Final AI Project Submission

***Name** – CHANDRIKKA PATHAK*

***Batch** - ARTIFICIAL INTELLIGENCE*

***Project name** - DETERMINING DIFFERENT ENTITY IN FOOD DELIVERY DATA*

***Certificate Code**- TCRIL01R07*

***Group**- OWN*

```
# downloading spacy language model
!pip install spacy==2.3.1
```

```
# importing libraries
import en_core_web_sm
import pandas as pd
import re
import random
import spacy
from spacy.util import minibatch, compounding
import warnings
import matplotlib.pyplot as plt
```

```
# Generating Food Data
```

```
# USDA's Branded Food's dataset:
#https://fdc.nal.usda.gov/fdc-
datasets/FoodData_Central_foundation_food_csv_2022-04-28.zip
```

```
# Preparing the food data
```

```
# read in the food csv file
food_df = pd.read_csv("/content/drive/MyDrive/INTERNSHIPS/TCR Internship
(AI)/Final Project/food.csv")

# print row and column information
food_df.head()
```

```
# print the size
food_df["description"].size
```

```
# disqualify foods with special characters, lowercase and extract results from
"description" column
foods = food_df[food_df["description"].str.contains("[^a-zA-Z ]") ==
False]["description"].apply(lambda food: food.lower())

# filter out foods with more than 3 words, drop any duplicates
foods = foods[foods.str.split().apply(len) <= 3].drop_duplicates()

# print the remaining size
foods.size
```

```
# find one-worded, two-worded and three-worded foods
one_worded_foods = foods[foods.str.split().apply(len) == 1]
two_worded_foods = foods[foods.str.split().apply(len) == 2]
three_worded_foods = foods[foods.str.split().apply(len) == 3]

# create a bar plot
fig, ax = plt.subplots(figsize=(10, 6))
ax.bar([1, 2, 3], [one_worded_foods.size, two_worded_foods.size,
three_worded_foods.size])

# label the x-axis instances
ax.set_xticks([1, 2, 3])
ax.set_xticklabels(["one", "two", "three"])

# set the title and the xy-axis labels
plt.title("Number of Words in Food Entities")
plt.xlabel("Number of Words")
plt.ylabel("Food Entities")

# display the plot
plt.show()
```

```
# total number of foods
total_num_foods = round(one_worded_foods.size / 45 * 100)

# shuffle the 2-worded and 3-worded foods since we'll be slicing them
two_worded_foods = two_worded_foods.sample(frac=1)
three_worded_foods = three_worded_foods.sample(frac=1)

# append the foods together
foods = one_worded_foods.append(two_worded_foods[:round(total_num_foods *
0.30)]).append(three_worded_foods[:round(total_num_foods * 0.25)])
```

```python
# print the resulting sizes
for i in range(3):
    print(f"{i+1}-worded food entities:", foods[foods.str.split().apply(len)
== i + 1].size)
```

```python
food_templates = [
    "I ate my {}",
    "I'm eating a {}",
    "I just ate a {}",
    "I only ate the {}",
    "I'm done eating a {}",
    "I've already eaten a {}",
    "I just finished my {}",
    "When I was having lunch I ate a {}",
    "I had a {} and a {} today",
    "I ate a {} and a {} for lunch",
    "I made a {} and {} for lunch",
    "I ate {} and {}",
    "today I ate a {} and a {} for lunch",
    "I had {} with my husband last night",
    "I brought you some {} on my birthday",
    "I made {} for yesterday's dinner",
    "last night, a {} was sent to me with {}",
    "I had {} yesterday and I'd like to eat it anyway",
    "I ate a couple of {} last night",
    "I had some {} at dinner last night",
    "Last night, I ordered some {}",
    "I made a {} last night",
    "I had a bowl of {} with {} and I wanted to go to the mall today",
    "I brought a basket of {} for breakfast this morning",
    "I had a bowl of {}",
    "I ate a {} with {} in the morning",
    "I made a bowl of {} for my breakfast",
    "There's {} for breakfast in the bowl this morning",
    "This morning, I made a bowl of {}",
    "I decided to have some {} as a little bonus",
    "I decided to enjoy some {}",
    "I've decided to have some {} for dessert",
    "I had a {}, a {} and {} at home",
    "I took a {}, {} and {} on the weekend",
    "I ate a {} with {} and {} just now",
    "Last night, I ate an {} with {} and {}",
    "I tasted some {}, {} and {} at the office",
    "There's a basket of {}, {} and {} that I consumed",
    "I devoured a {}, {} and {}",
    "I've already had a bag of {}, {} and {} from the fridge"
]
```

```
data = [
    ("I love chicken", [(8, 13, "FOOD")]),
    ...
]
```

```
# create dictionaries to store the generated food combinations. Do note that
one_food != one_worded_food. one_food == "barbecue sauce", one_worded_food ==
"sauce"
TRAIN_FOOD_DATA = {
    "one_food": [],
    "two_foods": [],
    "three_foods": []
}

TEST_FOOD_DATA = {
    "one_food": [],
    "two_foods": [],
    "three_foods": []
}

# one_food, two_food, and three_food combinations will be limited to 167
sentences
FOOD_SENTENCE_LIMIT = 167

# helper function for deciding what dictionary and subsequent array to append
the food sentence on to
def get_food_data(count):
    return {
        1: TRAIN_FOOD_DATA["one_food"] if len(TRAIN_FOOD_DATA["one_food"]) <
FOOD_SENTENCE_LIMIT else TEST_FOOD_DATA["one_food"],
        2: TRAIN_FOOD_DATA["two_foods"] if len(TRAIN_FOOD_DATA["two_foods"]) <
FOOD_SENTENCE_LIMIT else TEST_FOOD_DATA["two_foods"],
        3: TRAIN_FOOD_DATA["three_foods"] if
len(TRAIN_FOOD_DATA["three_foods"]) < FOOD_SENTENCE_LIMIT else
TEST_FOOD_DATA["three_foods"],
    }[count]

# the pattern to replace from the template sentences
pattern_to_replace = "{}"

# shuffle the data before starting
foods = foods.sample(frac=1)

# the count that helps us decide when to break from the for loop
food_entity_count = foods.size - 1
```

```
# start the while loop, ensure we don't get an index out of bounds error
while food_entity_count >= 2:
    entities = []

    # pick a random food template
    sentence = food_templates[random.randint(0, len(food_templates) - 1)]

    # find out how many braces "{}" need to be replaced in the template
    matches = re.findall(pattern_to_replace, sentence)

    # for each brace, replace with a food entity from the shuffled food data
    for match in matches:
        food = foods.iloc[food_entity_count]
        food_entity_count -= 1

        # replace the pattern, but then find the match of the food entity we
just inserted
        sentence = sentence.replace(match, food, 1)
        match_span = re.search(food, sentence).span()

        # use that match to find the index positions of the food entity in the
sentence, append
        entities.append((match_span[0], match_span[1], "FOOD"))

    # append the sentence and the position of the entities to the correct
dictionary and array
    get_food_data(len(matches)).append((sentence, {"entities": entities}))
```

```
# print the number of food sentences, as well as an example sentence
for key in TRAIN_FOOD_DATA:
    print("{} {} sentences: {}".format(len(TRAIN_FOOD_DATA[key]), key,
TRAIN_FOOD_DATA[key][0]))
```

```
for key in TEST_FOOD_DATA:
    print("{} {} items: {}".format(len(TEST_FOOD_DATA[key]), key,
TEST_FOOD_DATA[key][0]))
```

```
#  Generating Revision Data
```

```
# Preparing the revision data
```

```
npr_df = pd.read_csv("/content/drive/MyDrive/INTERNSHIPS/TCR Internship
(AI)/Final Project/npr.csv")
```

```python
# print row and column information
npr_df.head()
```

```python
# create an nlp object as we'll use this to seperate the sentences and
identify existing entities
import spacy.cli
spacy.cli.download("en_core_web_lg")
nlp = spacy.load("en_core_web_lg")
```

```python
revision_texts = []

# convert the articles to spacy objects to better identify the sentences.
Disabled unneeded components. # takes ~ 4 minutes
for doc in nlp.pipe(npr_df["Article"][:6000], batch_size=30,
disable=["tagger", "ner"]):
    for sentence in doc.sents:
        if  40 < len(sentence.text) < 80:
            # some of the sentences had excessive whitespace in between words,
so we're trimming that
            revision_texts.append(" ".join(re.split("\s+", sentence.text,
flags=re.UNICODE)))
```

```python
revisions = []

# Use the existing spaCy model to predict the entities, then append them to
revision
for doc in nlp.pipe(revision_texts, batch_size=50, disable=["tagger",
"parser"]):

    # don't append sentences that have no entities
    if len(doc.ents) > 0:
        revisions.append((doc.text, {"entities": [(e.start_char, e.end_char,
e.label_) for e in doc.ents]}))
```

```python
# Split train and test revision data
```

```python
# print an example of the revision sentence
print(revisions[0][0])

# print an example of the revision data
print(revisions[0][1])
```

```python
# create arrays to store the revision data
```

```python
TRAIN_REVISION_DATA = []
TEST_REVISION_DATA = []

# create dictionaries to keep count of the different entities
TRAIN_ENTITY_COUNTER = {}
TEST_ENTITY_COUNTER = {}

# This will help distribute the entities (i.e. we don't want 1000 PERSON
entities, but only 80 ORG entities)
REVISION_SENTENCE_SOFT_LIMIT = 100

# helper function for incrementing the revision counters
def increment_revision_counters(entity_counter, entities):
    for entity in entities:
        label = entity[2]
        if label in entity_counter:
            entity_counter[label] += 1
        else:
            entity_counter[label] = 1

random.shuffle(revisions)
for revision in revisions:
    # get the entities from the revision sentence
    entities = revision[1]["entities"]

    # simple hack to make sure spaCy entities don't get too one-sided
    should_append_to_train_counter = 0
    for _, _, label in entities:
        if label in TRAIN_ENTITY_COUNTER and TRAIN_ENTITY_COUNTER[label] >
REVISION_SENTENCE_SOFT_LIMIT:
            should_append_to_train_counter -= 1
        else:
            should_append_to_train_counter += 1

    # simple switch for deciding whether to append to train data or test data
    if should_append_to_train_counter >= 0:
        TRAIN_REVISION_DATA.append(revision)
        increment_revision_counters(TRAIN_ENTITY_COUNTER, entities)
    else:
        TEST_REVISION_DATA.append(revision)
        increment_revision_counters(TEST_ENTITY_COUNTER, entities)
```

```
{'DATE': 212,
 'GPE': 164,
 'CARDINAL': 195,
 'PERSON': 254,
 'LANGUAGE': 85,
 'ORG': 192,
```

```
 'WORK_OF_ART': 103,
 'TIME': 108,
 'ORDINAL': 110,
 'PERCENT': 101,
 'NORP': 115,
 'LOC': 106,
 'MONEY': 102,
 'QUANTITY': 101,
 'EVENT': 101,
 'PRODUCT': 101,
 'LAW': 95,
 'FAC': 101}
```

```
{'PERSON': 14027,
 'ORG': 10360,
 'DATE': 7153,
 'GPE': 5661,
 'NORP': 2739,
 'CARDINAL': 5397,
 'QUANTITY': 171,
 'PERCENT': 441,
 'TIME': 794,
 'FAC': 152,
 'LOC': 559,
 'ORDINAL': 1151,
 'MONEY': 560,
 'WORK_OF_ART': 592,
 'PRODUCT': 119,
 'EVENT': 104,
 'LANGUAGE': 24,
 'LAW': 12}
```

```
# Training the NER Model
```

```
# combine the food training data
TRAIN_FOOD_DATA_COMBINED = TRAIN_FOOD_DATA["one_food"] +
TRAIN_FOOD_DATA["two_foods"] + TRAIN_FOOD_DATA["three_foods"]

# print the length of the food training data
print("FOOD", len(TRAIN_FOOD_DATA_COMBINED))

# print the length of the revision training data
print("REVISION", len(TRAIN_REVISION_DATA))

# join and print the combined length
TRAIN_DATA = TRAIN_REVISION_DATA + TRAIN_FOOD_DATA_COMBINED
```

```
print("COMBINED", len(TRAIN_DATA))
```

```
# add NER to the pipeline and the new label
ner = nlp.get_pipe("ner")
ner.add_label("FOOD")

# get the names of the components we want to disable during training
pipe_exceptions = ["ner", "trf_wordpiecer", "trf_tok2vec"]
other_pipes = [pipe for pipe in nlp.pipe_names if pipe not in pipe_exceptions]

# start the training loop, only training NER
epochs = 30
optimizer = nlp.resume_training()
with nlp.disable_pipes(*other_pipes), warnings.catch_warnings():
    warnings.filterwarnings("once", category=UserWarning, module='spacy')
    sizes = compounding(1.0, 4.0, 1.001)

    # batch up the examples using spaCy's minibatc
    for epoch in range(epochs):
        examples = TRAIN_DATA
        random.shuffle(examples)
        batches = minibatch(examples, size=sizes)
        losses = {}

        for batch in batches:
            texts, annotations = zip(*batch)
            nlp.update(texts, annotations, sgd=optimizer, drop=0.35,
losses=losses)

        print("Losses ({}/{})".format(epoch + 1, epochs), losses)
```

```
# Evaluating the Model
```

```
# display sentence involving original entities
spacy.displacy.render(nlp("Apple is looking at buying U.K. startup for $1
billion"), style="ent")
```

```
# display sentences involving target entity
spacy.displacy.render(nlp("I had a hamburger and chips for lunch today."),
style="ent")
spacy.displacy.render(nlp("I decided to have chocolate ice cream as a little
treat for myself."), style="ent")
spacy.displacy.render(nlp("I ordered basmati rice, leaf spinach and cheese
from Tesco yesterday"), style="ent")
```

```
#Evaluating the Model
```

```
# display sentence involving original entities
spacy.displacy.render(nlp("Apple is looking at buying U.K. startup for $1 billion"), style="ent")
```

```
# display sentences involving target entity
spacy.displacy.render(nlp("I had a hamburger and chips for lunch today."), style="ent")
spacy.displacy.render(nlp("I decided to have chocolate ice cream as a little treat for myself."), style="ent")
spacy.displacy.render(nlp("I ordered basmati rice, leaf spinach and cheese from Tesco yesterday"), style="ent")
```

```
#Evaluating Food Entities
```

```
# dictionary to hold our evaluation data
food_evaluation = {
    "one_food": {
        "correct": 0,
        "total": 0,
    },
    "two_foods": {
        "correct": 0,
        "total": 0
    },
    "three_foods": {
        "correct": 0,
        "total": 0
    }
}

word_evaluation = {
    "1_worded_foods": {
        "correct": 0,
        "total": 0
    },
    "2_worded_foods": {
        "correct": 0,
        "total": 0
    },
    "3_worded_foods": {
        "correct": 0,
        "total": 0
    }
```

```
}

# loop over data from our test food set (3 keys in total)
for key in TEST_FOOD_DATA:
    foods = TEST_FOOD_DATA[key]

    for food in foods:
        # extract the sentence and correct food entities according to our test data
        sentence = food[0]
        entities = food[1]["entities"]

        # for each entity, use our updated model to make a prediction on the sentence
        for entity in entities:
            doc = nlp(sentence)
            correct_text = sentence[entity[0]:entity[1]]
            n_worded_food =  len(correct_text.split())

            # if we find that there's a match for predicted entity and predicted text, increment correct counters
            for ent in doc.ents:
                if ent.label_ == entity[2] and ent.text == correct_text:
                    food_evaluation[key]["correct"] += 1
                    if n_worded_food > 0:
                        word_evaluation[f"{n_worded_food}_worded_foods"]["correct"] += 1

                    # this break is important, ensures that we're not double counting on a correct match
                    break

            #  increment total counters after each entity loop
            food_evaluation[key]["total"] += 1
            if n_worded_food > 0:
                word_evaluation[f"{n_worded_food}_worded_foods"]["total"] += 1
```

```
for key in word_evaluation:
    correct = word_evaluation[key]["correct"]
    total = word_evaluation[key]["total"]

    print(f"{key}: {correct / total * 100:.2f}%")

food_total_sum = 0
food_correct_sum = 0

print("---")
for key in food_evaluation:
```

```
    correct = food_evaluation[key]["correct"]
    total = food_evaluation[key]["total"]

    food_total_sum += total
    food_correct_sum += correct

    print(f"{key}: {correct / total * 100:.2f}%")

print(f"\nTotal: {food_correct_sum/food_total_sum * 100:.2f}%")
```

```
#Evaluating Existing Entities
```

```
# dictionary which will be populated with the entities and result information
entity_evaluation = {}

# helper function to udpate the entity_evaluation dictionary
def update_results(entity, metric):
    if entity not in entity_evaluation:
        entity_evaluation[entity] = {"correct": 0, "total": 0}

    entity_evaluation[entity][metric] += 1

# same as before, see if entities from test set match what spaCy currently
predicts
for data in TEST_REVISION_DATA:
    sentence = data[0]
    entities = data[1]["entities"]

    for entity in entities:
        doc = nlp(sentence)
        correct_text = sentence[entity[0]:entity[1]]

        for ent in doc.ents:
            if ent.label_ == entity[2] and ent.text == correct_text:
                update_results(ent.label_, "correct")
                break

        update_results(entity[2], "total")
```

```
sum_total = 0
sum_correct = 0

for entity in entity_evaluation:
    total = entity_evaluation[entity]["total"]
    correct = entity_evaluation[entity]["correct"]
```

```python
    sum_total += total
    sum_correct += correct

    print("{} | {:.2f}%".format(entity, correct / total * 100))

print()
print("Overall accuracy: {:.2f}%".format(sum_correct / sum_total * 100))
```

```python
# Saving the model
```

```python
nlp.meta["name"] = "food_entity_extractor_v2"
nlp.to_disk("v2")
```

```python
# The results we arrived at is the following for our FOOD entities:
"""
1_worded_foods: 90.91%
2_worded_foods: 95.33%
3_worded_foods: 92.66%
---
one_food: 90.50%
two_foods: 94.69%
three_foods: 94.74%

Total: 92.66%
"""
```

```python
# The results for our existing entities:
"""
PERSON | 79.39%
ORG | 59.51%
DATE | 64.28%
GPE | 82.12%
ORDINAL | 97.22%
TIME | 62.69%
CARDINAL | 82.22%
MONEY | 85.36%
NORP | 83.74%
LOC | 66.07%
WORK_OF_ART | 61.25%
EVENT | 56.73%
PERCENT | 90.23%
FAC | 66.89%
QUANTITY | 70.18%
LANGUAGE | 92.31%
PRODUCT | 49.58%
```

```
LAW | 75.00%

Overall accuracy: 73.72%

"""
```

```
# chandrikapathak3001@gmail.com
```