End to end prediction model

This notebook records all the methods of predicting slots and intents.

In [101…

```
from llama_cpp import Llama
import pandas as pd
import time
from IPython.display import Image

from tqdm.auto import tqdm


from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

from transformers import AutoModelForQuestionAnswering, AutoTokenizer, pipel

from groq import Groq
import json

from sentence_transformers import SentenceTransformer
```

title

In [43]:

```
tqdm.pandas()
```

In [2]:

```
# Groq API key
f = open("./data/credentials.json")
key = json.load(f)
client = Groq(api_key=key.get("GROQ_API_KEY",""))
```

Read preprocessed data

In `01_preprocessing.ipynb`, we have our train, dev, and test sets preprocessed.

In [3]:

```
train_file = "./data/train.csv"
train_df = pd.read_csv(train_file)

dev_filename = "./data/dev.csv"
dev_df = pd.read_csv(dev_filename)

test_filename = "./data/test.csv"
test_df = pd.read_csv(test_filename)

# create `text_lower` column for all dfs
train_df["text_lower"] = train_df["text"].str.lower()
dev_df["text_lower"] = dev_df["text"].str.lower()
test_df["text_lower"] = test_df["text"].str.lower()
```

```
# change `slot` column back to dictionary type
train_df["slots"] = train_df["slots"].progress_apply(lambda s: eval(s))
dev_df["slots"] = dev_df["slots"].progress_apply(lambda s: eval(s))

print(train_df.columns)
print(dev_df.columns)
print(test_df.columns)
```

```
Index(['text', 'answer', 'intent', 'slots', 'text_lower'], dtype='object')
Index(['text', 'answer_raw', 'answer', 'intent', 'slots', 'text_lower'], dty
pe='object')
Index(['text', 'text_lower'], dtype='object')
```

Step 1: Intent Classifier

for this module, we are using simple TFIDF Vectorizer and Logistic Regression for binary classification. The code in this section is inspired by `02_intent_classifier.ipynb`.

In [4]:

```
vectorizer = TfidfVectorizer()
x_train = train_df["text_lower"]
x_train_tfidf = vectorizer.fit_transform(x_train)
print(x_train_tfidf.shape)
```

```
(3760, 989)
```

In [5]:

```
# train using logistic regression
clf = LogisticRegression(random_state=42)
y_train = train_df["intent"]
print(y_train.shape)
clf.fit(x_train_tfidf, y_train)
print("Completed")
```

```
(3760,)
Completed
```

In [6]:

```
train_pred = clf.predict(x_train_tfidf)
print("Train score =", clf.score(x_train_tfidf, y_train))
print("Accuracy score =", accuracy_score(y_train, train_pred)) # should be t
```

```
Train score = 0.9973404255319149
Accuracy score = 0.9973404255319149
```

In [7]:

```
# dev evaluation
x_dev = dev_df["text_lower"]
x_dev_tfidf = vectorizer.transform(x_dev)
print(x_dev_tfidf.shape)
y_dev = dev_df["intent"]
dev_score = clf.score(x_dev_tfidf, y_dev)
dev_pred = clf.predict(x_dev_tfidf)
print("Dev accuracy =", accuracy_score(y_dev, dev_pred) * 100)
print("Dev accuracy =", dev_score * 100)
```

```
(413, 989)
Dev accuracy = 99.27360774818402
Dev accuracy = 99.27360774818402
```

In [8]:

```python
# Assign predictions to dfs
train_df["pred_intent"] = train_pred
dev_df["pred_intent"] = dev_pred

x_test = test_df["text_lower"]
x_test_tfidf = vectorizer.transform(x_test)
test_df["pred_intent"] = clf.predict(x_test_tfidf)

print(train_df.columns)
print(dev_df.columns)
print(test_df.columns)
```

```
Index(['text', 'answer', 'intent', 'slots', 'text_lower', 'pred_intent'], dt
ype='object')
Index(['text', 'answer_raw', 'answer', 'intent', 'slots', 'text_lower',
       'pred_intent'],
      dtype='object')
Index(['text', 'text_lower', 'pred_intent'], dtype='object')
```

Step 2: Hotel Star Extractor

for this module, we will be using RoBERTa question-answering model to extract star rating from context.

In [12]:

```python
model_name = "deepset/roberta-base-squad2"

roberta_qa_model = pipeline('question-answering', model=model_name, tokenize

def get_star(text, nlp_model=roberta_qa_model):
    """
        extract the star from the text
    """
    query = {
    'question': 'What is the star rating?',
    'context': text
    }
    res = nlp_model(query)

    result = res["answer"]

    # post-processing result
    result = (result
              .replace("star", "")
              .replace("five", "5")
              .replace("four", "4")
              .replace("three", "3")
              .replace("two", "2")
              .replace("one", "1")
              .replace("zero", "0")
              .strip()
             )[0]  # get the first digit only

    if result not in ['0', '1', '2', '3', '4', '5']:
        result = ""
    return result

s = """I need a three star hotel with internet. Can you help?"""
get_star(s)
```

Out[12]: '3'

In [13]:

```
# Assign predictions to dfs
train_df["pred_hotel-stars"] = train_df["text_lower"].progress_apply(lambda
dev_df["pred_hotel-stars"] = dev_df["text_lower"].progress_apply(lambda cont
test_df["pred_hotel-stars"] = test_df["text_lower"].progress_apply(lambda cc

print(train_df.columns)
print(dev_df.columns)
print(test_df.columns)
```

```
Index(['text', 'answer', 'intent', 'slots', 'text_lower', 'pred_intent',
       'pred_hotel-stars'],
      dtype='object')
Index(['text', 'answer_raw', 'answer', 'intent', 'slots', 'text_lower',
       'pred_intent', 'pred_hotel-stars'],
      dtype='object')
Index(['text', 'text_lower', 'pred_intent', 'pred_hotel-stars'], dtype='obje
ct')
```

Step 3: Food Type Extractor

for this module, again we will be using RoBERTa question-answering model to extract restaurant food type from context.

In [15]:

```
model_name = "deepset/roberta-base-squad2"

roberta_qa_model = pipeline('question-answering', model=model_name, tokenize

possible_food_types = [
    'afghan', 'afternoon tea', 'turkish', 'mexican', 'swiss',
    'modern european', 'barbeque', 'swedish', 'french', 'kosher',
    'modern global', 'traditional', 'german', 'scandinavian', 'bbq',
    'corsica', 'brazilian', 'eritrean', 'european', 'gastropub', 'steakhouse
    'unusual', 'english', 'australian', 'north indian', 'spanish', 'korean',
    'morrocan', 'international', 'northern european', 'persian', 'vegetarian
    'south indian', 'danish', 'dontcare', 'singaporean', 'catalan', 'welsh',
    'north african', 'modern', 'japanese', 'muslim', 'middle eastern', 'glob
    'panasian', 'christmas', 'lebanese', 'hungarian', 'americas', 'jamaican'
    'british', 'chinese', 'romanian', 'bistro', 'cuban', 'russian', 'cantone
    'thai', 'mediterranean', 'fusion', 'greek', 'polynesian', 'latin america
    'asian oriental', 'australasian', 'sri lankan', 'irish', 'new zealand',
    'belgian', 'venetian', 'creative', 'modern eclectic', 'basque', 'molecul
    'caribbean', 'portuguese', 'scottish', 'tuscan', 'moroccan', 'light bite
    'canapes', 'halal', 'asian', 'indonesian', 'malaysian', 'crossover', 'ir
    'polish', 'the americas', 'italian', 'modern american', 'chines', 'worlc
    'singapore', 'seafood', 'vietnamese'
]
```

```
def get_food_type(text, nlp_model=roberta_qa_model, possible_labels=possible
    """
        extract the food type from the text
    """
    query = {
    'question': 'What is the food origin?',
    'context': text
    }
    res = nlp_model(query)

    result = res["answer"]

    # post-processing result
    result = (result
                 .lower()
                 # .replace("the", "")
                 .replace("food", "")
                 .replace("'s", "")
                 .replace(".", "")
                 .replace(",", "")
                 .replace("-", " ")
                 .replace("restaurant", "")
                 .replace(" or ", " ")
                 .strip()
                )

    # some special case for naming conventions
    if ("modern" in text.lower()) and (not "modern" in result):
        result = "modern " + result
    # if "cuisine" in text.lower():
    #     result = result + " cuisine"
    if result == "sea":
        result = "seafood"
    if (result == "americas") and ("the americas" in text.lower()):
        result = "the americas"
    if "gastropub" in result:
        result = "gastropub"
    if "bistro" in text.lower():
        result = "bistro"

    if result not in possible_labels:
        result = ""

    return result

s = """Hello, I'm looking for a modern european restaurant in the center."""
get_food_type(s)
```

Out[15]:

```
'modern european'
```

In [16]:

```
# Assign predictions to dfs
train_df["pred_restaurant-food"] = train_df["text_lower"].progress_apply(lam
dev_df["pred_restaurant-food"] = dev_df["text_lower"].progress_apply(lambda
test_df["pred_restaurant-food"] = test_df["text_lower"].progress_apply(lambd
print(train_df.columns)
print(dev_df.columns)
print(test_df.columns)
```

```
Index(['text', 'answer', 'intent', 'slots', 'text_lower', 'pred_intent',
       'pred_hotel-stars', 'pred_restaurant-food'],
      dtype='object')
Index(['text', 'answer_raw', 'answer', 'intent', 'slots', 'text_lower',
       'pred_intent', 'pred_hotel-stars', 'pred_restaurant-food'],
      dtype='object')
Index(['text', 'text_lower', 'pred_intent', 'pred_hotel-stars',
       'pred_restaurant-food'],
      dtype='object')
```

Step 4: Restaurant Name and Hotel Name Extractor

In this module, we will use Mixtral 7X8B to extract restaurant name and hotel name from context.

In [17]:

```
# method_1 using local GPU resource
model_path = "/Users/haydenchiu/.cache/lm-studio/models/TheBloke/Mixtral-8x7
```

In [19]:

```
n_gpu_layers = -1
n_batch = 512
n_ctx=512

# Set gpu_layers to the number of layers to offload to GPU. Set to 0 if no G
llm = Llama(
    model_path= model_path + "mixtral-8x7b-instruct-v0.1.Q4_K_M.gguf",  # Do
    n_ctx=n_ctx,  # The max sequence length to use - note that longer sequen
    n_threads=8,            # The number of CPU threads to use, tailor to yo
    n_gpu_layers=n_gpu_layers, # The number of layers to offload to GPU, if
    n_batch=n_batch,
    f16_kv=True,
    chat_format="llama-2",
    verbose=False, #change to True if you want to investigate the logs
)
```

In [26]:

```
def get_name(context, llm=llm, quiet=True):
    """
        get a name of using Mixtral
    """
    tic = time.perf_counter()

    output = llm.create_chat_completion(
    messages = [
        {"role": "system", "content": "You are a helpful assistant that outp
        {"role": "user", "content": str({"context": context,
                                        'question':'what is the name of restaurant o
                                       }) + "Concise answer in json format using inf
        }
        ],
    response_format={
        "type": "json_object",
        },
    )

    result = output['choices'][0]['message']['content']

    try:
        result = eval(result).get("name", "")
    except:
        print("Error parsing result ", result)
        result = ""

    # post processing
    result = (result
                 .lower()
                 # .replace("the", "")
                 .replace("hotel", "")
                 .replace("'s", "")
                 .replace(".", "")
                 # .replace(",", "")
                 # .replace("-", " ")
                 # .replace("restaurant", "")
                 # .replace(" or ", " ")
                 .strip()
             )

    if ("not " in result) or ("no " in result) or ("none " in result):
        # indication name not found
        result = ""

    toc = time.perf_counter()
    if not quiet:
        print(f"extracted gloss in {toc - tic:0.4f} seconds")

    return result
```

In [27]:

```
context = "I am looking for info about a hotel called city centre north b ar
print(get_name(context, quiet=False))

context = "Hi, I am looking for a guesthouse with free parking. Can you help
print(get_name(context, quiet=False))
```

```
extracted gloss in 2.3521 seconds
city centre north b and b
extracted gloss in 19.2120 seconds
```

As we can see, the runtime of running Mixtral 7X8B locally is not ideal. Therefore we will switch to API call approach

In [131…

```
# method_2: using Groq API calls
SLEEP_TIME = 0.1

def get_name_api(row, intent, client=client, quiet=True):
    """
        get name using Mixtral API
    """
    tic = time.perf_counter()

    pred_intent = row["pred_intent"]
    context = row["text_lower"]

    if pred_intent == intent:
        try:
            completion = client.chat.completions.create(
                model="mixtral-8x7b-32768",
                messages=[
                    {
                        "role": "system",
                        "content": "JSON"
                    },
                    {
                        "role": "user",
                        "content": str({"context": context,
                                    'question':f'what is the name of {pred_i
                                   }) + "Concise answer in json format using
                    }
                ],
                temperature=0.0,
                max_tokens=50,
                top_p=1,
                stream=False,
                response_format={"type": "json_object"},
                stop=None,
            )

            time.sleep(SLEEP_TIME)
            result = completion.choices[0].message
        except Exception as error:
            print("Error calling API ", error)
            result = ""

        try:
            result = eval(result.content).get("name", "")
        except:
            print("Error parsing result ", result)
            result = ""

        # post processing
        result = (result
                     .lower()
                     # .replace("the", "")
                     .replace("hotel", "")
                     .replace("'s", "")
                     .replace(".", "")
                     # .replace(",", "")
                     # .replace("-", " ")
                     # .replace("restaurant", "")
                     # .replace(" or ", " ")
                     .strip()
                 )
        if ("not " in result) or ("no " in result) or ("none " in result):
            # indication name not found
            result = ""

        toc = time.perf_counter()
        if not quiet:
            print(f"extracted gloss in {toc - tic:0.4f} seconds")

    else:
        result = ""

    return result


# unit test
intent = "find_hotel"
data = {'text_lower': ["I am looking for info about a hotel called city cent
        'pred_intent': ["find_hotel"]}
row = pd.DataFrame.from_dict(data)
context = "I am looking for info about a hotel called city centre north b ar
print(get_name_api(row.iloc[0], intent, quiet=False))


intent = "find_hotel"
data = {'text_lower': ["Hi, I am looking for a guesthouse with free parking.
        'pred_intent': ["find_hotel"]}
row = pd.DataFrame.from_dict(data)
print(get_name_api(row.iloc[0], intent, quiet=False))

intent = "find_hotel"
data = {'text_lower': ["I am looking for a hotel named alyesbray lodge guest
        'pred_intent': ["find_hotel"]}
row = pd.DataFrame.from_dict(data)
print(get_name_api(row.iloc[0], intent, quiet=False))
```

```
extracted gloss in 0.5364 seconds
city centre north b and b
extracted gloss in 0.3158 seconds

extracted gloss in 0.2878 seconds
alyesbray lodge guest house
```

In [132…

```
# Assign predictions to dfs

# We are not running inference on train set to save time...

# train_df["pred_hotel-name"] = train_df.progress_apply(lambda x: get_name_a
dev_df["pred_hotel-name"] = dev_df.progress_apply(lambda x: get_name_api(x,"
test_df["pred_hotel-name"] = test_df.progress_apply(lambda x: get_name_api(x

# train_df["pred_restaurant-name"] = train_df.progress_apply(lambda x: get_n
dev_df["pred_restaurant-name"] = dev_df.progress_apply(lambda x: get_name_ap
test_df["pred_restaurant-name"] = test_df.progress_apply(lambda x: get_name_

# print(train_df.columns)
```

```
print(dev_df.columns)
print(test_df.columns)
```

```
  0%|          | 0/413 [00:00<?, ?it/s]
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
  0%|          | 0/400 [00:00<?, ?it/s]
```

```
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
```

```
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
  0%|          | 0/413 [00:00<?, ?it/s]
```

```
Error calling API  Error code: 400 - {'error': {'message': "Failed to genera
te JSON. Please adjust your prompt. See 'failed_generation' for more detail
s.", 'type': 'invalid_request_error', 'code': 'json_validate_failed', 'faile
d_generation': '{"name": "cotto"}\n\n(Note: This answer is based on the assu
mption that the name of the restaurant in the given context is cotto. If tha
t is not the case, the answer would be {"name": ""}'}}
Error parsing result
Error calling API  Error code: 400 - {'error': {'message': "Failed to genera
te JSON. Please adjust your prompt. See 'failed_generation' for more detail
s.", 'type': 'invalid_request_error', 'code': 'json_validate_failed', 'faile
d_generation': '{"name": "The Rice Ship" or "The Rice Boat"}'}}
Error parsing result
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
  0%|          | 0/400 [00:00<?, ?it/s]
```

```
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error calling API  Error code: 400 - {'error': {'message': "Failed to genera
te JSON. Please adjust your prompt. See 'failed_generation' for more detail
s.", 'type': 'invalid_request_error', 'code': 'json_validate_failed', 'faile
d_generation': '{"name": "Kohinoor"}\n\nThe context provided contains the na
me of the restaurant, which is "Kohinoor". The name of the restaurant is bei
ng returned in the format {"name": "Kohinoor"}'}}
Error parsing result
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Error parsing result  ChoiceMessage(content='{"name": null}', role='assistan
t', tool_calls=None)
Index(['text', 'answer_raw', 'answer', 'intent', 'slots', 'text_lower',
       'pred_intent', 'pred_hotel-stars', 'pred_restaurant-food',
       'pred_hotel-name', 'pred_restaurant-name', 'pred_slots', 'pred_answer
s',
       'pred_answer_raw'],
      dtype='object')
Index(['text', 'text_lower', 'pred_intent', 'pred_hotel-stars',
       'pred_restaurant-food', 'pred_hotel-name', 'pred_restaurant-name',
       'pred_slots', 'pred_answers', 'pred_answer_raw'],
      dtype='object')
```

In [136…

```
hotel_possible_names = ['lovell lodge', 'hobsons house', 'arbury lodge guest
```

In [137…

```
restaurant_possible_names = ['pipasha restaurant', 'the slug and lettuce', '
```

In [138…

```
def find_closet_match(w, w_list):
    result = ""

    if w.strip() == "":
        return ""

    if ("unknown " in w
        or "no " in w
        or "not " in w
        or "don't " in w
       ):
        return ""


    # pre-process
    w = (w.lower()
         .replace("the", "")
         .replace("hotel", "")
         .replace("restaurant", "")
         .strip()
        )
    print("W =", w)
    for item in w_list:
        if (w in item) or (w == item):
            print("Partial match found", item)
            result = item
            break
    return result

find_closet_match("the lovell lodge hotel", hotel_possible_names + restaurar
```

```
W = lovell lodge
Partial match found lovell lodge
```

Out[138…

```
'lovell lodge'
```

In [139…

```
dev_df["pred_match_hotel-name"] = dev_df["pred_hotel-name"].progress_apply(l
test_df["pred_match_hotel-name"] = test_df["pred_hotel-name"].progress_apply


dev_df["pred_match_restaurant-name"] = dev_df["pred_restaurant-name"].progre
test_df["pred_match_restaurant-name"] = test_df["pred_restaurant-name"].prog

# print(train_df.columns)
print(dev_df.columns)
print(test_df.columns)
```

```
 0%|          | 0/413 [00:00<?, ?it/s]
```

```
W = gonville
Partial match found gonville
W = 0-star guesthouse
W = a and b guest house
Partial match found a and b guest house
W = home from home
Partial match found home from home
W = kirkwood house
Partial match found kirkwood house
W = lovell lodge
Partial match found lovell lodge
W = worth house
Partial match found worth house
W = home from home
Partial match found home from home
W = autumn house
Partial match found autumn house
W = alexander bed and breakfast
Partial match found alexander bed and breakfast
W = express by holiday inn
Partial match found express by holiday inn in cambridge
W = university arms
Partial match found university arms hotel
W = lovell lodge
Partial match found lovell lodge
W = finches bed and breakfast
Partial match found finches bed and breakfast
W = finches bed and breakfast
Partial match found finches bed and breakfast
W = a and b guest house
Partial match found a and b guest house
W = express by holiday inn cambridge
Partial match found express by holiday inn cambridge
W = warkworth house
Partial match found warkworth house
W = hamilton lodge
Partial match found hamilton lodge
W = allenbell
Partial match found allenbell
W = hobsons house
Partial match found hobsons house
W = university arms
Partial match found university arms hotel
W = kirkwood house
Partial match found kirkwood house
W = arbury lodge guesthouse
Partial match found arbury lodge guesthouse
W = arbury lodge guesthouse
Partial match found arbury lodge guesthouse
W = limehouse
Partial match found limehouse
W = kirkwood house
Partial match found kirkwood house
W = acorn guest house
Partial match found acorn guest house
W = el shaddai
```

```
Partial match found el shaddai
W = kirkwood house
Partial match found kirkwood house
W = worth house
Partial match found worth house
W = lovell lodge
Partial match found lovell lodge
W = cityroomz
Partial match found cityroomz
W = huntingdon marriott
Partial match found huntingdon marriott hotel
W = cityroomz
Partial match found cityroomz
W = alpha-milton guest house
Partial match found alpha-milton guest house
W = worth house
Partial match found worth house
W = archway house
Partial match found archway house
W = worth house
Partial match found worth house
W = arbury lodge guesthouse
Partial match found arbury lodge guesthouse
W = avalon
Partial match found avalon
W = autumn house
Partial match found autumn house
W = guest house
Partial match found alpha milton guest house
W = home from home
Partial match found home from home
W = express by holiday inn cambridge
Partial match found express by holiday inn cambridge
W = alpha-milton guest house
Partial match found alpha-milton guest house
W = hamilton lodge
Partial match found hamilton lodge
W = warkworth house
Partial match found warkworth house
W = alpha-milton guest house
Partial match found alpha-milton guest house
W = alpha-milton guest house
Partial match found alpha-milton guest house
W = home from home
Partial match found home from home
W = limehouse
Partial match found limehouse
  0%|          | 0/400 [00:00<?, ?it/s]
```

```
W = carolina bed and breakfast
Partial match found carolina bed and breakfast
W = carolina bed and breakfast
Partial match found carolina bed and breakfast
W = a and b guest house
Partial match found a and b guest house
W = guesthouse
Partial match found arbury lodge guesthouse
W = lensfield
Partial match found lensfield hotel
W = flinches bed and breakfast
W = el shaddai
Partial match found el shaddai
W = university arms
Partial match found university arms hotel
W = aylesbray lodge guest house
Partial match found aylesbray lodge guest house
W = warkworth house
Partial match found warkworth house
W = aylesbray lodge guest house
Partial match found aylesbray lodge guest house
W = a
Partial match found arbury lodge guesthouse
W = hamilton lodge
Partial match found hamilton lodge
W = finches bed and breakfast
Partial match found finches bed and breakfast
W = worth house
Partial match found worth house
W = warkworth house
Partial match found warkworth house
W = huntingdon marriott
Partial match found huntingdon marriott hotel
W = carolina bed and breakfast
Partial match found carolina bed and breakfast
W = warkworth house
Partial match found warkworth house
W = gonville
Partial match found gonville
W = arbury lodge guesthouse
Partial match found arbury lodge guesthouse
W = a and b guest house
Partial match found a and b guest house
W = ashley
Partial match found ashley hotel
W = archway house
Partial match found archway house
W = limehouse
Partial match found limehouse
W = cityroomz
Partial match found cityroomz
W = hobsons house
Partial match found hobsons house
W = archway house
Partial match found archway house
W = allenbell
```

```
Partial match found allenbell
W = aylesbray lodge guest house
Partial match found aylesbray lodge guest house
W = rosa bed and breakfast
W = a and b guest house
Partial match found a and b guest house
W = warkworth house
Partial match found warkworth house
W = alpha-milton guest house
Partial match found alpha-milton guest house
W = expensive  with free parking
W = acorn guest house
Partial match found acorn guest house
W = cambridge belfry
Partial match found cambridge belfry
W = el shaddai
Partial match found el shaddai
W = autumn house
Partial match found autumn house
W = cityroomz
Partial match found cityroomz
W = alpha-milton guest house
Partial match found alpha-milton guest house
W = warkworth house
Partial match found warkworth house
W = alpha-milton guest house
Partial match found alpha-milton guest house
  0%|          | 0/413 [00:00<?, ?it/s]
```

```
W = city stop
Partial match found city stop restaurant
W = riverside brasserie
Partial match found riverside brasserie
W = la raza
Partial match found la raza
W = clowns cafe
Partial match found clowns cafe
W = expensive
W = efes
Partial match found efes restaurant
W = steakhouse
Partial match found river bar steakhouse and grill
W = meghna
Partial match found meghna
W = pizza express fen ditton
Partial match found pizza express fen ditton
W = maharajah tandoori
Partial match found maharajah tandoori restaurant
W = rice boat
Partial match found rice boat
W = golden wok
Partial match found golden wok
W = a cheap
W = tang chinese
Partial match found tang chinese
W = nandos
Partial match found nandos
W = meghna
Partial match found meghna
W = de luca cucina and bar
Partial match found de luca cucina and bar
W = a caribbean  in  center
W = traditional american fine dining
W = expensive  in  center of town
W = lan hong house
Partial match found lan hong house
W = expensive german
W = rajmahal
Partial match found rajmahal
W = pizza hut cherry hinton
W = kymmoy
Partial match found kymmoy
W = sitar tandoori
Partial match found sitar tandoori
W = saint john chop house
W = golden wok
Partial match found golden wok
W = panahar
Partial match found panahar
W = lucky star
Partial match found the lucky star
W = tang chinese
Partial match found tang chinese
W = du vin and bistro
Partial match found hotel du vin and bistro
```

```
W = rice house
Partial match found rice house
W = unknown
W = yippee noodle bar
Partial match found yippee noodle bar
W = michaelhouse cafe
Partial match found michaelhouse cafe
W = pizza hut cherry hinton
W = gandhi
Partial match found the gandhi
W = golden house
Partial match found golden house
W = saint john chop house
W = travellers rest
Partial match found travellers rest
W = shiraz
Partial match found shiraz
W = turkish
W = sitar tandoori
Partial match found sitar tandoori
W = michaelhouse cafe
Partial match found michaelhouse cafe
W = wagamama
Partial match found wagamama
W = royal spice
Partial match found royal spice
W = jinling noodle bar
Partial match found jinling noodle bar
W = cote
Partial match found cote
W = wagamama
Partial match found wagamama
W = nirala
Partial match found nirala
W = clowns cafe
Partial match found clowns cafe
W = cote
Partial match found cote
W = j
Partial match found dojo noodle bar
W = backstreet bistro
Partial match found backstreet bistro
W = rice house
Partial match found rice house
W = copper kettle
Partial match found copper kettle
W = city centre north b&b
W = expensive italian
  0%|          | 0/400 [00:00<?, ?it/s]
```

```
W = golden wok
Partial match found golden wok
W = chiquito
Partial match found chiquito restaurant bar
W = charlie chan
Partial match found charlie chan
W = pizza hut fen ditton
W = meghna
Partial match found meghna
W = bedouin
Partial match found bedouin
W = sala thong
Partial match found sala thong
W = nandos
Partial match found nandos
W = cocum
Partial match found cocum
W = saigon city
Partial match found saigon city
W = cocum
Partial match found cocum
W = slug and lettuce
Partial match found the slug and lettuce
W = upscale
W = one seven
Partial match found restaurant one seven
W = your local s
W = name of   is thanh binh
W = bangkok city
Partial match found bangkok city
W = expensive  in  center
W = city stop
Partial match found city stop restaurant
W = pizza hut
Partial match found pizza hut city centre
W = gastropub
W = prezzo
Partial match found prezzo
W = good luck chinese food takeaway
W = turkish
W = midsummer house
W = mahal of cambridge
Partial match found mahal of cambridge
W = chinese food
W = gardenia
Partial match found the gardenia
W = curry garden
Partial match found curry garden
W = varsity
Partial match found the varsity restaurant
W = meze bar
Partial match found meze bar
W = expensive modern global
W = little seoul
Partial match found little seoul
W = la raza
```

```
Partial match found la raza
W = golden wok
Partial match found golden wok
W = india house
Partial match found india house
W = loch fyne
Partial match found loch fyne
W = la margherita
Partial match found la margherita
W = saigon city
Partial match found saigon city
W = stazione  and coffee bar
W = nandos city centre
Partial match found nandos city centre
W = darrys cookhouse and wine shop
Partial match found darrys cookhouse and wine shop
W = unable to find specific  name in  context
Index(['text', 'answer_raw', 'answer', 'intent', 'slots', 'text_lower',
       'pred_intent', 'pred_hotel-stars', 'pred_restaurant-food',
       'pred_hotel-name', 'pred_restaurant-name', 'pred_slots', 'pred_answer
s',
       'pred_answer_raw', 'pred_match_hotel-name',
       'pred_match_restaurant-name'],
      dtype='object')
Index(['text', 'text_lower', 'pred_intent', 'pred_hotel-stars',
       'pred_restaurant-food', 'pred_hotel-name', 'pred_restaurant-name',
       'pred_slots', 'pred_answers', 'pred_answer_raw',
       'pred_match_hotel-name', 'pred_match_restaurant-name'],
      dtype='object')
```

Step 5: All of the other slots extraction

This module we will use `LaBASE` embedding and logistic regression to classify the rest of the slots.

In [140…

```
slot_names = [
    #'hotel-name',
    #'hotel-stars',
    'hotel-area', 'hotel-internet', 'hotel-pricerange', 'hotel-parking','hot
    #'restaurant-food',
    #'restaurant-name',
    'restaurant-pricerange',
    'restaurant-area'
]

model = SentenceTransformer("sentence-transformers/LaBSE")
embeddings = model.encode(['I want to have some chinese food.', 'I love Japa
embeddings.shape
```

Out[140…

```
(2, 768)
```

In [141…

```
# train LR with training set text LaBSE sentence embedding
x_train = train_df["text_lower"]
x_train = list(train_df["text_lower"])
```

```
x_train_labse = model.encode(x_train)

print(x_train_labse.shape)
```

```
(3760, 768)
```

In [142…

```
master_clf = dict()

for slot in slot_names:
    print("Training for slot =", slot)
    clf = LogisticRegression(random_state=42)
    y_train = train_df["slots"].progress_apply(lambda slots:slots.get(slot,
    # print(y_train.shape)
    print()
    clf.fit(x_train_labse, y_train)
    print(f"Train score = {clf.score(x_train_labse, y_train) * 100:.2f} %")
    master_clf[slot] = clf
    print("------------")

print("Completed")
```

```
Training for slot = hotel-area
  0%|          | 0/3760 [00:00<?, ?it/s]
Train score = 97.90 %
------------
Training for slot = hotel-internet
  0%|          | 0/3760 [00:00<?, ?it/s]
Train score = 98.67 %
------------
Training for slot = hotel-pricerange
  0%|          | 0/3760 [00:00<?, ?it/s]
Train score = 98.22 %
------------
Training for slot = hotel-parking
  0%|          | 0/3760 [00:00<?, ?it/s]
Train score = 98.67 %
------------
Training for slot = hotel-type
  0%|          | 0/3760 [00:00<?, ?it/s]
Train score = 98.96 %
------------
Training for slot = restaurant-pricerange
  0%|          | 0/3760 [00:00<?, ?it/s]
Train score = 97.74 %
------------
Training for slot = restaurant-area
  0%|          | 0/3760 [00:00<?, ?it/s]
Train score = 98.70 %
------------
Completed
```

In [143…

```
# create predicted dictionary for each item
train_df["pred_slots"] = train_df["text"].apply(lambda x: dict())
print(train_df.iloc[0]["pred_slots"])
train_df.head()
```

```
{}
```

Out [143...

	text	answer	intent	slots	text_lower	pred_int
0	Guten Tag, I am staying overnight in Cambridge...	['find_hotel', 'hotel-area=centre', 'hotel-int...	find_hotel	{'hotel-area': 'centre', 'hotel-internet': 'ye...	guten tag, i am staying overnight in cambridge...	find_h
1	Hi there! Can you give me some info on Cityroomz?	['find_hotel', 'hotel-name=cityroomz']	find_hotel	{'hotel-name': 'cityroomz'}	hi there! can you give me some info on cityroomz?	find_h
2	I am looking for a hotel named alyesbray lodge...	['find_hotel', 'hotel-name=alyesbray lodge gue...	find_hotel	{'hotel-name': 'alyesbray lodge guest house'}	i am looking for a hotel named alyesbray lodge...	find_h
3	I am looking for a restaurant. I would like so...	['find_restaurant', 'restaurant-food=chinese',...	find_restaurant	{'restaurant-food': 'chinese', 'restaurant-pri...	i am looking for a restaurant. i would like so...	find_restaur
4	I'm looking for an expensive restaurant in the...	['find_restaurant', 'restaurant-area=centre', ...	find_restaurant	{'restaurant-area': 'centre', 'restaurant-pric...	i'm looking for an expensive restaurant in the...	find_restaur

In [144...

```
# double check train scores
for slot in slot_names:
    y_pred = master_clf[slot].predict(x_train_labse)
    # print("Train score =", master_clf[slot].score(x_train_tfidf, y_train))
    y_train = train_df["slots"].apply(lambda slots:slots.get(slot, ""))
    print("Accuracy score =", accuracy_score(y_train, y_pred) * 100) # shoul

    # go through non-empty result and add to pred_slots
    for i, item in enumerate(y_pred):
        # print(i)
        if item is not None and item != "":
            item_slot = train_df.iloc[i]["pred_slots"]
            # print(item_slot)
            item_slot.update({slot:item})
            train_df.at[i, "pred_slots"] = item_slot
```

```
Accuracy score = 97.89893617021276
Accuracy score = 98.67021276595744
Accuracy score = 98.21808510638299
Accuracy score = 98.67021276595744
Accuracy score = 98.9627659574468
Accuracy score = 97.73936170212765
Accuracy score = 98.6968085106383
```

In [145…

```python
# get accuracy on slots
def get_accuracy(gold_slots, pred_slots, slot_lists=slot_names):
    """
        return accuracy of predicted slots vs gold slots in dictionary form
    """
    correct_count = 0
    for gold_slot, pred_slot in zip(gold_slots, pred_slots):
        gold = {k:v
                for k, v in gold_slot.items()
                if k in slot_names
               }
        # print(gold)
        sys = {k:v
                for k, v in pred_slot.items()
                if k in slot_names
               }
        # print(sys)
        # if gold_slot == pred_slot:
        if gold == sys:
            correct_count += 1
    return correct_count / len(gold_slots)

get_accuracy(train_df["slots"], train_df["pred_slots"])
```

Out[145…

```
0.8970744680851064
```

In [146…

```python
# dev
x_dev = dev_df["text_lower"]
x_dev = list(x_dev)
x_dev_labse = model.encode(x_dev)

print(x_dev_labse.shape)

dev_df["pred_slots"] = dev_df["text"].apply(lambda x: dict())
dev_df.head()
```

```
(413, 768)
```

Out [146…

	text	answer_raw	answer	intent	s
0	I'm looking for a local place to dine in the c...	find_restaurant|restaurant-area=centre|restaur...	['find_restaurant', 'restaurant-area=centre', ...	find_restaurant	{'restaur a 'cer 'restaur fo
1	My husband and I are celebrating our anniversa...	find_hotel	['find_hotel']	find_hotel	
2	I'm looking for an expensive restaurant in the...	find_restaurant|restaurant-area=centre|restaur...	['find_restaurant', 'restaurant-area=centre', ...	find_restaurant	{'restaur a 'cer 'restaur p
3	Are there any accommodations in the east part ...	find_hotel|hotel-area=east|hotel-parking=yes	['find_hotel', 'hotel-area=east', 'hotel-parki...	find_hotel	{'h area': 'e 'h park '
4	I'm looking for a nice place to stay, somewher...	find_hotel|hotel-internet=yes|hotel-pricerange...	['find_hotel', 'hotel-internet=yes', 'hotel-pr...	find_hotel	{'h inter 'yes', 'h pricerar

In [147…

```
# dev
for slot in slot_names:
    # y_dev = dev_df["target"]
    y_dev = dev_df["slots"].apply(lambda slots:slots.get(slot, ""))

    dev_pred = master_clf[slot].predict(x_dev_labse)
    # dev_score = clf.score(x_dev_tfidf, y_dev)
    print("Slot prediction =", slot)
    print(f"Dev accuracy = {accuracy_score(y_dev, dev_pred) * 100:.2f} %")

    # go through non-empty result and add to pred_slots
    for i, item in enumerate(dev_pred):
    # print(i)
        if item is not None and item != "":
            item_slot = dev_df.iloc[i]["pred_slots"]
            # print(item_slot)
            item_slot.update({slot:item})
            dev_df.at[i, "pred_slots"] = item_slot

    # print("Dev accuracy =", dev_score * 100)
```

```
Slot prediction = hotel-area
Dev accuracy = 95.88 %
Slot prediction = hotel-internet
Dev accuracy = 99.52 %
Slot prediction = hotel-pricerange
Dev accuracy = 98.55 %
Slot prediction = hotel-parking
Dev accuracy = 98.06 %
Slot prediction = hotel-type
Dev accuracy = 99.03 %
Slot prediction = restaurant-pricerange
Dev accuracy = 98.31 %
Slot prediction = restaurant-area
Dev accuracy = 98.06 %
```

In [148…

```
# calculate overall accuracy
score = get_accuracy(dev_df["slots"], dev_df["pred_slots"])
print(f"Overall accuracy = {score:.2f} %")
```

```
Overall accuracy = 0.89 %
```

In [149…

```
# test
x_test = test_df["text_lower"]
x_test = list(x_test)
x_test_labse = model.encode(x_test)

print(x_test_labse.shape)

# create predicted dictionary for each item
test_df["pred_slots"] = test_df["text"].apply(lambda x: dict())
test_df.head()
```

```
(400, 768)
```

Out[149…

	text	text_lower	pred_intent	pred_hotel-stars	pred_restaurant-food	pred_hotel-name
0	Hello, I am looking for a restaurant in Cambri...	hello, i am looking for a restaurant in cambri...	find_restaurant			
1	Hi, I'm looking for a hotel to stay in that in...	hi, i'm looking for a hotel to stay in that in...	find_hotel			
2	I am looking for a place to stay in the north ...	i am looking for a place to stay in the north ...	find_hotel	4		
3	I need a place to dine, and I'd like to know w...	i need a place to dine, and i'd like to know w...	find_restaurant		asian oriental	
4	I need a five starts hotel close to a mall and...	i need a five starts hotel close to a mall and...	find_hotel	5		

In [150…

```
for slot in slot_names:

    test_pred = master_clf[slot].predict(x_test_labse)

    # go through non-empty result and add to pred_slots
    for i, item in enumerate(test_pred):
    # print(i)
        if item is not None and item != "":
            item_slot = test_df.iloc[i]["pred_slots"]
            # print(item_slot)
            item_slot.update({slot:item})
            test_df.at[i, "pred_slots"] = item_slot
```

Step 6: Consolidate intent and 11 slots

In [151…

```
dev_df.columns
dev_df.head()
```

Out[151…

	text	answer_raw	answer	intent	s
0	I'm looking for a local place to dine in the c...	find_restaurant|restaurant-area=centre|restaur...	['find_restaurant', 'restaurant-area=centre', ...	find_restaurant	{'restaur a 'cer 'restaur fo
1	My husband and I are celebrating our anniversa...	find_hotel	['find_hotel']	find_hotel	
2	I'm looking for an expensive restaurant in the...	find_restaurant|restaurant-area=centre|restaur...	['find_restaurant', 'restaurant-area=centre', ...	find_restaurant	{'restaur a 'cer 'restaur p
3	Are there any accommodations in the east part ...	find_hotel|hotel-area=east|hotel-parking=yes	['find_hotel', 'hotel-area=east', 'hotel-parki...	find_hotel	{'h area': 'e 'h park '
4	I'm looking for a nice place to stay, somewher...	find_hotel|hotel-internet=yes|hotel-pricerange...	['find_hotel', 'hotel-internet=yes', 'hotel-pr...	find_hotel	{'h inter 'yes', 'h pricerar

In [152…

```
def update_pred_slots(row):
    pred_slots = row["pred_slots"]
    for slot in ['pred_hotel-stars', 'pred_restaurant-food', 'pred_match_hot
        if row[slot] != "":
            pred_slots[slot.split("_")[-1]] = row[slot]
    return pred_slots

dev_df["pred_slots"] = dev_df.progress_apply(update_pred_slots, axis=1)
test_df["pred_slots"] = test_df.progress_apply(update_pred_slots, axis=1)
```

```
 0%|          | 0/413 [00:00<?, ?it/s]
 0%|          | 0/400 [00:00<?, ?it/s]
```

In [155…

```
full_slots = ['hotel-name', 'hotel-stars', 'hotel-area', 'hotel-internet', '
```

In [156…

```
score = get_accuracy(dev_df["slots"], dev_df["pred_slots"], slot_lists=full_
print(f"Overall accuracy = {score:.2f} %")
```

```
Overall accuracy = 0.89 %
```

In [157…

```
# consolidate step 1 - 5
def consolidation(data):
    """


    """
    df = data.copy()

    order = [
        "hotel-area","hotel-internet","hotel-name","hotel-parking","hotel-pr
        "restaurant-area","restaurant-food","restaurant-name","restaurant-pr
    ]

    def reorder_dict(dict, order):
        reordered_dict = {key: dict[key] for key in order if key in dict.key
        return reordered_dict

    df['pred_slots'] = df.progress_apply(lambda x: reorder_dict(x.pred_slots

    def dict_2_list(dict):
        l = []
        for k, v in dict.items():
            l.append(k +"="+v)
        return l

    df['pred_answers'] = df.progress_apply(lambda x: dict_2_list(x.pred_slot

    def to_answer_raw(pred_intent, pred_answers):
        pred_answers.insert(0, pred_intent)
        s = "|".join(pred_answers)
        return s

    df['pred_answer_raw'] = df.progress_apply(lambda x: to_answer_raw(x.pred

    return(df)
```

In [158…

```
dev_df = consolidation(dev_df)

test_df = consolidation(test_df)
```

```
 0%|          | 0/413 [00:00<?, ?it/s]
 0%|          | 0/413 [00:00<?, ?it/s]
 0%|          | 0/413 [00:00<?, ?it/s]
 0%|          | 0/400 [00:00<?, ?it/s]
 0%|          | 0/400 [00:00<?, ?it/s]
 0%|          | 0/400 [00:00<?, ?it/s]
```

In [159…

```
dev_df
```

Out[159…

	text	answer_raw	answer	intent	
0	I'm looking for a local place to dine in the c...	find_restaurant|restaurant-area=centre|restaur...	['find_restaurant', 'restaurant-area=centre', ...	find_restaurant	{'rest ' 'rest
1	My husband and I are celebrating our anniversa...	find_hotel	['find_hotel']	find_hotel	
2	I'm looking for an expensive restaurant in the...	find_restaurant|restaurant-area=centre|restaur...	['find_restaurant', 'restaurant-area=centre', ...	find_restaurant	{'rest ' 'rest
3	Are there any accommodations in the east part ...	find_hotel|hotel-area=east|hotel-parking=yes	['find_hotel', 'hotel-area=east', 'hotel-parki...	find_hotel	area' p
4	I'm looking for a nice place to stay, somewher...	find_hotel|hotel-internet=yes|hotel-pricerange...	['find_hotel', 'hotel-internet=yes', 'hotel-pr...	find_hotel	in 'yes', price
...	...	...	...	...	
408	I'm looking for info about 4-star accommodatio...	find_hotel|hotel-internet=yes|hotel-stars=4	['find_hotel', 'hotel-internet=yes', 'hotel-st...	find_hotel	in 'yes', sta
409	I'm looking for a place to eat that is cheap a...	find_restaurant|restaurant-area=centre|restaur...	['find_restaurant', 'restaurant-area=centre', ...	find_restaurant	{'rest ' 'rest
410	Hi, I'm looking for an expensive restaurant in...	find_restaurant|restaurant-area=north|restaura...	['find_restaurant', 'restaurant-area=north', '...	find_restaurant	{'rest 'rest
411	Can you help me find a restaurant? I want some...	find_restaurant|restaurant-pricerange=expensive	['find_restaurant', 'restaurant-pricerange=exp...	find_restaurant	{'rest price 'expe
412	I'm going to Cambridge and interested in tryin...	find_restaurant|restaurant-food=traditional	['find_restaurant', 'restaurant-food=tradition...	find_restaurant	{'rest 'trad

413 rows × 16 columns

In [ ]:

```
```

In [160…

```
# dev evaluation
print(f'slots accuracy: {get_accuracy(dev_df["slots"], dev_df["pred_slots"],
print(f'intent accuracy: {accuracy_score(dev_df["intent"], dev_df["pred_inte
print(f'answer accuracy: {accuracy_score(dev_df["answer_raw"], dev_df["pred_
```

```
slots accuracy: 0.8910411622276029
intent accuracy: 0.9927360774818402
answer accuracy: 0.837772397094431
```

In [161…

```
# inspect the errors
dev_df[dev_df["slots"]!=dev_df["pred_slots"]]
```

Out [161…

	text	answer_raw	answer	intent	
5	I'm looking for a 4 star hotel in the south.	find_hotel|hotel-area=south|hotel-stars=4	['find_hotel', 'hotel-area=south', 'hotel-star...	find_hotel	{'hotel- 's 'hotel-s
13	I am looking to get some information on gonvil...	find_hotel|hotel-name=gonville hotel	['find_hotel', 'hotel-name=gonville hotel']	find_hotel	{'l n 'gc h
14	Could you tell me where Cotto is located?	find_restaurant|restaurant-name=cotto	['find_restaurant', 'restaurant-name=cotto']	find_restaurant	{'restau n 'c
18	Yes, hello. I need a place to crash so I'm thi...	find_hotel|hotel-stars=0|hotel-type=guesthouse	['find_hotel', 'hotel-stars=0', 'hotel-type=gu...	find_hotel	{'hotel-s '0', 'l 'guestho
27	I need to find a barbeque restaurant in the ce...	find_restaurant|restaurant-food=barbeque	['find_restaurant', 'restaurant-food=barbeque']	find_restaurant	{'restau 'barbe
...	...	...	...	...	
393	I want to go to a french food restaurant in th...	find_restaurant|restaurant-area=north|restaura...	['find_restaurant', 'restaurant-area=north', '...	find_restaurant	{'restau area': 'r 'restau fc
394	A friend recommended the City Centre North B&B...	find_hotel|hotel-name=city centre north b and b	['find_hotel', 'hotel-name=city centre north b...	find_hotel	{'l name' centre b a
398	I'm looking for a gueshouse that includes free...	find_hotel|hotel-parking=yes|hotel-type=guesth...	['find_hotel', 'hotel-parking=yes', 'hotel-typ...	find_hotel	{'l par 'yes', 'l 'gue
401	I need an expensive place to stay that include...	find_hotel|hotel-internet=yes|hotel-pricerange...	['find_hotel', 'hotel-internet=yes', 'hotel-pr...	find_hotel	{'l inte 'yes', 'l pricera
410	Hi, I'm looking for an expensive restaurant in...	find_restaurant|restaurant-area=north|restaura...	['find_restaurant', 'restaurant-area=north', '...	find_restaurant	{'restau area': 'r 'restau p

66 rows × 16 columns

Step 7: Output

In [162…

```
dev_df.to_csv("./data/dev_3nd_model.csv")
dev_df[["text","pred_answer_raw"]].to_csv('./data/dev_3nd_model_pred.txt', s
```

In [163…

```
test_df.to_csv("./data/test_3nd_model.csv")
test_df[["text","pred_answer_raw"]].to_csv('./data/test_3nd_model_pred.txt',
```

In [164…

```
kaggle_df = test_df.copy()
kaggle_df = test_df.reset_index()

kaggle_df = kaggle_df.rename(columns={"index":"ID", "pred_answer_raw":"Expec

kaggle_df[["ID", "Expected"]].to_csv('./data/WOZ_test_3nd_model_ans.csv', ir
```

In [ ]:

```

```