support.
2	"The product quality is amazing!,Positive"	Positive	The product quality is amazing!
3	"Fast and efficient customer support.,Positive"	Positive	Fast and efficient customer support.
4	"I am disappointed with the delivery.,Neutral"	Neutral	I am disappointed with the delivery.
...	...	...	...
19998	"Fast and efficient customer support,Positive"	Positive	Fast and efficient customer support
19999	"The room was clean but noisy.,Neutral"	Neutral	The room was clean but noisy
20000	"The product quality is amazing!,Positive"	Positive	The product quality is amazing!
20001	"I am disappointed with the delivery.,Negative"	Negative	I am disappointed with the delivery.
20002	"The food was cold and tasteless,Negative"	Negative	The food was cold and tasteless.

20003 rows × 3 columns

Now that we have both the comments & sentiments separated, we can drop the attached column.

In [4]:
```
data.drop(columns = ['"Comments,Sentiment"'], inplace = True)
data
```

Out[4]:

	Sentiment	Comments
0	Positive	I love the new features in the update.
1	Positive	Fast and efficient customer support.
2	Positive	The product quality is amazing!
3	Positive	Fast and efficient customer support.
4	Neutral	I am disappointed with the delivery.
...	...	...
19998	Positive	Fast and efficient customer support
19999	Neutral	The room was clean but noisy
20000	Positive	The product quality is amazing!
20001	Negative	I am disappointed with the delivery.
20002	Negative	The food was cold and tasteless.

20003 rows × 2 columns

Since our dataframe is ready, lets move on to our data manipulation phase.

Section 2: Data Cleaning / Manipulation

Initially, we can encode our sentiment values.

In [5]:
```
d = {
    "Positive" : 1,
    "Negative" : -1,
    "Neutral" : 0
}
data["Sentiment"] = data["Sentiment"].map(d)
```

Lets start with dealing with null values.

In [6]:
```
print(data[data["Comments"].isnull()])
print(data[data["Sentiment"].isnull()])
print(data[data['Comments'] == ''])
```

```
Empty DataFrame
Columns: [Sentiment, Comments]
Index: []
Empty DataFrame
Columns: [Sentiment, Comments]
Index: []
      Sentiment Comments
86            0
153          -1
188           1
381           1
392           1
...         ...      ...
19854        -1
19903        -1
19941        -1
19951        -1
19993        -1

[1985 rows x 2 columns]
```

In [7]:
```
data = data.data[data["Comments"]!= '']
```

As observed, our dataset has no null values.

One thing commonly observed in textual datasets is redundancy, lets deal with redundancy now.

In [8]:
```
data["Sentiment"].value_counts()
```

Out[8]:
```
Sentiment
 1    7661
-1    7358
 0    4183
Name: count, dtype: int64
```

As we can see, there is an deficit of neutral values. This may lead to underfitting / bias.

We have to deal with this imbalance.

In [9]:
```
data["Comments"].value_counts()
```

Out[9]:
```
Comments
I love the new features in the update.                    1827
The product quality is amazing!                           1811
Fast and efficient customer support.                      1810
The app interface is confusing.                           1801
I am disappointed with the delivery.                      1799
Excellent ambiance and friendly staff.                    1796
The food was cold and tasteless.                          1793
Very poor customer service experience.                    1793
The service was okay, not great.                          1712
The room was clean but noisy.                             1712
The room was clean but noisy.                              208
The service was okay, not great.                           193
I am disappointed with the delivery.                       106
Excellent ambiance and friendly staff                      104
Very poor customer service experience                      100
The food was cold and tasteless                             97
I love the new features in the update                       91
Fast and efficient customer support                         90
The app interface is confusing                              87
The product quality is amazing                              86
Blah blah blah.                                              1
Ambrandon-test-12                                            1
Name: count, dtype: int64
```

As we can see, our dataset is extremely poor in terms of redundancy & annotations.

With the occurence frequency in mind, the

Countering the problems -

- redundancy : we do not try to solve the redundancy since dropping duplicates greatly reduces data size
- annotations : for same data being annotated differently, we can adjust the annotation assigned to majority of the sub-data.
- imbalance : we train using 4k values each, and then fine tune while adding left over data.

Section 2.1 : Addressing Annotation Errors

In [10]:
```
unique_coms = data['Comments'].unique()
unique_coms
```

Out[10]:
```
array(['I love the new features in the update.',
       'Fast and efficient customer support.',
       'The product quality is amazing!',
       'I am disappointed with the delivery.',
       'The service was okay, not great.',
       'The food was cold and tasteless.',
       'I am disappointed with the delivery.',
       'Excellent ambiance and friendly staff.',
       'Very poor customer service experience.',
       'The room was clean but noisy.', 'The app interface is confusing.',
       'The room was clean but noisy.',
       'Excellent ambiance and friendly staff',
       'Very poor customer service experience',
       'The product quality is amazing',
       'The app interface is confusing',
       'The service was okay, not great.',
       'Fast and efficient customer support',
       'The app interface is confusing',
       'The product quality is amazing', 'Blah blah blah.',
       'Ambrandon-test-12'], dtype=object)
```

We will proceed with removing the annotation inconsistency, by selecting the one with higher frequency.

In [11]:
```
for i in unique_coms:
    f1 = len(data[data["Comments"] == i])
    f2 = len(data[data["Comments"] == i+"."])
    if f1<f2:
        data.drop(data[data["Comments"] == i+"."].index, inplace=True)
    else:
        data.drop(data[data["Comments"] == i].index, inplace=True)
```

In [12]:
```
data['Comments'].unique()
```

Out[12]:
```
array(['I love the new features in the update.',
       'Fast and efficient customer support.',
       'The product quality is amazing!',
       'The service was okay, not great.',
       'The food was cold and tasteless.',
       'I am disappointed with the delivery.',
       'Excellent ambiance and friendly staff.',
       'Very poor customer service experience.',
       'The room was clean but noisy.', 'The app interface is confusing.',
       'The room was clean but noisy.',
       'Excellent ambiance and friendly staff',
       'Very poor customer service experience',
       'The product quality is amazing',
       'The app interface is confusing',
       'The service was okay, not great.',
       'Fast and efficient customer support',
       'The app interface is confusing',
       'The product quality is amazing', 'Blah blah blah.',
       'Ambrandon-test-12'], dtype=object)
```

Now that our annotation issues are clear, we can move on data manipulation techniques for textual data.

We do the following:

- remove stop words
- lemmatize
- basic formatting

In [13]:
```
import re
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def process(text):
    text = text.lower()
    text = re.sub(r'[^a-z]', ' ', text)
    words = text.split()
    words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words]
    return ' '.join(words)

data["Comments"] = data["Comments"].apply(lambda x: process(x))
```

we create three parts of the dataset with different sentiments

In [14]:
```
pos_data = data[data["Sentiment"] == 1]
neg_data = data[data["Sentiment"] == -1]
net_data = data[data["Sentiment"] == 0]
```

We merge the three data parts into one, while leaving out parts of positive & negative data to ensure equal representation to neutral data

In [15]:
```
use_pos = pos_data[:4000]
use_neg = neg_data[:4000]
data1 = pd.concat([use_pos, use_neg, net_data])
```

We shuffle the data to maintain equal representation of data in all parts

[Concept of Stratified Shuffle Split]

In [16]:
```
from sklearn.utils import shuffle
data1 = shuffle(data1, random_state = 42)
data1
```

Out[16]:

	Sentiment	Comments
7785	-1	app interface confusing
7532	1	fast efficient customer support
9761	-1	disappointed delivery
9807	0	product quality amazing
11814	0	room clean noisy
...	...	...
18833	0	room clean noisy
3345	-1	app interface confusing
3865	-1	food cold tasteless
2255	1	app interface confusing
9132	-1	food cold tasteless

11510 rows × 2 columns

In [17]:
```
x = data["Comments"]
y = data["Sentiment"]

x_train = x[:int(len(x)*0.75)]
y_train = y[:int(len(y)*0.75)]
x_test = x[int(len(x)*0.75):]
y_test = y[int(len(y)*0.75):]
```

In [18]:
```
x_train[5]
```

Out[18]:
```
'fast efficient customer support'
```

Our dataset is now ready for training phase 1.

Section 3: Model selection

Google's bert is a very popular & reliable model.

We will process with it's use for our model.

In [19]:
```
from transformers import BertTokenizer as BT, TFBertForSequenceClassification as tbsc
from sklearn.metrics import classification_report as cr
```

```
WARNING:tensorflow:From C:\Users\user\anaconda3\lib\site-packages\tf_keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

We use TFBertSC for our classification task and BertTokenizer of tokenization of text.

In [20]:
```
tokenizer = BT.from_pretrained('bert-base-uncased', do_lower_case=True)

x_train_en = tokenizer.batch_encode_plus(x_train.astype(str).tolist(),
                                         padding=True,
                                         truncation=True,
                                         max_length = 128,
                                         return_tensors='tf')

x_test_en= tokenizer.batch_encode_plus(x_test.astype(str).tolist(),
                                       padding=True,
                                       truncation=True,
                                       max_length = 128,
                                       return_tensors='tf')
```

In [21]:
```
model = tbsc.from_pretrained('bert-base-uncased', num_labels=3)
```

```
WARNING:tensorflow:From C:\Users\user\anaconda3\lib\site-packages\tf_keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

All PyTorch model weights were used when initializing TFBertForSequenceClassification.

Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification were not initialized from the PyTorch model and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

As we have three classifications, we set num_labels to 3.

This allows the model to predict values between [0-3)

To avoid errors, we add 1 to all values of y_train & y_test (since minimum value is -1)

In [22]:
```
y_train = y_train + 1
y_test = y_test + 1
```

Now, we compile our model using an optimizer, specifying criteria of loss & accuracy.

In [23]:
```
import tensorflow as tf
optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')
model.compile(optimizer=optimizer, loss=loss, metrics=[metric])
```

We finally fit our data into the model.

In [24]:
```
'''
model.fit(
    [x_train_en['input_ids'], x_train_en['token_type_ids'], x_train_en['attention_mask']],
    y_train,
    validation_data=(
        [x_test_en['input_ids'], x_test_en['token_type_ids'], x_test_en['attention_mask']],y_test),
    batch_size=32,
    epochs=1
)
'''
```

Out[24]:
```
"\nmodel.fit(\n    [x_train_en['input_ids'], x_train_en['token_type_ids'], x_train_en['attention_mask']],\n    y_train,\n    validation_data=(\n        [x_test_en['input_ids'], x_test_en['token_type_ids'], x_test_en['attention_mask']],y_test),\n    batch_size=32,\n    epochs=1\n)\n"
```

Since the training time is large, and no availability of an accelerator in this environment,

the model has been trained using a GPU in a kaggle notebook

(https://www.kaggle.com/code/gouranginarula/notebook86c5597d7a/notebook?scriptVersionId=218325013)

as an continuation to this project.

The next few code blocks would just be repetition of a few steps from the original notebook,
Please skip to continue from the new additions

In [1]:
```
import pandas as pd
import numpy as np
data = pd.read_csv("/kaggle/input/aihsentiment/AI-Hackathon-test-data-set-3.csv")
pos_data = data[data["Sentiment"] == 1]
neg_data = data[data["Sentiment"] == -1]
net_data = data[data["Sentiment"] == 0]
use_pos = pos_data[:4000]
use_neg = neg_data[:4000]
data1 = pd.concat([use_pos, use_neg, net_data])
from sklearn.utils import shuffle
data1 = shuffle(data1, random_state = 42)
x = data1["Comments"]
y = data1["Sentiment"]

x_train = x[:int(len(x)*0.75)]
y_train = y[:int(len(y)*0.75)]
x_test = x[int(len(x)*0.75):]
y_test = y[int(len(y)*0.75):]
from transformers import BertTokenizer as BT, TFBertForSequenceClassification as tbsc
from sklearn.metrics import classification_report as cr
tokenizer = BT.from_pretrained('bert-base-uncased', do_lower_case=True)

x_train_en = tokenizer.batch_encode_plus(x_train.astype(str).tolist(),
                                         padding=True,
                                         truncation=True,
                                         max_length = 128,
                                         return_tensors='tf')

x_test_en= tokenizer.batch_encode_plus(x_test.astype(str).tolist(),
                                         padding=True,
                                         truncation=True,
                                         max_length = 128,
                                         return_tensors='tf')
model = tbsc.from_pretrained('bert-base-uncased', num_labels=3)
y_train = y_train + 1
y_test = y_test + 1
import tensorflow as tf
optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')
model.compile(optimizer=optimizer, loss=loss, metrics=[metric])
```

```
tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]
vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]
model.safetensors:   0%|          | 0.00/440M [00:00<?, ?B/s]
```

```
All PyTorch model weights were used when initializing TFBertForSequenceClassification.

Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification were not initialized from the PyTorch model and are newly initialized: ['classif
ier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

This is where the continuation of the notebook starts from

We start with the model training

In [2]:
```
model.fit(
    [x_train_en['input_ids'], x_train_en['token_type_ids'], x_train_en['attention_mask']],
    y_train,
    validation_data=(
      [x_test_en['input_ids'], x_test_en['token_type_ids'], x_test_en['attention_mask']],y_test),
    batch_size=32,
    epochs=3
)
```

```
Epoch 1/3
286/286 [==============================] - 70s 109ms/step - loss: 0.2305 - accuracy: 0.9387 - val_loss: 0.1629 - val_accuracy: 0.9524
Epoch 2/3
286/286 [==============================] - 28s 99ms/step - loss: 0.1464 - accuracy: 0.9594 - val_loss: 0.1573 - val_accuracy: 0.9573
Epoch 3/3
286/286 [==============================] - 29s 102ms/step - loss: 0.1458 - accuracy: 0.9578 - val_loss: 0.1622 - val_accuracy: 0.9524
```

Out[2]:
```
<tf_keras.src.callbacks.History at 0x7915b8aa3e50>
```

As the evaluation shows, our model gains an accuracy of about ~96%.
Lets evaluate our model on the test data.

In [3]:
```
test_loss, test_accuracy = model.evaluate(
    [x_test_en['input_ids'], x_test_en['token_type_ids'], x_test_en['attention_mask']],
    y_test
)
print(f'Test loss: {test_loss}, Test accuracy: {test_accuracy}')
```

```
96/96 [==============================] - 3s 32ms/step - loss: 0.1622 - accuracy: 0.9524
Test loss: 0.1621810644865036, Test accuracy: 0.9523965716362
```

As we can see, our model performs well on the test data as well.
We can proceed with the use of this model for our goal.

Section 4: User interactive space

Here, we allow the model to be used for different data.

Note: The performance of this model on real world data is extremely poor,
due to the source dataset having effectively only 22 data points.
As the motive of the hackathon's organizers was to test the preprocessing abilities of the participants, this model's performance is relevant in the scope of evaluating against the dataset itself.

In [4]:
```
while True:
    print("Enter the array of reviews you want to predict.")
    arr = list(input().rstrip().split(", "))
    df = pd.DataFrame(arr, columns=['Comments'])
    df_en = tokenizer.batch_encode_plus(x_train.astype(str).tolist(),
                                        padding=True,
                                        truncation=True,
                                        max_length = 128,
                                        return_tensors='tf')
    pred = model.predict(
        [df_en['input_ids'],df_en['token_type_ids'], df_en['attention_mask']])
    logits = pred.logits
    pred_labels = tf.argmax(logits, axis=1)
    pred_labels = pred_labels.numpy()
    label = {
        2: 'Positive',
        0: 'Negative',
        1: 'Neutral'
    }
    pred_labels = [label[i] for i in pred_labels]
    print("Predicted sentiments are - \n")
    for i in range(len(arr)):
        print(arr[i],":",pred_labels[i])
    print("Do you wish to continue?\n1 - Yes\n2 - No")
    if int(input()) == 2:
        break

print("Thank you.")
```

```
Enter the array of reviews you want to predict.
286/286 [==============================] - 12s 33ms/step
Predicted sentiments are -

Good product : Positive
Do you wish to continue?
1 - Yes
2 - No
Thank you.
```

This marks the end of our project.

Conclusion

This project is aimed at determining the sentiment of reviews of products accurately.
The true core challenge of the project was to counter the challenges faced by having a poor dataset.
Challenges faces in the dataset -

- Heavy redundancy (>98%), causing no real input value in the dataset (the dataset had 22 rows post dropping duplicate values, which originally had 20000 rows)
- Wrong, with low variety in data in the first place, different instances of identical features having different label added to the loss of value in the data

Due to the above reasons (very low variety in data), the model trained on the same is essentially not of practical use.

With the challenge being to make the model accurate for data within the dataset itself,
our model gained an accuracy of ~96%, which originally hit an bottleneck accuracy of 60% prior to all preprocessing ideas being implemented. With that, the goal has been met & the project is complete.

What makes this project different (& better)? -> Accuracy of ~95% compared to the bottleneck average of 60%

What was the main factor behind the accuracy increase?
Resolving massive data annotation errors in the original dataset.

Due to the inconsistent annotation, fitting the model with the entire dataset couldn't reproduce a high accuracy. The reason being the same data being present multiple times, with different annotations.