```
!nvidia-smi
```

```
Thu Aug 31 22:28:13 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   57C    P8    10W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

HW1-part-B-ZeroShot Classification- 8 Points

- **You have to submit two files for this part of the HW**

  > (1) ipynb (colab notebook) and
  > (2) pdf file (pdf version of the colab file).**

- **Files should be named as follows**:

  > FirstName_LastName_HW_1B**

Question: Zero Shot Classification using Huggingface Pipeline - 8 Points.

This task involves practical application of zero-shot classification to categorize restaurant review sentences. By leveraging a pre-trained model , students will gain hands-on experience in utilizing natural language inference techniques for text classification.

Zero Shot Classification - Brief Overview

Introduction to Natural Language Inference (NLI)

Natural Language Inference (NLI) is a task in natural language processing where a model determines the relationship between two sentences, often referred to as the premise and the hypothesis. The relationship might be **entailment (the hypothesis follows from the premise)**, contradiction, or neutrality.

Zero-Shot Classification Using Hugging Face's Pipeline

Building on NLI, Hugging Face's zero-shot classification leverages entailment (hypotheses follows from the premise) relationships to classify text into various categories.

1. **Premise:** The input text, like a restaurant review.
2. **Labels**:
   Possible labels (categories) represent distinct classifications within a given problem. In the context of the review dataset that you will be working with, the possible labels correspond to specific aspects of a dining experience. The labels include: 'food,' referring to the quality and taste of the dishes; 'ambiance,' reflecting the overall atmosphere and decor of the restaurant; 'service,' pertaining to the staff's attentiveness and professionalism; and 'other,' a category that encompasses any additional comments or observations not covered by the aforementioned labels.
3. **Hypothesis Template:**
   A specially formatted string that transforms each label into an NLI-style hypothesis. An example of hypothesis template could be "**This example is {}.**". The {} is a placeholder where the candidate label is inserted, allowing the model to consider each label as a potential classification.
4. **Forming Hypotheses using template and labels:**
   Based on the label examples and hypothesis template, the resulting hypotheses will be:
   - "This example is food."
   - "This example is ambiance."

- "This example is service."
- "This example is other."

5. **Zero-Shot Classification Process:**
   - **Step 1:** Combine premise and hypotheses.
   - **Step 2:** Utilize a pre-trained NLI model to calculate entailment probabilities (probability that hypothesis follows from premise) for each hypothesis.
   - **Step 3:** Return the probabilities for each label.

Example

Given a review like "The food was excellent," the model might return:

- food: 0.95
- ambiance: 0.05
- service: 0.02
- other: 0.023

The probability of 0.95 for the label "food" means that the model has a 95% confidence that the premise "The food was excellent" entails the hypothesis "This example is food." The other probabilities represent the model's confidence for the remaining labels.

Conclusion

Hugging Face's zero-shot classification pipeline calculates probabilities for multiple categories by leveraging NLI concepts. This approach offers flexibility in text analysis without the need for task-specific training, providing insights into the relevance of different labels to a given text.

Data and Task Description

- For this Question, you are provided a csv file (review_sentences.csv) that has 584 sentences from restautant reviews from Yelp. You have to classify the sentences in to four labels: `["food," "ambiance," "service," and "other."]`.
- The csv file is availble in eLearning in 0_Data folder.
- You will use a zero-shot-classification pipleine from huggingface to make predictions.
- You are also provided the actual labels. You will use these labels to access the accuracy of the model.

Install/Import Modules

```
if 'google.colab' in str(get_ipython()):
    !pip install transformers -U -qq
    !pip install sentencepiece -U -qq
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.5/7.5 MB 13.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 268.8/268.8 kB 23.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 31.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 41.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 9.7 MB/s eta 0:00:00
```

```
# Import the pandas library for data manipulation and analysis
import pandas as pd

# Import the numpy library for numerical computing
import numpy as np

# Import the Matplotlib library for creating visualizations such as plots, graphs, etc.
import matplotlib.pyplot as plt

# Import the pathlib library for working with file paths in a way that is cross-platform
from pathlib import Path

# Import functions for metrics computation like confusion matrix, and accuracy
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_score, classification_report

# Import the transformers library for state-of-the-art Natural Language Processing (NLP) models like BERT, GPT, etc.
from transformers import pipeline
```

```
!pip show transformers
```

```
Name: transformers
Version: 4.32.1
Summary: State-of-the-art Machine Learning for JAX, PyTorch and TensorFlow
Home-page: https://github.com/huggingface/transformers
```

```
Author: The Hugging Face team (past and future) with the help of all our contributors (https://github.com/huggingface/transformers/
Author-email: transformers@huggingface.co
License: Apache 2.0 License
Location: /usr/local/lib/python3.10/dist-packages
Requires: filelock, huggingface-hub, numpy, packaging, pyyaml, regex, requests, safetensors, tokenizers, tqdm
Required-by:
```

Specify Base folder for Project

```
if 'google.colab' in str(get_ipython()):
  from google.colab import drive
  drive.mount('/content/drive')
  base_folder = Path('/content/drive/MyDrive/NLP') # MAKE SURE TO CHANGE THE PATH

# (the else block is required only if you have local GPU machine, other wise you can ignore the else block)
else:
  base_folder = Path('/home/harpreet/Insync/google_drive_shaannoor/data') # MAKE SURE TO CHANGE THE PATH
```

```
Mounted at /content/drive
```

Create DataFrame

```
data_folder = base_folder/'Datasets'  # MAKE SURE TO CHANGE THE PATH
```

```
# location of train and test files
data_file = data_folder /'review_sentences.csv'
```

```
# creating Pandas Dataframe
train_data = pd.read_csv(data_file, index_col=0, encoding='ISO-8859-1')
```

```
# print shape of the datasets
print(f'Shape of Training data set is : {train_data.shape}')
```

```
Shape of Training data set is : (584, 2)
```

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 584 entries, 0 to 599
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   text         584 non-null    object
 1   final_label  584 non-null    object
dtypes: object(2)
memory usage: 13.7+ KB
```

```
# check first five examples
train_data.head()
```

sentences	text	final_label
0	If you love unagi (eel) on rice, you'll absolu...	food
1	But it is definitely worth the wait.	service
3	Sometimes get the Pho w/ rare beef which is de...	food
4	Other items on this menu was a crab soup, duck...	food
5	I visited the restaurant for lunch and arrived...	other

```
# check disytribution of labels
train_data['final_label'].value_counts()
```

```
food        279
other       139
service     120
ambiance     46
Name: final_label, dtype: int64
```

Load Pipeline

```
# Load zero shot classification pipleine - do not pass any model. We will use the default model
# use pytorch as framework and move the pipleine to GPU
# syntax : zero_shot_classifier = pipeline(name-of-pipeline, framework=, device =)

zero_shot_classifier = pipeline("zero-shot-classification" , framework = 'pt' , device = 0)
```

```
No model was supplied, defaulted to facebook/bart-large-mnli and revision c626438 (https://huggingface.co/facebook/bart-large-mnli)
Using a pipeline without specifying a model name and revision in production is not recommended.
Downloading (…)lve/main/config.json: 100%  1.15k/1.15k [00:00<00:00, 69.1kB/s]
Downloading model.safetensors: 100%  1.63G/1.63G [00:15<00:00, 192MB/s]
Downloading (…)okenizer_config.json: 100%  26.0/26.0 [00:00<00:00, 2.03kB/s]
Downloading (…)olve/main/vocab.json: 100%  899k/899k [00:00<00:00, 13.4MB/s]
Downloading (…)olve/main/merges.txt: 100%  456k/456k [00:00<00:00, 6.39MB/s]
Downloading (…)/main/tokenizer.json: 100%  1.36M/1.36M [00:00<00:00, 37.1MB/s]
```

Task1 : Base experiment.

- **In this Task you will pass all the four labels to the classifier.**

Create a list of reviews

```
# create a list of reviews (hint use column.tolist() to convert numpy array series to python list)
texts_train =   train_data['text'].tolist()

# list of labels
candidate_labels = ['food', 'ambiance' , 'service' , 'other']
```

Get Probbailities

Pass the list of text and labels you created in the previous step to the classifier to get predictions.

The syntax is : **`probs = zero_shot_classifier(sequences= , candidate_labels = )`**

```
probs = zero_shot_classifier(sequences = texts_train , candidate_labels = candidate_labels)
```

```
probs[0: 3]
```

```
[{'sequence': "If you love unagi (eel) on rice, you'll absolutely enjoy their version.",
  'labels': ['food', 'ambiance', 'service', 'other'],
  'scores': [0.9423722624778748,
   0.023495187982916832,
   0.021316785365343094,
   0.012815762311220169]},
 {'sequence': 'But it is definitely worth the wait.',
  'labels': ['other', 'service', 'ambiance', 'food'],
  'scores': [0.4905829429626465,
   0.2534791827201843,
   0.20231030881404877,
   0.05362754687666893]},
 {'sequence': 'Sometimes get the Pho w/ rare beef which is decent.',
  'labels': ['food', 'other', 'service', 'ambiance'],
  'scores': [0.7790170311927795,
   0.10868176817893982,
   0.08358323574066162,
   0.02871803753077984]}]
```

```
# DO NOT RUN THIS CELL - This cells gives you an idea of the expected output
```

```
[{'sequence': "If you love unagi (eel) on rice, you'll absolutely enjoy their version.",
  'labels': ['food', 'ambiance', 'service', 'other'],
  'scores': [0.9423722624778748,
   0.023495187982916832,
   0.021316785365343094,
   0.012815762311220169]},
 {'sequence': 'But it is definitely worth the wait.',
  'labels': ['other', 'service', 'ambiance', 'food'],
  'scores': [0.4905829429626465,
   0.2534791827201843,
   0.20231030881404877,
```

```
  0.05362754687666893]},
 {'sequence': 'Sometimes get the Pho w/ rare beef which is decent.',
  'labels': ['food', 'other', 'service', 'ambiance'],
  'scores': [0.7790170311927795,
   0.10868176817893982,
   0.08358323574066162,
   0.02871803753077984]}]
```

Get Predictions

- The output from the classifier will give you the probbaility for each label
- The label with the highest probbaility should be your prediction
- You might need to use more than one line of the code to complete this step

```
predictions = [review['labels'][np.argmax(review['scores'])] for review in probs]
```

```
# get first five preeictions
predictions[0:5]
```

```
['food', 'other', 'food', 'food', 'food']
```

```
# DO NOT RUN THIS CELL - This cells gives you an idea of the expected output
```

Accuracy

- You might need to use more than one line of the code to complete this step
- Now you have the actual Label and predicted Label for each sentence.
- Calculate the overall acacuracy (Hint : you can use accuracy_score form from sklearn - `from sklearn.metrics import accuracy_score`

```
final_labels_list = train_data['final_label'].tolist()
accuracy =  accuracy_score(final_labels_list, predictions)*100
accuracy
```

```
66.6095890410959
```

```
# DO NOT RUN THIS CELL - If you have done everything correctly, you should get accurtacy close to reported below
accuracy
```

```
66.6095890410959
```

Classification Report

- Print the classification report
- HINT- `from sklearn.metrics import classification_report`

```
# print classification report
class_report =  classification_report(final_labels_list , predictions)
print(class_report)
```

```
                precision    recall  f1-score   support

     ambiance       0.39      0.33      0.36        46
         food       0.76      0.84      0.80       279
        other       0.52      0.54      0.53       139
      service       0.69      0.54      0.61       120

     accuracy                           0.67       584
    macro avg       0.59      0.56      0.57       584
 weighted avg       0.66      0.67      0.66       584
```

```
# DO NOT RUN THIS CELL - If you have done everything correctly, your report should look similar
```

```
                precision    recall  f1-score   support

     ambiance       0.39      0.33      0.36        46
         food       0.76      0.84      0.80       279
        other       0.52      0.54      0.53       139
      service       0.69      0.54      0.61       120

     accuracy                           0.67       584
    macro avg       0.59      0.56      0.57       584
 weighted avg       0.66      0.67      0.66       584
```

Conclusion from the classification report

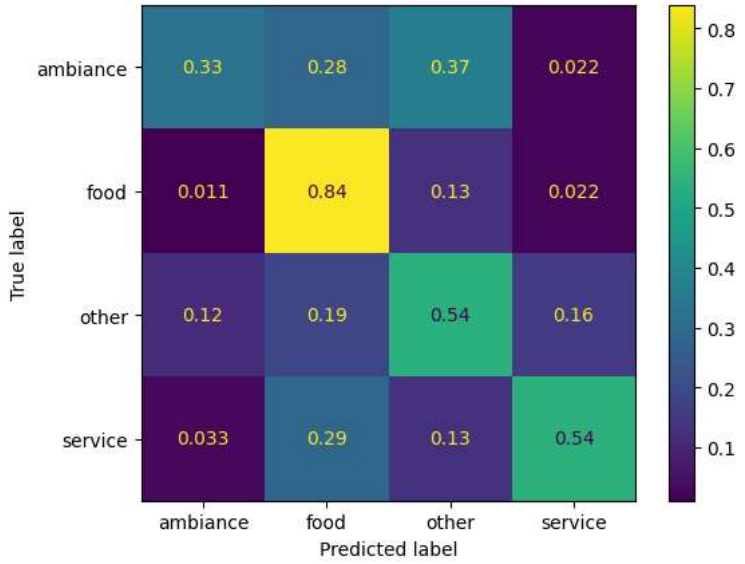
What do you conclude from the classification report?

The model is able to classify 84% of the food review from the total actual food review as compared to the other labels. The accuracy of positive predictions is low for the ambiance labels as compared to food, other, and service. The Overall accuracy of the model is 67% and the food labels have higher precision and recall than the other labels.

Confusion Matrix

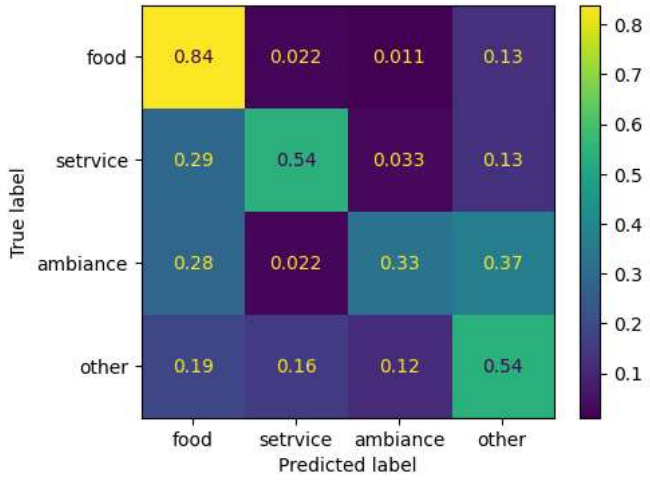
- You might need to use more than one line of the code to complete this step
- Print confusionm matrix
- Hint:use ConfusionMatrixDisplay.from_predictions from sklearn

```
labels = ['ambiance', 'food', 'other', 'service']
cm = confusion_matrix(final_labels_list, predictions , normalize='true')
disp = ConfusionMatrixDisplay(confusion_matrix=cm , display_labels=labels)
disp.plot()
plt.show()
```




```
# DO NOT RUN THIS CELL - this gives you an idea of expected output. Your results might differ slightly
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f3bcdf6bcd0>
```



Conclusions from the Confusion Matrix

What do you conclude from the Confusion Matrix?

The model is able to correctly predict about 84% of food reviews. 37% of ambiance reviews are predicted as other reviews by the model. The model wrongly predicted 29% of service reviews as food reviews.

Task2 : Change hypothesis template.

In this experiment, we are exploring the impact of changing the hypothesis template in a zero-shot classification model. The default template used in the pipeline is "**This example is {}**". In the experiment the default template is being replaced with a domain-specific template to classify restaurant reviews into categories like food, service, ambiance, and other.

Steps:

1. **Modify Hypothesis Template:** Adjust the template to better fit the context of restaurant reviews.
2. **Evaluate Model:** Apply the modified template and report accuracy, a classification report, and confusion matrix.
3. **Compare Results:** Analyze the findings in comparison to the previous experiment using the default template.

Conclusion:

Changing the hypothesis template may lead to variations in model performance, possibly improving accuracy in certain categories. The comparison between the experiments can reveal insights into how the model interprets premises differently with various templates, helping to tailor the model to specific tasks or domains.

Create a list of reviews

```
# create a list of reviews (hint use column.tolist() to convert pandas series to python list)
texts_train =  train_data['text'].tolist()

# list of labels
candidate_labels =  ['ambiance', 'food', 'other', 'service']
```

Get Probbailities

In the previous experiment, you only passed sequences and candiadte labels. Since you did not pass any hypothesis template, the classifier used the default template "This example is {}." Now, in this experiment, you will introduce a custom hypothesis template to replace the default. The suggested custom template is "This review is related to the restaurant's {}." By specifying a template more aligned with the context of restaurant reviews, you may influence how the classifier interprets the relationship between the premise and each candidate label. Feel free to explore other custom templates that may be suitable for this task, and compare how these changes affect the classification performance.

The syntax is : **`zero_shot_classifier(sequences= , candidate_labels = , hypothesis_template = )`**

```
# pass the sequences, candiadte labels and hypothesis_template to the classifier
probs = zero_shot_classifier(sequences= texts_train , candidate_labels= candidate_labels , hypothesis_template = "This review is related
```

```
probs[0:3]
```

```
[{'sequence': "If you love unagi (eel) on rice, you'll absolutely enjoy their version.",
  'labels': ['food', 'other', 'service', 'ambiance'],
  'scores': [0.8286234140396118,
   0.1660204827785492,
   0.002787330187857151,
   0.0025688130408525467]},
 {'sequence': 'But it is definitely worth the wait.',
  'labels': ['other', 'food', 'service', 'ambiance'],
  'scores': [0.3562210202217102,
   0.30787432193756104,
   0.24831949174404144,
   0.08758512139320374]},
 {'sequence': 'Sometimes get the Pho w/ rare beef which is decent.',
  'labels': ['food', 'other', 'ambiance', 'service'],
  'scores': [0.809067964553833,
   0.18316298723220825,
   0.004174679517745972,
   0.003594418754801154]}]
```

Get Predictions

- The output from the classifier will give you the probbaility for each label
- The label with the highest probbaility should be your prediction
- You might need to use more than one line of the code to complete this step

```
predictions = [review['labels'][np.argmax(review['scores'])] for review in probs]
```

```
# get first five preeictions
predictions[0:5]
```

```
['food', 'other', 'food', 'food', 'other']
```

Accuracy

- You might need to use more than one line of the code to complete this step
- Now you have the actual Label and predicted Label for each sentence.
- Calculate the overall acacuracy (Hint : you can use accuracy_score form from sklearn - `from sklearn.metrics import accuracy_score`

```
accuracy =  accuracy_score(final_labels_list, predictions)*100
accuracy
```

```
78.76712328767124
```

```
# DO NOT RUN THIS CELL - If you have done everything correctly, you should get accurtacy close to reported below
accuracy
```

```
78.76712328767124
```

Classification Report

- Print the classification report
- HINT- `from sklearn.metrics import classification_report`

```
# print classification report
class_report =  classification_report(final_labels_list , predictions)
print(class_report)
```

```
                  precision    recall  f1-score   support

       ambiance       0.90      0.41      0.57        46
           food       0.88      0.84      0.86       279
          other       0.60      0.91      0.72       139
        service       0.93      0.68      0.78       120

       accuracy                           0.79       584
      macro avg       0.83      0.71      0.73       584
   weighted avg       0.83      0.79      0.79       584
```

```
# DO NOT RUN THIS CELL - If you have done everything correctly, your report should look similar
```

```
                  precision    recall  f1-score   support

       ambiance       0.90      0.41      0.57        46
           food       0.88      0.84      0.86       279
          other       0.60      0.91      0.72       139
        service       0.93      0.68      0.78       120

       accuracy                           0.79       584
      macro avg       0.83      0.71      0.73       584
   weighted avg       0.83      0.79      0.79       584
```

Compare the classification report

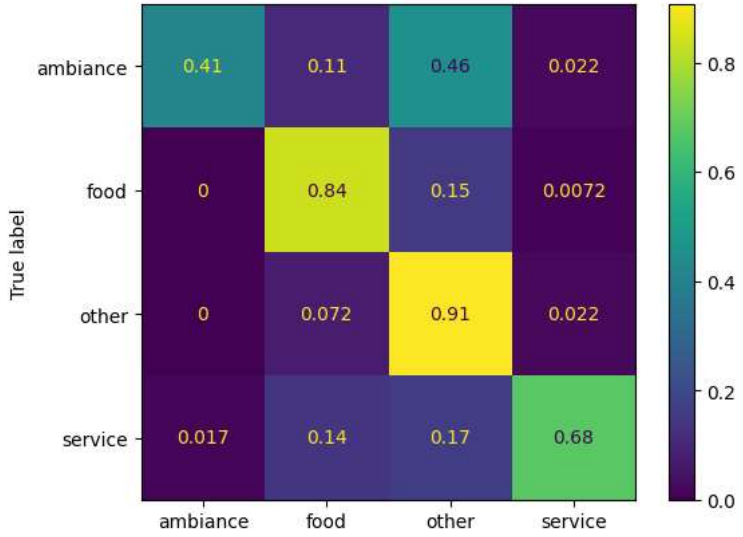
Compare the classification report with previous experiment and provide your conclusion.

The accuracy of positive predictions is high for ambiance, food and service with precision greater than 0.8. The new hypothesis has a higher accuracy than the previous hypothesis template. The precision-recall trade-off can be observed throught all the labels. All the labels which have high precision, have a low recall and vice versa.

Confusion Matrix

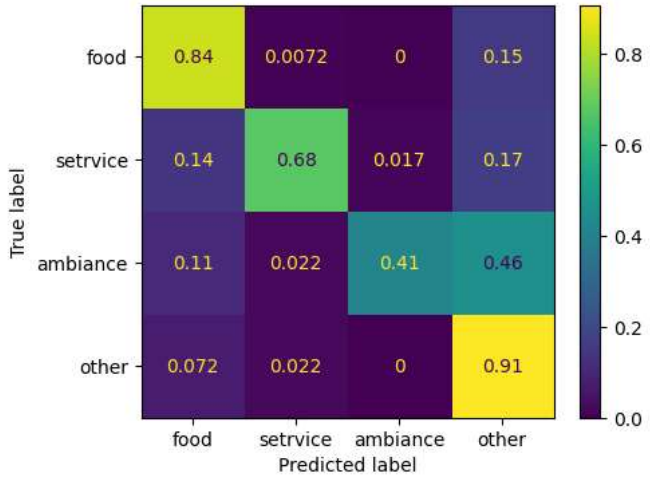
- You might need to use more than one line of the code to complete this step
- Print confusionm matrix
- Hint:use ConfusionMatrixDisplay.from_predictions from sklearn

```
cm = confusion_matrix(final_labels_list, predictions , normalize='true')
disp = ConfusionMatrixDisplay(confusion_matrix=cm , display_labels=labels)
disp.plot()
plt.show()
```




```
# DO NOT RUN THIS CELL - this gives you an idea of expected output. Your results might differ slightly
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f63984de980>
```



Conclusions from the Confusion Matrix

What do you conclude from the Confusion Matrix (compare with previous experiment)?

The model is able to correctly predict the 91% 'other' labels. The model predicted 46% of ambiance reviews as 'other'. 84% of food reviews are predicted correctly by the model. The new hypothesis model is more focused on the 'other' label than the previous hypothesis.

Task3 : Use threshold probbaility to classify Others.

In this next experiment, you will explore a nuanced approach to classification by **only considering the labels 'food', 'service', and 'ambiance'**. This is a departure from the previous experiments, where you also included 'other' as a label, and it brings some key distinctions:

Unlike traditional classification, where the probabilities across all labels sum to one, in this experiment, you'll assess the probabilities for just the three given labels.

By not including 'other' as a label, you are essentially allowing the model to classify instances into 'other' if none of the given labels strongly apply. If the maximum probability across 'food', 'service', and 'ambiance' is less than the threshold, the instance is classified as 'other'.

This approach not only recognizes that some instances may not fit neatly into one of the specific categories but also aims to refine the handling of those instances. It encourages a more flexible and discerning classification, possibly reducing misclassification.

The task will allow you to explore how this method of classification, guided by a threshold, can impact the model's performance. It offers an insightful comparison with the previous approaches and underlines the significance of understanding the nature of your data, the relationships between categories, and selecting an appropriate classification strategy accordingly.

Create a list of reviews

```
# create a list of reviews (hint use column.tolist() to convert pandas series to python list)
texts_train = train_data['text'].tolist()
```

```
# list of labels
candidate_labels = ['ambiance', 'food', 'service']
```

Get Probbailities

Pass the list of text and labels you craeted in the previous step to the classifier to get predictions.

```
# pass the sequences, candiadte labels and hypothesis_template to the classifier
probs = zero_shot_classifier(sequences= texts_train , candidate_labels= candidate_labels , hypothesis_template = "This review is related
```

```
probs[0: 5]
```

```
[{'sequence': "If you love unagi (eel) on rice, you'll absolutely enjoy their version.",
  'labels': ['food', 'service', 'ambiance'],
  'scores': [0.9935776591300964,
   0.0033422044944018126,
   0.0030801871325820684]},
 {'sequence': 'But it is definitely worth the wait.',
  'labels': ['food', 'service', 'ambiance'],
  'scores': [0.4782298803329468, 0.38572168350219727, 0.13604843616485596]},
 {'sequence': 'Sometimes get the Pho w/ rare beef which is decent.',
  'labels': ['food', 'ambiance', 'service'],
  'scores': [0.9904888272285461, 0.00511078629642725, 0.004400411155074835]},
 {'sequence': 'Other items on this menu was a crab soup, duck and clams congee, 2 desserts (one chinese and one western) and free
appetizers.',
  'labels': ['food', 'service', 'ambiance'],
  'scores': [0.9874481558799744, 0.006339206825941801, 0.006212593987584114]},
 {'sequence': 'I visited the restaurant for lunch and arrived there just after 12.30pm on a Sunday.',
  'labels': ['food', 'ambiance', 'service'],
  'scores': [0.7476781010627747, 0.13911722600460052, 0.11320469528436661]}]
```

Get Predictions

In this experiment , you'll be considering three specific categories: 'food', 'service', 'ambiance', and classify any instance that doesn't fit these categories as 'other'. To make this decision, you will use a concept called a threshold.

Understanding the Threshold Concept:

The threshold is a cutoff value that helps determine the class labels based on predicted probabilities. Here's how it works:

Imagine your model predicts the following probabilities for a review:

- 'food': 0.7
- 'service': 0.2
- 'ambiance': 0.1

With a threshold of 0.5, you check whether the highest probability (0.7 for 'food') is greater than this threshold. Since 0.7 > 0.5, the review is classified as 'food'. If the highest probability dees not surpass the threshold, you classify the review as 'other'. By adjusting the threshold, you control how confident the model must be to assign a particular label.

In this experiment, you will use a theshold of 0.8

```
threshold = 0.8
```

```
predictions = []
for review in probs:
  high_score = max(review['scores'])
  predict_label = review['labels'][np.argmax(review['scores'])]
  if high_score >= threshold:
    predictions.append(predict_label)
  else:
    predictions.append('other')
```

```
# get first five predictions
predictions[0:5]
```

```
['food', 'other', 'food', 'food', 'other']
```

Accuracy

- You might need to use more than one line of the code to complete this step
- Now you have the True Label and predicted Label for each sentence.
- Calculate the overall acacuracy (Hint : you can use accuracy_score form from sklearn - `from sklearn.metrics import accuracy_score`

```
accuracy =  accuracy_score(final_labels_list, predictions)*100
accuracy
```

```
76.02739726027397
```

```
# DO NOT RUN THIS CELL - If you have done everything correctly, you should get accurtacy close to reported below
accuracy
```

```
76.02739726027397
```

The overall accurtacy has decreased. This might be the result of chosen threshold

Classification Report

- Print the classification report
- HINT- `from sklearn.metrics import classification_report`

```
# print classification report
class_report =  classification_report(final_labels_list , predictions)
print(class_report)
```

```
                  precision    recall  f1-score   support

         ambiance       0.95      0.41      0.58        46
             food       0.91      0.85      0.88       279
            other       0.53      0.88      0.66       139
          service       0.93      0.53      0.68       120

         accuracy                           0.76       584
        macro avg       0.83      0.67      0.70       584
     weighted avg       0.83      0.76      0.76       584
```

```
# DO NOT RUN THIS CELL - If you have done everything correctly, your report should look similar
```

```
                  precision    recall  f1-score   support

         ambiance       0.95      0.41      0.58        46
             food       0.91      0.85      0.88       279
            other       0.53      0.88      0.66       139
          service       0.93      0.53      0.68       120

         accuracy                           0.76       584
        macro avg       0.83      0.67      0.70       584
     weighted avg       0.83      0.76      0.76       584
```

Compare the classification report

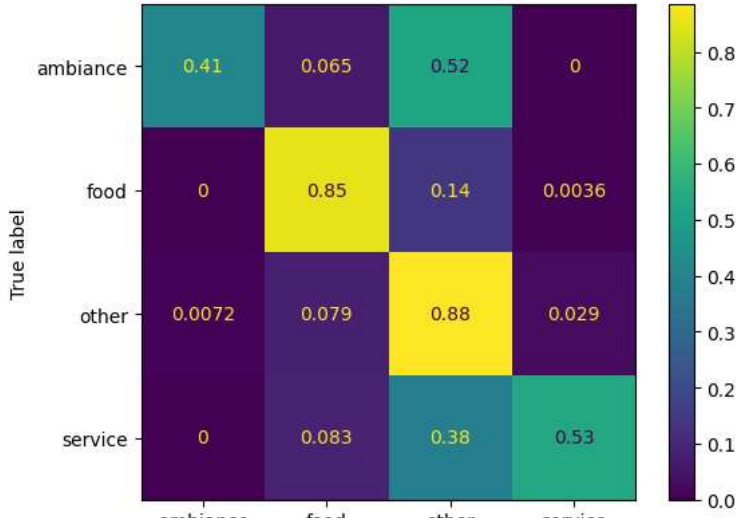
Compare the classification report with previous experiment and provide your conclusion.

The precision of the labels 'ambiance' and 'food' increased in the new model whereas, the 'service' label has the same precision. Even though, the service label has the same precision, the recall has a small increase in the new model. The accuracy of the new model is lower than the previous model.

Confusion Matrix

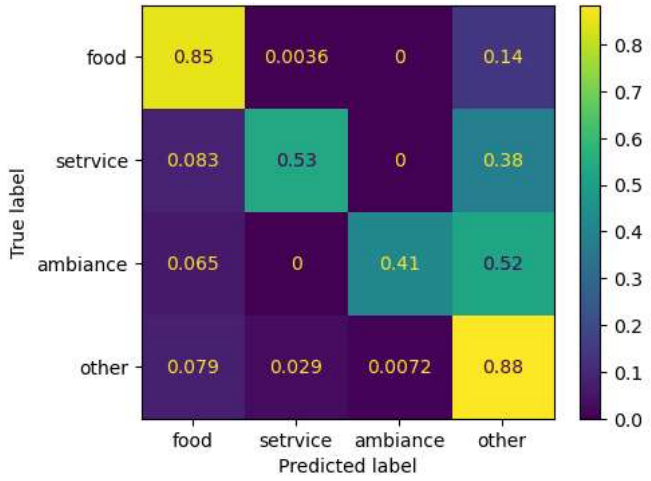
- You might need to use more than one line of the code to complete this step
- Print confusionm matrix
- Hint:`from sklearn.metrics import confusion_matrix``

```
cm = confusion_matrix(final_labels_list, predictions , normalize='true')
disp = ConfusionMatrixDisplay(confusion_matrix=cm , display_labels=labels)
disp.plot()
plt.show()
```




```
# DO NOT RUN THIS CELL - this gives you an idea of expected output. Your results might differ slightly
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff9c876bdf0>
```



Conclusions from the Confusion Matrix

What do you conclude from the Confusion Matrix (compare with previous experiments?

The model is able to correctly predict 88% of 'other' labels and 85% of 'food' labels. The model predicted 52% of ambiance reviews as 'other'. 38% of 'service' labels are predicted as 'other'.

BONUS TASK (Not Graded) - Function to choose threshold to maximize accuracy.

Let us assume that our business goal is to maximize accuracy. Write a function to determine the optimal threshold (maximum accuracy for classification) and to provide predictions that align with that threshold. You will also keep track of accuracy at each threshold level.

Pseudo Code:

```
1. Function predict_labels(probs, actual_labels, thresholds=[0.5]):
2.     Initialize best_threshold = 0
3.     Initialize best_accuracy = 0
4.     Initialize best_predictions = empty list
5.     Initialize accuracy_history = empty list
6.     FOR EACH threshold IN thresholds:
7.         Initialize predictions = empty list
8.         FOR EACH prob IN probs:
9.             Extract max_prob as the highest score among the probabilities in prob['scores']
10.            IF max_prob > current threshold THEN:
11.                Append the corresponding label to predictions (e.g., 'food')
12.            ELSE:
13.                Append 'other' to predictions
14.            END IF
15.        END FOR EACH prob
16.        Calculate accuracy using accuracy_score comparing predictions with actual_labels
```

```
17.        Append the accuracy to accuracy_history
18.        IF current accuracy > best_accuracy THEN:
19.            Update best_threshold, best_accuracy, and best_predictions
20.        END IF
21.    END FOR EACH threshold
22.    Return best_threshold, best_predictions and accuracy_history
23. END Function
```

```
def predict_labels(probs, actual_labels, thresholds):
  best_threshold = 0
  best_accuracy = 0
  best_predictions = []
  accuracy_history = []
  for threshold in thresholds:
    predictions = []
    for prob in probs:
      max_prob = max(prob['scores'])
      if max_prob > threshold:
        predictions.append([prob['labels'][np.argmax(prob['scores'])]])
      else:
        predictions.append('other')
      accuracy = accuracy_score(final_labels_list, predictions)*100
      accuracy_history.append(accuracy)
      if accuracy > best_accuracy:
        best_threshold = threshold
        best_predictions = predictions
        best_accuracy = accuracy
    return best_threshold, best_predictions, accuracy_history
```

Get best predictions and threshold

Use the function to get best_threshold, best_predictions and accuracy_history

```
thresholds=np.linspace(0.1, 0.95, 18)
thresholds
```

```
array([0.1 , 0.15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55, 0.6 ,
       0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95])
```

```
actual_labels = train_data['final_label'].values
```

```
(584,)
```

```
best_threshold, best_predictions, accuracy_history =  predict_labels(np.array(probs), actual_labels, thresholds)
```

```
best_threshold
```

```
# DO NOT RUN THIS CELL - your best threshold will be close to the result below
```

```
0.6
```

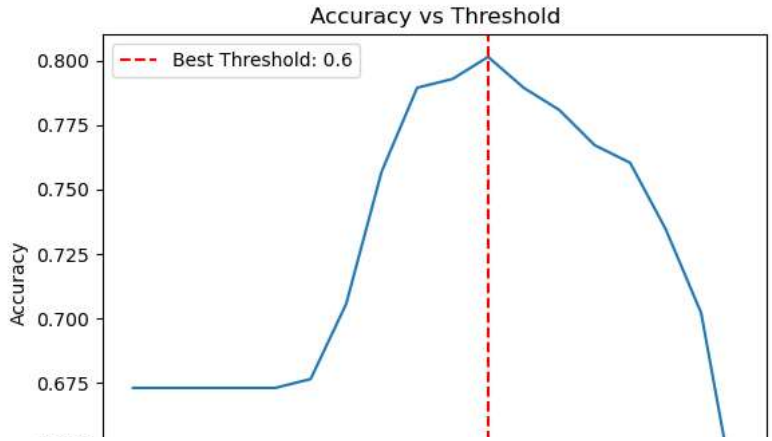
```
best_predictions[0:5]
```

Accuracy

Plot accuracy_history against thresholds

```
# CODE HERE
```

```
# DO NOT RUN THIS CELL - this gives you an idea of how the graog will look
```




```
# Best  Accuracy
accuracy = # CODE HERE
accuracy
```

```
# DO NOT RUN THIS CELL - If you have done everything correctly, you should get accurtacy close to reported below
```

```
80.13698630136986
```

Is the accuracy better than Task 2?

We can see that accuracy has slightly improved from Task2 (where we passed all four labels)

Confusion Matrix

```
# CODE HERE
```

```
# CODE HERE
```

```
# DO NOT RUN THIS CELL - this gives you an idea of expected output. Your results might differ slightly
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff9c85d0df0>
```