SpaCy and Named Entity Recognition

Group project report submitted in partial fulfillment for the degree of

MSc in Data Analytics

Authors:
Bharath Shakthivel - 20057027
Harshitha Ramesh - 20061096
Serene Itani - 20029895
Tiangen Liu - 20058052

March 7th 2025

Table of Contents

1. Introduction

This report details all processes taken by the group to create our model with SpaCy using Named Entity Recognition (NER). The group consists of 4 individuals: Bharath Shakthivel, Harshitha Ramesh, Serene Itani, and Tiangen Liu.

1.1 SpaCy and its' features

SpaCy is a library that specializes in Natural Language Processing (NLP) which is the ideal library to use, since NER falls into the umbrella of NLP. Our aim was to create a model that can efficiently and consistently evaluate entities from the chosen dataset. We will achieve this by providing our model with continuous manual annotations through iterations, specifically looking at confidence based random sampling.

1.2 Active Learning

Active learning is a technique designed to enhance the annotation process by identifying the most informative samples from a dataset for human labeling. This method is especially beneficial when working with large datasets or when the annotation costs are significant. In the realm of Named Entity Recognition (NER), active learning can boost model performance while minimizing the requirement for extensive labeled data.

Active learning improves efficiency by concentrating on samples that provide the greatest benefit to model enhancement, thus lowering annotation expenses and increasing model accuracy.

1.3 Dataset Selection

The WNUT-16 dataset has been selected for this project because of its challenging characteristics. It comprises social media text, which is often noisy and informal, and includes a variety of entity types. This dataset demands robust models that can effectively manage diverse and variable entity mentions. It also contains 10 different types of entities.

1.4 Libraries, packages, and SpaCy pipelines

To utilize NER with spaCy, we begin by establishing a basic NER pipeline. This process includes loading a pre-trained model or creating a new one if necessary. The `Trainable Pipe` class in spaCy allows for the customization of trainable components. Our group used many different

libraries and packages to achieve our goal which will be covered more in-depth in the "Methodologies" portion of the report. The key features we used throughout this model, for pre-processing our train datasets were; Tokenization and Rule-based matching. Tokenization is a feature which formats each word into its own entity and location - this will help later on with specifying locations for manual annotations. Rule-based matching takes each token and applies rules onto them based on our given parameters.

1.5 Evaluation Metrics

Using precision to find the ratio of correctly identified entities to all identified entities. Moreover, recall is looked at as it is the ratio of correctly identified entities to all actual entities. Finally F1-score tells us the harmonic mean of precision and recall.

1.6 Model Deployment

After achieving the desired performance, deploy the final NER model for real-world applications. Make sure the model is robust and capable of handling unseen data effectively.

Some keys our group kept in mind are; Model generalizability - to confirm that the model performs well on unseen data as well as, scalability to consider how the model will manage large volumes of data. These steps helped us deploy a robot and scalable NER model that can effectively identify entities in real-world scenarios.

2. Literature Review

2.1 Real-World Example (Twitter NER)

The Maxar Blog post titled "Named Entity Recognition for Twitter" explores the unique challenges and strategies involved in applying Named Entity Recognition (NER) to Twitter data.

<u>Key Challenges</u>

- Informal Language: Twitter is known for its informal language, which includes misspellings, abbreviations, hashtags, mentions, and inconsistent capitalization and punctuation. These elements can hinder the effectiveness of standard NER systems.

- Data Sparsity and Variability: Tweets are typically brief and provide limited context, leading to data sparsity. Moreover, the fast-paced nature of Twitter means that new entities and topics frequently emerge, making it challenging to keep training data current.

- Entity Types: The range of entities found in Twitter data is broad, encompassing people, places, organizations, and events. However, the casual style of tweets often blurs the distinctions between these categories, complicating the process of entity recognition.

Tools and Models

- TwitterNER: This tool is recognized for its high precision and F1-score in identifying entities on Twitter. It is particularly recommended for applications that require high precision, although its recall may not be as strong as some other models.

- Stanford CoreNLP: Although not specifically tailored for Twitter, CoreNLP is known for its excellent recall performance. It may be the preferred choice when recall is more important than precision.

Summary

The Maxar Blog post offers valuable insights into the challenges of applying NER to Twitter data and underscores the necessity for custom training and specialized tools to achieve the best results. It stresses the importance of adapting NLP models to the unique features of social media data to enhance entity recognition accuracy.

2.2 Matthew Honnibal

There is a plethora of literature pertaining to SpaCy for NER, taking into consideration ease of understanding and reliability a lot of our information was taken from SpaCy itself, namely from Matthew Honnibal as he is the author of SpaCys' natural language processing library.

His work on "SpaCy 101: Everything You Need to Know" was an especially useful resource as it outlines the basics of this library, which as we all know, is the best place to start and provide foundational knowledge for creating our model. In this article, Hannibal starts by explaining what SpaCy is and the many features and capabilities it poses to help us reach our goal through both linguistic concepts and machine learning processes.

2.3 Pijush Pathak, Linson Thomas Verghese, and Dr. G. Divya

The research paper titled "Sentiment Analysis Text Extraction from Tweets with Spacy NER," authored by Pijush Pathak, Linson Thomas Verghese, and Dr. G. Divya, delves into how spaCy's Named Entity Recognition (NER) can be applied for sentiment analysis on Twitter data. The

main objective is to create algorithms that can pinpoint and extract segments of tweets that express sentiment, thus offering insights into public opinions and emotional subtleties in online conversations.

Key Aspects

- The researchers employ spaCy, a robust NLP library, to develop two distinct NER models: one focused on positive sentiment and the other on negative sentiment.

- They compile a tagged dataset of tweets with sentiment labels to train these models, allowing them to identify patterns associated with both positive and negative sentiment expressions
- The study aims to assess how well spaCy NER models can detect and extract text excerpts rich in sentiment from Twitter data.

- It aspires to uncover the intricate emotional nuances found in public discussions by analyzing the sentiment snippets that are extracted.

- This research enhances our understanding of public sentiment on Twitter, which is vital for social media monitoring, analyzing customer feedback, and managing brand reputation.

- It underscores the potential of NLP and machine learning in deriving valuable insights from social media data.

Summary

This research highlights how effective spaCy's NER capabilities can be for analyzing sentiment in Twitter data. It introduces a unique method by training distinct models for both positive and negative sentiments, which enables a more detailed extraction of text fragments that convey sentiment. The results are significant for stakeholders who want to grasp public opinion and the dynamics of sentiment on social media platforms.

3. Methodology

From the knowledge we have gained through the mentioned literature review we took the following approach in creating our model. Using features to allow our program to break down the dataset and annotate it while still storing all the original information. Tokenization which segments the text into words, punctuation, etc... Moreover, an important and main aspect of our model is named entities, which - with the help of the features previously mentioned - allows us to

train our model on what categories “real-world entities” fall into. This, like many machine learning programs, is not perfect as it relies on the developer to efficiently train the model using manual annotations to cover all possible scenarios. This was done through approximately 14 iterations of low-confidence based random sampling. The following figures will show all libraries and packages used in our code and their purpose.

3.1 Dataset refresh

The WNUT-16 dataset serves as the foundation for this task, which consists of; Train and Development Sets - These sets are employed for active learning, allowing the model to be trained and updated iteratively with fresh annotations. As well as a test set which is designated for the final evaluation of the trained model's performance.

3.2 Data Preprocessing

Tokenization was a key factor of preprocessing as it broke the text into individual tokens. This allowed us to add manual annotation to train our data.

Normalization and cleaning also played a role in eliminating unnecessary characters like punctuation and special symbols to standardize the text. This improves the model's focus on significant content.

Preprocessing is crucial to ensure that the data is formatted appropriately for model training, minimizing noise and enhancing model efficiency.

3.3 Active Learning Strategy

Using rule-based matching, SpaCy’s entity ruler initial annotation was created. This involved establishing rules to identify specific patterns in the text, aiding in entity recognition.

Uncertainty sampling chose predictions with high uncertainty for human annotation. This method ensures that the most unclear cases are manually examined and corrected, boosting model accuracy. This was done over 14 iterations.

Incremental model training continuously updated the model with newly annotated data. This strategy enables the model to learn from its errors and adjust to new information, improving its performance over time.

The active learning strategy merges automated and human annotation processes to effectively enhance model performance by concentrating on uncertain predictions.

3.4 Model Training

Creating a custom Named Entity Recognition (NER) model with SpaCy involves several important steps, as follows:

1. Data Preparation: This is the initial phase, where the dataset is annotated by labeling named entities according to their categories, such as person, organization, or location. The text data also needs to be cleaned and preprocessed by removing any unnecessary characters and normalizing it.

2. Model Creation: This step starts with a blank SpaCy model, meaning no pre-trained weights are used. This method demands more data and computational power but offers complete customization. The model is then trained using the annotated dataset, allowing it to learn patterns and relationships between words and their corresponding entity labels.

3. Model Training and Iteration: This phase consists of training the model in cycles, tweaking parameters and hyperparameters as necessary to enhance performance. After each cycle, the model is evaluated using metrics like precision, recall, and F1 score to measure its effectiveness. This phase often involves refining the model by adjusting training parameters or even the model architecture to achieve the best results.

4. Model Deployment: This step involves utilizing the best model identified during the project. This can be achieved by integrating it into a larger application or using it as a standalone tool for NER tasks.

The benefits of custom models include the ability to customize, control, and adapt to specific needs, enabling the recognition of domain-specific entities. However, challenges include being resource-intensive and time-consuming, requiring substantial computational resources and a large, annotated dataset.

3.5 Evaluation metrics

To evaluate the performance of the NER model, we use metrics such as precision, recall, and F1-score. The F1-score, for example, can be calculated using the following formula:

$$F1 = 2 \times (Precision \times Recall) / (Precision + Recall)$$
Where:
$$Precision = TP / (TP + FP)$$
$$Recall = TP / (TP + FN)$$

In this context, TP represents the number of true positives, FP is the number of false positives, and FN is the number of false negatives.

The F1-score serves as a balanced measure of precision and recall, providing a thorough understanding of the model's effectiveness in identifying entities.

3.6 Mathematical Formulation for Active Learning

The confidence-based random sampling strategy can be described as follows:

Uncertainty Calculation:

$U(xi) = H(xi) = -\sum y\ P(y \mid xi) \log P(y \mid xi)$

Confidence Thresholding:

$S = \{xi \in U \mid U(xi) > C\}$

Random Sampling if Necessary:

If $S = \varnothing$, select k samples randomly from U.

Update Labeled Dataset:

$L \leftarrow L \cup S$

Retrain Model:

$M \leftarrow$ Train (M, L)

Let's denote:

U as the unlabelled dataset.

L as the labelled dataset.

M as the machine learning model.

P(y | x) as the probability of class y given input x.

3.7 Mathematical Formulation for Cross-Domain Generalization

To assess a model's ability to generalize across different domains, metrics like domain adaptation accuracy can be utilized. This process involves training a model on one domain and evaluating its performance on another.

The accuracy can be computed using the formula:

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

Key:

TP is the number of true positives
TN is the number of true negatives
FP is the number of false positives
FN is the number of false negatives

Achieving high accuracy across various domains suggests strong generalization capabilities.

3.8 Libraries/Packages

```
import spacy
import random
from spacy.matcher import Matcher, PhraseMatcher
from spacy.training import Example
import requests
import json
import os
import numpy as np
from spacy.training.iob_utils import offsets_to_biluo_tags
from spacy.scorer import Scorer
from spacy import displacy
import pickle
```

Figure 1: This shows all the imported libraries and packages that were used in our model to perform our defined functions.

The following table explains the nature of the above as well as their use in our code.

SpaCy	Python library specializing in Natural Language Processing (NLP)
Random	Pulls random samples from the dataset to be manually annotated and become our train dataset
Matcher/PhraseMatcher	SpaCy's Matcher and Phrase Matcher are tools used to find specific patterns or phrases within text. Both are useful for natural language processing (NLP) tasks such as information extraction, text analysis, and pattern matching, but they serve different purposes and work in slightly different ways.
Example	In spaCy, the Example class from spacy.training is a data structure used to represent training examples for a model,

	specifically for supervised training tasks like Named Entity Recognition (NER), part-of-speech tagging, and text classification.
Requests	Pulls data from a given URL
Json	JSON (JavaScript Object Notation) is a lightweight, text-based format used for storing and transmitting data objects consisting of attribute-value pairs. It is commonly used in APIs, configuration files, and data exchange between systems. JSON is easy for both humans to read and write and machines to parse and generate
OS	Operating system
Numpy	Python library specializing in numerical operations.
Biluo_tags	BILUO stands for Beginning, Inside, Last, Unit, Outside, and it's a tagging scheme used to represent the boundaries of entities in Named Entity Recognition (NER). It's useful when you want to denote multi-token entities clearly, such as "New York" or "United States". B (Beginning): Marks the first token of an entity. I (Inside): Marks tokens inside an entity, except the first one. L (Last): Marks the last token of an entity. U (Unit): Marks single-token entities. O (Outside): Marks tokens that are not part of any entity.
Scorer	The scorer is a utility in spaCy for evaluating the performance of models on specific tasks, like Named Entity Recognition (NER), part-of-speech tagging, or dependency parsing. The Scorer computes various metrics to assess the model’s accuracy, including Precision, Recall, and F1-Score, for the different labels or entities in your annotations.

Displacy	DisplaCy is a built-in visualization tool in spaCy for displaying the results of your NLP model. It helps visualize syntactic structures, named entities, or custom visualizations, making it easier to understand how the model processes and interprets text.
Pickle	Assists with file handling

4. Results

The results obtained showed that our model had a 20 - 30% confidence level, considering the challenges the dataset poses and time constraint, our group agreed to continue with this model and brainstorm future corrections to further optimize our model. We will discuss this in more detail in the conclusion.

4.1 Train Dataset Performance Metrics

Overall Metrics: Precision=0.96, Recall=0.76,F1=0.85

Per Entity Type:

geo-loc: Precision=0.97, Recall=0.76, F1=0.85

facility: Precision=0.97, Recall=0.71, F1=0.82

movie: Precision=0.96, Recall=0.76, F1=0.85

company: Precision=0.97, Recall=0.81, F1=0.88

product: Precision=0.96, Recall=0.84, F1=0.90

person: Precision=0.98, Recall=0.78, F1=0.87

other: Precision=0.92, Recall=0.76, F1=0.83

sportsteam: Precision=0.95, Recall=0.75, F1=0.84

tvshow: Precision=0.91, Recall=0.62, F1=0.74

musicartist: Precision=0.93, Recall=0.69, F1=0.79

4.2 Dev Dataset Performance Metrics

Overall Metrics: Precision=0.30, Recall=0.12,F1=0.18

Per Entity Type:

other: Precision=0.18, Recall=0.10, F1=0.13

company: Precision=0.27, Recall=0.21, F1=0.23

geo-loc: Precision=0.44, Recall=0.24, F1=0.31

product: Precision=0.12, Recall=0.05, F1=0.07

facility: Precision=0.14, Recall=0.05, F1=0.08

person: Precision=0.37, Recall=0.15, F1=0.21

sportsteam: Precision=0.60, Recall=0.04, F1=0.08

musicartist: Precision=0.00, Recall=0.00, F1=0.00

movie: Precision=0.50, Recall=0.07, F1=0.12

tvshow: Precision=0.00, Recall=0.00, F1=0.00

4.3 Test Dataset Performance Metrics

Overall Metrics: Precision=0.35, Recall=0.15,F1=0.21

Per Entity Type:

other: Precision=0.17, Recall=0.08, F1=0.11

movie: Precision=0.00, Recall=0.00, F1=0.00

person: Precision=0.25, Recall=0.18, F1=0.21

geo-loc: Precision=0.55, Recall=0.30, F1=0.39

company: Precision=0.44, Recall=0.12, F1=0.19

product: Precision=0.08, Recall=0.03, F1=0.04

musicartist: Precision=0.06, Recall=0.01, F1=0.01

sportsteam: Precision=0.08, Recall=0.01, F1=0.01

facility: Precision=0.42, Recall=0.15, F1=0.22

tvshow: Precision=0.00, Recall=0.00, F1=0.00

5. Conclusion

This project has given our group extensive knowledge, as well as first hand experience, of Named Entity Recognition and its important role in machine learning and pattern recognition. Some observations we made that may improve our models efficiency are; parameter fine tuning, more specified rules/labels, condensing our code, reducing loss, and additional iterations of manual annotations.

Moreover, we have gained an understanding of the plethora of benefits that come from using SpaCy for NER and Active Learning.

While our results do not inherently show a good confidence percentile, as a group, we have used all our combined efforts to create this model. It is our hope that by the end of the program we will be able to not only increase our models' confidence, but also write our code more concisely.

5.1 Benefit of Using SpaCy for NER

Efficient Entity Recognition is a strong suit of spaCy's models as they are designed for high efficiency and accuracy, ensuring that relevant entities are effectively identified. Customization is another great feature as the pipeline allows for customization to target specific entity types or to integrate with other NLP tasks seamlessly. Furthermore, spaCy's scalability models are capable of processing large volumes of data, making them ideal for real-world applications.

5.2 Benefits of Active Learning

Efficient Use of Resources allows Active learning to focus on human annotation efforts on the most informative samples, which minimizes the need for extensive labeling. Additionally, improved model performance is achieved by continuously updating the model with new annotations, active learning boosts model accuracy over time.This adaptability method enables models to adjust to changing data distributions or new entity types as they arise.

Active learning, when paired with spaCy's EntityRuler, presents a strong strategy for named entity recognition (NER) on noisy datasets like WNUT-16. Utilizing these techniques allows for the creation of robust and efficient models capable of accurately identifying entities in real-world data.

5.3 Future Directions

Some observations the group has made to boost efficiency and consistency are as follows;

- Fine tuning parameters: Taking a closer look at the iterations, finding outliers, and fine tuning them to increase confidence.

- Specifying more rules/labels:
- Simplifying our approach: attempting to condense lines of code to decrease the chance of error.
- Trying to reduce loss - deviation from datapoint.
- Manually annotating more rigidly and increasing the iterations as well as quantity of samples used.

6. Code

```
import spacy

import random

from spacy.matcher import Matcher, PhraseMatcher

from spacy.training import Example

import requests

import json

import os

import numpy as np

from spacy.training.iob_utils import offsets_to_biluo_tags

from spacy.scorer import Scorer

from spacy import displacy

import pickle

#1

def load_data(filepath, pure_text=False):

   """

   Load the WNUT16 dataset, supporting both plain text and labeled modes.



   Parameters:

       filepath: Path to the data file.

       pure_text: If True, only load plain text; otherwise, load data with
BIO annotations.
```

```
    Returns:
        sentences: List of sentences.
        labeled_sentences: List of labeled sentences in the format (text, {"entities": [(start, end, label)]}).
    """
    sentences = []
    labeled_sentences = []
    response = requests.get(filepath)
    blocks = response.text.strip().split('\n\n')

    for block in blocks:
        if not block.strip():
            continue
        lines = block.split('\n')
        tokens = []
        labels = []
        for line in lines:
            if line.strip() and '\t' in line:
                token, label = line.split('\t')
                tokens.append(token.strip())
                labels.append(label.strip())
            else:
                continue

        if not tokens or not labels:
            continue
```

```
        text = " ".join(tokens)
        if pure_text:
            sentences.append(text)
            labeled_sentences.append((text, {"entities": []}))
            continue

        char_positions = []
        current_pos = 0
        for token in tokens:
            char_positions.append(current_pos)
            current_pos += len(token) + 1

        entities = []
        seen_entities = set()
        start = None
        current_label = None
        for i in range(len(tokens)):
            label = labels[i]
            if label.startswith("B-"):
                if current_label is not None:
                    start_char = char_positions[start]
                    end_char = char_positions[i - 1] + len(tokens[i - 1])
                    entity_key = (start, i - 1, current_label)
                    if entity_key not in seen_entities:
                        seen_entities.add(entity_key)
                        entities.append((start_char, end_char,
current_label))
```

```
                start = i
                current_label = label[2:]
            elif label.startswith("I-") and current_label:
                expected_label = label[2:]
                if expected_label != current_label:
                    print(f"Warning: Mismatched I- label found for '{text}'
at position {i}: Expected {current_label}, but got {expected_label}")
                continue
            else:
                if current_label is not None:
                    start_char = char_positions[start]
                    end_char = char_positions[i - 1] + len(tokens[i - 1])
                    entity_key = (start, i - 1, current_label)
                    if entity_key not in seen_entities:
                        seen_entities.add(entity_key)
                        entities.append((start_char, end_char,
current_label))
                start = None
                current_label = None

        if current_label is not None:
            start_char = char_positions[start]
            end_char = char_positions[-1] + len(tokens[-1])
            entity_key = (start, len(tokens) - 1, current_label)
            if entity_key not in seen_entities:
                seen_entities.add(entity_key)
                entities.append((start_char, end_char, current_label))
```

```
            sentences.append(text)
            labeled_sentences.append((text, {"entities": entities}))

    return sentences, labeled_sentences

# 1.2 Sampling Function (Confidence-Based and Random Selection)
def sample_sentences(unlabeled_sentences, n_confidence, n_random, ner):
    """
    Mix confidence-based and random selection of sentences for active learning sampling.

    Parameters:
        unlabeled_sentences: List of unlabeled sentences.
        n_confidence: Number of sentences selected based on confidence.

        n_random: Number of sentences selected randomly.

        ner: NER model component.

    Returns:
        List of sampled sentences.
    """

    def compute_confidence_score(doc, ner):
        """
        Compute confidence score for uncertainty sampling """
        entity_scores = []
```

```
        try:
            doc_with_entities = ner(doc)
            spans = doc_with_entities.ents
            model_output = ner.model.predict([doc])
            logits = model_output.logits if hasattr(model_output, 'logits')
else model_output
            if isinstance(logits, (list, tuple, np.ndarray)) and
len(logits) > 0:
                logits = logits[0] if isinstance(logits[0], (list, tuple,
np.ndarray)) else logits
            else:
                logits = np.ones((len(doc), len(ner.labels)))
            logits_np = np.asarray(logits)
            def numpy_softmax(x, axis=-1):
                exp_x = np.exp(x - np.max(x, axis=axis, keepdims=True))
                return exp_x / np.sum(exp_x, axis=axis, keepdims=True)
            probs = numpy_softmax(logits_np, axis=-1)
            for span in spans:
                probs_span = probs[span.start:span.end]
                max_prob = np.max(probs_span) if len(probs_span) > 0 else
1.0
                weighted_prob = max_prob * (span.end - span.start)
                entity_scores.append(weighted_prob)
        except Exception as e:
            entity_scores.append(1.0)
        return sum(entity_scores) / len(entity_scores) if entity_scores
else 1.0
```

```
    random_samples = random.sample(unlabeled_sentences, min(n_random,
len(unlabeled_sentences)))
    remaining_sentences = [s for s in unlabeled_sentences if s not in
random_samples]


    scores = []
    for sentence in remaining_sentences:
        doc = nlp.make_doc(sentence)
        avg_score = compute_confidence_score(doc, ner)
        scores.append((sentence, avg_score))


    scores.sort(key=lambda x: x[1])
    print("\nTop 5 sentences with the lowest confidence:")
    for sentence, score in scores[:5]:
        print(f"Sentence: {sentence},  Confidence Score: {score}")
    confidence_samples = [s[0] for s in scores[:n_confidence]]
    return confidence_samples + random_samples



# 1.3 Import and Process Annotated Data
def import_annotated_data(nlp, annotated_file, priority,
low_freq_categories=None):
    """
    Import and process annotated data, converting it into SpaCy format
while avoiding forced ordering.


    Parameters:
        nlp: SpaCy model.
```

```
        annotated_file: Path to the annotated file (Label Studio format).
        priority: Dictionary of entity priorities (for reference only).
        low_freq_categories: List of low-frequency categories (prioritized for retention).

    Returns:
        List of newly annotated data in the format [(text, {"entities": [(start, end, label)]})].
    """
    low_freq_categories = low_freq_categories or []
    with open(annotated_file, "r", encoding="utf-8") as f:
        annotated_data = json.load(f)

    new_labeled_data = []
    total_entities = 0
    ignored_entities = 0
    for item in annotated_data:
        if "data" in item and "text" in item["data"]:
            text = item["data"]["text"]
            annotations = item.get("annotations", [{}])[0].get("result", [])
        elif "text" in item:
            text = item["text"]
            annotations = item.get("annotations", [{}])[0].get("result", [])
        else:
            raise ValueError(f"Failed to extract 'text' from annotated data, incorrect item format: {item}")
```

```
        doc = nlp.make_doc(text)

        entities_list = []

        for r in annotations:

            char_start = r["value"]["start"]

            char_end = r["value"]["end"]

            label = r["value"]["labels"][0]

            total_entities += 1

            span = doc.char_span(char_start, char_end, label=label,
alignment_mode="expand")

            if span is not None:

                token_start = span.start

                token_end = span.end

                aligned_char_start = doc[token_start].idx

                aligned_char_end = doc[token_end-1].idx +
len(doc[token_end-1].text) if token_end > token_start else
aligned_char_start + len(doc[token_start].text)

                entities_list.append((aligned_char_start, aligned_char_end,
label))

            else:

                print(f"Failed to convert character span, Sentence: {text},
Entity: {char_start, char_end, label}")


        # Sort by start and end positions, but do not sort by priority.

        entities_list.sort(key=lambda x: (x[0], x[1]))


        # Conflict resolution: Prioritize retaining entities of
low-frequency categories to avoid forced removal.


        entities = []
```

```
        used_tokens = set()
        for char_start, char_end, label in entities_list:
            span = doc.char_span(char_start, char_end, label=label,
alignment_mode="expand")
            if span is None:
                continue
            start = span.start
            end = span.end
            overlap = False
            for i in range(start, end):
                if i in used_tokens:
                    overlap = True
                    break
            # If there is an overlap, check if it belongs to a
low-frequency category.


            if overlap:
                # If the current entity is a low-frequency category,
prioritize its retention.
                if label in low_freq_categories:
                    # Remove previously conflicting entities and reassign
tokens.
                    new_entities = []
                    new_used_tokens = set()
                    for e in entities:
                        e_span = doc.char_span(e[0], e[1], label=e[2],
alignment_mode="expand")
                        if e_span is None:
                            continue
```

```
                        e_start = e_span.start
                        e_end = e_span.end
                        e_overlap = False
                        for j in range(e_start, e_end):
                            if j in range(start, end):
                                e_overlap = True
                                break
                        if not e_overlap or (e_overlap and e[2] not in
low_freq_categories):
                            new_entities.append(e)
                            for j in range(e_start, e_end):
                                new_used_tokens.add(j)
                    entities = new_entities
                    used_tokens = new_used_tokens
                    overlap = False
            if not overlap:
                entities.append((char_start, char_end, label))
                for i in range(start, end):
                    used_tokens.add(i)


        try:
            biluo_tags = offsets_to_biluo_tags(doc, entities)
            valid_entities = []
            for char_start, char_end, label in entities:
                span = doc.char_span(char_start, char_end, label=label,
alignment_mode="expand")
                if span and all(biluo_tags[i] != "-" for i in
range(span.start, span.end)):
```

```
                    valid_entities.append((char_start, char_end, label))
                else:
                    print(f"Skipping misaligned entity, Sentence: '{text}',
Entity: {char_start, char_end, label}")
                    ignored_entities += 1
            new_labeled_data.append((text, {"entities": valid_entities}))
        except ValueError as e:
            print(f"Error aligning entities, Sentence: '{text}': {e}")
            new_labeled_data.append((text, {"entities": []}))

    print(f"Loaded {len(new_labeled_data)} new labeled samples.")
    print(f"Total entities: {total_entities}, Ignored entities:
{ignored_entities}, Ignored ratio: {ignored_entities/total_entities:.2%}")
    if new_labeled_data:
        print("Sample:", new_labeled_data[0])
    return new_labeled_data


# 1.4 Train the Model
def train_model(nlp, train_data, other_pipes, optimizer, epochs=30,
dropout=0.15):
    """
    Train the model.

    Parameters:
        nlp: SpaCy model.

        train_data: Training data in the format [(text, {"entities":
[(start, end, label)]})].
```

```
        other_pipes: Other pipelines to disable.

        optimizer: Optimizer.

        epochs: Number of training epochs.

        dropout: Dropout rate.

    Returns:
        The trained model.
    """

    print("Starting model training")
    with nlp.disable_pipes(*other_pipes):
        for i in range(epochs):
            random.shuffle(train_data)
            losses = {}
            for text, annotations in train_data:
                doc = nlp.make_doc(text)
                valid_entities = [(start, end, label) for start, end, label
in annotations["entities"]
                                  if start >= 0 and end <= len(doc.text)
and start < end]
                example = Example.from_dict(doc, {"entities":
valid_entities})
                nlp.update([example], drop=dropout, sgd=optimizer,
losses=losses)
```

```
            print(f" {i + 1} : {losses}")
    return nlp


# 1.5 Evaluate the Model
def evaluate_model(nlp, data, dataset_name="Dev"):
    """
    Evaluate model performance.

    Parameters:
        nlp: SpaCy model.

        data: Evaluation data in the format [(text, {"entities": [(start,
end, label)]})].

        dataset_name: Name of the dataset (for printing purposes).

    Returns:

        Evaluation scores.
    """
    scorer = spacy.scorer.Scorer()
    examples = []
    for text, annotations in data:
        doc = nlp(text)
        example = Example.from_dict(doc, annotations)
        examples.append(example)
    scorer.score(examples)
```

```
    print(f"\n{dataset_name} Metrics : ")
    print(f"Overall : {scorer.scores}")
    print("\nPer Entity Type : ")
    for entity_type, metrics in scorer.scores["ents_per_type"].items():
        print(f"{entity_type}: {metrics}")
    return scorer.scores
base_url =
"https://raw.githubusercontent.com/aritter/twitter_nlp/master/data/annotated/wnut16/data/"
priority = {
    "company": 1, "facility": 2, "geo-loc": 3, "movie": 4, "musicartist":
5,
    "person": 6, "product": 7, "sportsteam": 8, "tvshow": 9, "other": 10
}


print("Preprocessing data")
files = ["train", "dev", "test"]
data = {}
for file in files:
    file_url = f"{base_url}{file}"
    sentences, labeled_sentences = load_data(file_url, pure_text=True)
    data[file] = sentences
print(f"train: {len(data['train'])}, dev: {len(data['dev'])}, test:
{len(data['test'])}")
print("\nCreating model")
nlp = spacy.blank("en")
ner = nlp.add_pipe("ner")
entity_types = ["company", "facility", "geo-loc", "movie", "musicartist",
"other", "person", "product", "sportsteam", "tvshow"]
```

```
for label in entity_types:
   ner.add_label(label)
nlp.initialize()
nlp.to_disk("initial_ner_model")
print("Initial model has been saved as 'initial_ner_model'")
print("\nGenerating pseudo labels based on rules")
matcher = Matcher(nlp.vocab)
phrase_matcher = PhraseMatcher(nlp.vocab, attr="LOWER")
matcher.add("person", [[{"IS_UPPER": True}, {"IS_UPPER": True}]])
matcher.add("company", [[{"TEXT": {"REGEX": r"(Inc\.|Ltd\.|Corp\.)$"}}]])
matcher.add("facility", [[{"IS_UPPER": True}, {"LOWER": {"IN": ["stadium",
"airport", "museum"]}}]])



geo_loc_dict = ["new york", "london", "tokyo", "shanghai", "paris", "los
angeles", "california", "texas", "florida", "chicago"]
movie_dict = ["the matrix", "titanic", "inception", "star wars", "the
godfather", "pulp fiction", "the avengers"]



phrase_matcher.add("geo-loc", [nlp.make_doc(loc) for loc in geo_loc_dict])
phrase_matcher.add("movie", [nlp.make_doc(movie) for movie in movie_dict])



excluded_tokens = ["RT", ":D", "...", "ALL", "FML", "KK", "TIME", "LOW",
"I", "WARNING", "do", "IN", "MY", "IS", "DONE"]


sample_size = 100
```

```
sampled_sentences = random.sample(data["train"], min(sample_size,
len(data["train"])))

train_data = []


for sentence in sampled_sentences:

   doc = nlp(sentence)

   entities_with_priority = []

   for match_id, start, end in matcher(doc):

       label = nlp.vocab.strings[match_id]

       if any(token in excluded_tokens for token in
doc[start:end].text.split()):

           continue

       entities_with_priority.append((start, end, label, priority[label]))

   for match_id, start, end in phrase_matcher(doc):

       label = nlp.vocab.strings[match_id]

       entities_with_priority.append((start, end, label, priority[label]))


   entities_with_priority.sort(key=lambda x: (x[0], x[1], x[3]))

   entities = []

   used_tokens = set()

   for start, end, label, _ in entities_with_priority:

       if start < 0 or end > len(doc) or start >= end:

           continue

       overlap = any(i in used_tokens for i in range(start, end))

       if not overlap:

           char_start = doc[start].idx

           char_end = doc[end-1].idx + len(doc[end-1].text) if end > start
else char_start + len(doc[start].text)
```

```
            if char_end > len(doc.text):
                char_end = len(doc.text)
            span = doc.char_span(char_start, char_end, label=label,
alignment_mode="strict")
            if span:
                entities.append((char_start, char_end, label))
                for i in range(start, end):
                    used_tokens.add(i)
    try:
        biluo_tags = offsets_to_biluo_tags(doc, entities)
        valid_entities = [(char_start, char_end, label) for char_start,
char_end, label in entities
                          if char_start >= 0 and char_end <= len(doc.text)
and char_start < char_end and
                          all(biluo_tags[i] != "-" for i in
range(doc.char_span(char_start, char_end, label=label,
alignment_mode="expand").start,
                                doc.char_span(char_start, char_end,
label=label, alignment_mode="expand").end))]
        train_data.append((sentence, {"entities": valid_entities}))
    except ValueError as e:
        print(f"Error aligning entities, Sentence: '{sentence}': {e}")
        train_data.append((sentence, {"entities": []}))

with open("initial_pseudo_labels.json", "w", encoding="utf-8") as f:
    json.dump(train_data, f, ensure_ascii=False)
print(f"Generated {len(train_data)} pseudo-labeled samples.")
```

```
print("\nInitial Training")

nlp = spacy.load("initial_ner_model")

other_pipes = [pipe for pipe in nlp.pipe_names if pipe != "ner"]

optimizer = nlp.begin_training()

nlp = train_model(nlp, train_data, other_pipes, optimizer, epochs=10)

nlp.to_disk("initial_trained_model")

unlabeled_sentences = [s for s in data["train"] if s not in [t[0] for t in
train_data]]

with open("train_data.json", "w", encoding="utf-8") as f:

   json.dump(train_data, f, ensure_ascii=False)

with open("unlabeled_sentences.json", "w", encoding="utf-8") as f:

   json.dump(unlabeled_sentences, f, ensure_ascii=False)

print("Initial training completed. Model saved as
'initial_trained_model'")

def run_first_iteration(iteration, model_name, export_file, n_confidence,
n_random , epochs=15, dropout = 0.3):

   """

   First Iteration: Sampling and exporting samples (without manual
annotation file)


   Parameters:

       iteration: Iteration number (used for printing steps)

       model_name: Current model name (e.g., "initial_trained_model")

       export_file: File name for exported samples (e.g.,
"manual_samples_first.json")

       n_confidence: Number of confidence-based samples

       n_random: Number of randomly selected samples


   Returns:
```

```
        None
    """
    print(f"\nStep {2*iteration+5} : Active Learning Iteration {iteration} - Exporting data")
    global nlp, train_data, unlabeled_sentences


    # Load model
    nlp = spacy.load(model_name)
    ner = nlp.get_pipe("ner")


    # Load train_data and unlabeled_sentences
    with open("train_data.json", "r", encoding="utf-8") as f:
        train_data = json.load(f)
    with open("unlabeled_sentences.json", "r", encoding="utf-8") as f:
        unlabeled_sentences = json.load(f)


    # Sample sentences
    samples = sample_sentences(unlabeled_sentences, n_confidence, n_random, ner)
    with open(export_file, "w", encoding="utf-8") as f:
        json.dump([{"text": s} for s in samples], f, ensure_ascii=False)
    unlabeled_sentences = [s for s in unlabeled_sentences if s not in samples]
    with open("train_data.json", "w", encoding="utf-8") as f:
        json.dump(train_data, f, ensure_ascii=False)
    with open("unlabeled_sentences.json", "w", encoding="utf-8") as f:
        json.dump(unlabeled_sentences, f, ensure_ascii=False)
```

```
    print(f"Exported {n_confidence + n_random} samples to '{export_file}'.
Please annotate manually.")

def run_subsequent_iteration(iteration, model_name, export_file,
annotated_file, n_confidence, n_random, save_model_name, epochs=30,
dropout=0.15, low_freq_categories=None, dev_url=None, target_f1=85.0,
max_iterations=10, f1_history=None):

    """

    Subsequent Iterations: Sampling, importing annotated data, training
(single iteration), and evaluating performance.



    Parameters:

    iteration: Iteration number (used for printing steps)

    model_name: Current model name (e.g., "initial_trained_model" or
"active_learning_ner_model_iteration_1")

    export_file: File name for exported samples (e.g.,
"uncertain_samples_iteration_1.json")

    annotated_file: Manually annotated file name (e.g.,
"manual_annotated_first.json" or "annotated_samples_iteration_1.json")

    n_confidence: Number of confidence-based samples

    n_random: Number of randomly selected samples

    save_model_name: Name of the saved model (e.g.,
"active_learning_ner_model_iteration_1")

    epochs: Number of training epochs

    """

    print(f"\nStep {iteration}：Active Learning Iteration {iteration} -
Sampling, Training")

    global nlp, train_data, unlabeled_sentences



    # Load model

    nlp = spacy.load(model_name)
```

```
    ner = nlp.get_pipe("ner")
    other_pipes = [pipe for pipe in nlp.pipe_names if pipe != "ner"]

    # Load optimizer state
    optimizer_file = f"{model_name}_optimizer.pkl"
    if os.path.exists(optimizer_file):
        with open(optimizer_file, "rb") as f:
            optimizer = pickle.load(f)
        print(f"Loaded optimizer state from '{optimizer_file}'.")
    else:
        optimizer = nlp.begin_training()
        optimizer.learn_rate = 0.0001
        print("Initialized new optimizer with learn_rate=0.0001.")

    # Load train_data and unlabeled_sentences
    with open("train_data.json", "r", encoding="utf-8") as f:
        train_data = json.load(f)
    with open("unlabeled_sentences.json", "r", encoding="utf-8") as f:
        unlabeled_sentences = json.load(f)

    # Sample sentences
    samples = sample_sentences_subsequent(unlabeled_sentences,
n_confidence, n_random, ner, low_freq_categories=low_freq_categories)
    with open(export_file, "w", encoding="utf-8") as f:
        json.dump([{"text": s} for s in samples], f, ensure_ascii=False)
    unlabeled_sentences = [s for s in unlabeled_sentences if s not in
samples]
    with open("unlabeled_sentences.json", "w", encoding="utf-8") as f:
```

```
        json.dump(unlabeled_sentences, f, ensure_ascii=False)
    print(f"Exported {n_confidence + n_random} samples to '{export_file}'. Please annotate '{annotated_file}', 然后继续下一次迭代。")

    # Check if annotated_file exists
    if not os.path.exists(annotated_file):
        raise FileNotFoundError(f"Annotated file '{annotated_file}' not found. Please ensure it has been annotated.")

    # Load new annotated data
    new_labeled_data = import_annotated_data(nlp, annotated_file, priority, low_freq_categories=low_freq_categories if low_freq_categories is not None else [])
    train_data.extend(new_labeled_data)

    # Train model
    print(f"Training data size:  {len(train_data)}")
    nlp = train_model(nlp, train_data, other_pipes, optimizer, epochs=epochs, dropout=dropout)

    # Save model and optimizer state
    nlp.to_disk(save_model_name)
    with open(f"{save_model_name}_optimizer.pkl", "wb") as f:
        pickle.dump(optimizer, f)
    with open("train_data.json", "w", encoding="utf-8") as f:
        json.dump(train_data, f, ensure_ascii=False)
    with open("unlabeled_sentences.json", "w", encoding="utf-8") as f:
        json.dump(unlabeled_sentences, f, ensure_ascii=False)
```

```
    print(f"Iteration {iteration} completed: Model saved as
'{save_model_name}'。")
    return save_model_name, True


def sample_sentences_subsequent(unlabeled_sentences, n_confidence,
n_random, ner, low_freq_categories=None):
    """
    Mix confidence-based and random sentence sampling, with support for
prioritizing low-frequency categories.

    Parameters:
        unlabeled_sentences: List of unlabeled sentences
        n_confidence: Number of confidence-based samples
        n_random: Number of randomly selected samples
        ner: NER model component

    Returns:
        List of selected samples
    """
    def compute_confidence_score(doc, ner):
        entity_scores = []
        entities_detected = []
        try:
            doc_with_entities = ner(doc)
            spans = doc_with_entities.ents
            model_output = ner.model.predict([doc])
            logits = model_output.logits if hasattr(model_output, 'logits')
else model_output
```

```
            if isinstance(logits, (list, tuple, np.ndarray)) and
len(logits) > 0:
                logits = logits[0] if isinstance(logits[0], (list, tuple,
np.ndarray)) else logits
            else:
                logits = np.ones((len(doc), len(ner.labels)))
            logits_np = np.asarray(logits)
            def numpy_softmax(x, axis=-1):
                exp_x = np.exp(x - np.max(x, axis=axis, keepdims=True))
                return exp_x / np.sum(exp_x, axis=axis, keepdims=True)
            probs = numpy_softmax(logits_np, axis=-1)
            for span in spans:
                probs_span = probs[span.start:span.end]
                max_prob = np.max(probs_span) if len(probs_span) > 0 else
1.0
                weighted_prob = max_prob * (span.end - span.start)
                entity_scores.append(weighted_prob)
                entities_detected.append(span.label_)
        except Exception as e:
            entity_scores.append(1.0)
        return sum(entity_scores) / len(entity_scores) if entity_scores
else 1.0, entities_detected


    # Random sampling
    random_samples = random.sample(unlabeled_sentences, min(n_random,
len(unlabeled_sentences)))
    remaining_sentences = [s for s in unlabeled_sentences if s not in
random_samples]
```

```
    # Prioritize sentences containing low-frequency categories
    low_freq_samples = []
    other_samples = []

    for sentence in remaining_sentences:
        doc = nlp.make_doc(sentence)
        avg_score, entities_detected = compute_confidence_score(doc, ner)
        # Check if the sentence contains entities from low-frequency
categories
        if low_freq_categories and any(entity in low_freq_categories for
entity in entities_detected):
            low_freq_samples.append((sentence, avg_score))
        else:
            other_samples.append((sentence, avg_score))

    # Sort by confidence score
    low_freq_samples.sort(key=lambda x: x[1])
    other_samples.sort(key=lambda x: x[1])

    # Prioritize sampling sentences from low-frequency categories
    n_low_freq = min(len(low_freq_samples), int(n_confidence * 0.8))
    n_other = n_confidence - n_low_freq

    scores = low_freq_samples + other_samples
    print("\nTop 5 sentences with the lowest confidence：")
    for sentence, score in scores[:5]:
        print(f"Sentence: {sentence}, Confidence Score: {score}")
```

```
    confidence_samples = [s[0] for s in low_freq_samples[:n_low_freq]] +
[s[0] for s in other_samples[:n_other]]


    if low_freq_categories:
        target_min_count = 30
        category_counts = {cat: 0 for cat in low_freq_categories}


        for sentence in confidence_samples:
            doc = nlp.make_doc(sentence)
            doc_with_entities = ner(doc)
            for ent in doc_with_entities.ents:
                if ent.label_ in low_freq_categories:
                    category_counts[ent.label_] += 1


        print("\nLow-Frequency Category Sampling Statistics:")
        for cat, count in category_counts.items():
            print(f"{cat}: {count} samples")
            if count < target_min_count:
                print(f"Warning: The number of samples for category {cat}
({count}) did not reach the target ({target_min_count}), . Resampling is
recommended.")


    return confidence_samples + random_samples




un_first_iteration(
    iteration=1,
    model_name="initial_trained_model",
```

```
    export_file="manual_samples_1.json",
    n_confidence=10,
    n_random=90,
    epochs=20,
    dropout=0.3
)
run_subsequent_iteration(
    iteration=2,
    model_name="initial_trained_model",
    export_file="manual_samples_2.json",
    annotated_file="manual_annotated_05_01.json",
    n_confidence=10,
    n_random=90,
    save_model_name="active_learning_ner_model_iteration_1",
    epochs=20,
    dropout=0.3
)
###1-2
run_subsequent_iteration(
    iteration=3,
    model_name="active_learning_ner_model_iteration_1",
    export_file="manual_samples_3.json",
    annotated_file="manual_annotated_05_02.json",
    n_confidence=30,
    n_random=70,
    save_model_name="active_learning_ner_model_iteration_2",
    epochs=20,
```

```
    dropout=0.3
)
###2-3
run_subsequent_iteration(
    iteration=4,
    model_name="active_learning_ner_model_iteration_2",
    export_file="manual_samples_4.json",
    annotated_file="manual_annotated_05_03.json",
    n_confidence=40,
    n_random=60,
    save_model_name="active_learning_ner_model_iteration_3",
    epochs=20,
    dropout=0.3
)
###3-4
run_subsequent_iteration(
    iteration=5,
    model_name="active_learning_ner_model_iteration_3",
    export_file="manual_samples_5.json",
    annotated_file="manual_annotated_05_04.json",
    n_confidence=60,
    n_random=60,
    save_model_name="active_learning_ner_model_iteration_4",
    epochs=20,
    dropout=0.25
)
###4-5
```

```
run_subsequent_iteration(
    iteration=6,
    model_name="active_learning_ner_model_iteration_4",
    export_file="manual_samples_6.json",
    annotated_file="manual_annotated_05_05.json",
    n_confidence=90,
    n_random=30,
    save_model_name="active_learning_ner_model_iteration_5",
    epochs=20,
    dropout=0.25
)
###5-6
run_subsequent_iteration(
    iteration=7,
    model_name="active_learning_ner_model_iteration_5",
    export_file="manual_samples_7.json",
    annotated_file="manual_annotated_05_06.json",
    n_confidence=100,
    n_random=30,
    save_model_name="active_learning_ner_model_iteration_6",
    epochs=20,
    dropout=0.2
)
###6-7
run_subsequent_iteration(
    iteration=8,
    model_name="active_learning_ner_model_iteration_6",
```

```
    export_file="manual_samples_8.json",
    annotated_file="manual_annotated_05_07.json",
    n_confidence=120,
    n_random=30,
    save_model_name="active_learning_ner_model_iteration_7",
    epochs=20,
    dropout=0.2,
    low_freq_categories =  ["company", "facility", "geo-loc", "movie",
"musicartist",
    "person", "product", "sportsteam", "tvshow", "other"]
)
###7-8
run_subsequent_iteration(
    iteration=9,
    model_name="active_learning_ner_model_iteration_7",
    export_file="manual_samples_9.json",
    annotated_file="manual_annotated_05_08.json",
    n_confidence=120,
    n_random=30,
    save_model_name="active_learning_ner_model_iteration_8",
    epochs=20,
    dropout=0.2,
    low_freq_categories =  ["company", "facility", "geo-loc", "movie",
"musicartist",
    "person", "product", "sportsteam", "tvshow", "other"]
)
###8-9
run_subsequent_iteration(
```

```
   iteration=10,
   model_name="active_learning_ner_model_iteration_8",
   export_file="manual_samples_10.json",
   annotated_file="manual_annotated_05_09.json",
   n_confidence=120,
   n_random=30,
   save_model_name="active_learning_ner_model_iteration_9",
   epochs=15,
   dropout=0.2,
   low_freq_categories =  ["company", "facility", "geo-loc", "movie",
"musicartist",
   "person", "product", "sportsteam", "tvshow", "other"]
)
###9-10
run_subsequent_iteration(
   iteration=11,
   model_name="active_learning_ner_model_iteration_9",
   export_file="manual_samples_11.json",
   annotated_file="manual_annotated_05_10.json",
   n_confidence=100,
   n_random=50,
   save_model_name="active_learning_ner_model_iteration_10",
   epochs=15,
   dropout=0.2,
   low_freq_categories =  ["company", "facility", "geo-loc", "movie",
"musicartist",
   "person", "product", "sportsteam", "tvshow", "other"]
)
```

```
###10-11
run_subsequent_iteration(
   iteration=12,
   model_name="active_learning_ner_model_iteration_10",
   export_file="manual_samples_12.json",
   annotated_file="manual_annotated_05_11.json",
   n_confidence=90,
   n_random=10,
   save_model_name="active_learning_ner_model_iteration_11",
   epochs=15,
   dropout=0.2,
   low_freq_categories =  ["company", "facility", "geo-loc", "movie",
"musicartist",
   "person", "product", "sportsteam", "tvshow", "other"]
)
###11-12
run_subsequent_iteration(
   iteration=13,
   model_name="active_learning_ner_model_iteration_11",
   export_file="manual_samples_13.json",
   annotated_file="manual_annotated_05_12.json",
   n_confidence=100,
   n_random=20,
   save_model_name="active_learning_ner_model_iteration_12",
   epochs=10,
   dropout=0.2,
   low_freq_categories =  ["company", "facility", "geo-loc", "movie",
"musicartist",
```

```
    "person", "product", "sportsteam", "tvshow", "other"]
)
run_subsequent_iteration(
    iteration=14,
    model_name="active_learning_ner_model_iteration_12",
    export_file="manual_samples_14.json",
    annotated_file="manual_annotated_05_13.json",
    n_confidence=120,
    n_random=30,
    save_model_name="active_learning_ner_model_iteration_13",
    epochs=10,
    dropout=0.2,
)
###13-14
def evaluate_model_on_datasets(model, datasets, dataset_names):
    results = {}
    for dataset, name in zip(datasets, dataset_names):
        scorer = spacy.scorer.Scorer()
        examples = []
        for text, annotations in dataset:
            doc = model(text)
            example = Example.from_dict(doc, annotations)
            examples.append(example)
        scores = scorer.score(examples)

        # Print performance metrics
        print(f"\n{name} Dataset Performance Metrics:")
```

```
        print(f"Overall Metrics: Precision={scores['ents_p']:.2f},
Recall={scores['ents_r']:.2f},F1={scores['ents_f']:.2f}")

        print("\nPer Entity Type:")

        for entity_type, metrics in scores["ents_per_type"].items():

            print(f"{entity_type}: Precision={metrics['p']:.2f},
Recall={metrics['r']:.2f}, F1={metrics['f']:.2f}")


        # Save results

        results[name] = scores


    return results

base_url =
"https://raw.githubusercontent.com/aritter/twitter_nlp/master/data/annotat
ed/wnut16/data/"

train_url = f"{base_url}train"

dev_url = f"{base_url}dev"

test_url = f"{base_url}test"


_, train_data = load_data(train_url)

_, dev_data = load_data(dev_url)

_, test_data = load_data(test_url)


model = spacy.load("active_learning_ner_model_iteration_10")


datasets = [train_data, dev_data, test_data]

dataset_names = ["Train", "Dev", "Test"]

results = evaluate_model_on_datasets(model, datasets, dataset_names)
```

```
with open("evaluation_results.json", "w", encoding="utf-8") as f:
    json.dump(results, f, ensure_ascii=False)
print("Test results have been saved to 'evaluation_results.json'。")
```

7. Appendix

The following is a summarization of minutes taken during group meetings, as well as each member's individual contribution to the project.

7.1 Bharath Shakthivel Contribution

Over the past month, I have made significant progress in developing the custom Named Entity Recognition (NER) model using spaCy. I have been actively involved in designing and implementing the NER model, ensuring it meets project requirements. My contributions included setting up and refining the spaCy pipeline, focusing on data preprocessing and fine-tuning to improve entity identification and classification. Additionally, I contributed to model testing and evaluation, identifying areas for improvement and optimizing based on the results. Continuous collaboration and code reviews helped resolve technical challenges and integrate new features.

Although the model's accuracy currently stands at 20-30%, these insights and experiences will be crucial for improving its performance in future iterations.

7.2 Harshitha Ramesh Contribution

Collecting and studying resources from different websites that could be helpful for our project, gathering information about some techniques we could include, and identifying potential challenges we may face and the benefits of using different datasets. Additionally, we are assisting each other in writing the project report.

7.3 Serene Itani Contribution

Compiling resources to reference throughout creation of our model. Researched various programming books from the library to gain an understanding of code, since this is a weakness of mine. Helped with brainstorming of issues that arise with code and offered solutions. Assisted in writing the report as well as finalizing organization format for ease of readability.

7.4 Tiangeng Liu Contribution

I was responsible for the code-related tasks, including data preprocessing, rule-based pseudo-label generation, active learning framework implementation, and model training and optimization. I implemented the NER training process using spaCy and dynamically adjusted the sampling strategy (random sampling and confidence-based sampling) to improve model

performance. Additionally, I fine-tuned and optimized training parameters (such as epochs and dropout) to ensure stable learning on the MU16 Twitter dataset. Finally, I completed the model training, evaluation, and generated the experimental results.

7.5 Meeting Minutes

Our group had 5 meetings total during this duration of this project to discuss individual tasks, our written report and code, random samples, results, and evaluation metrics.

Meeting 1: This initial meeting was fairly short and consisted of making sure we all had an understanding of the assignment. We also discussed our individual strengths and assigned tasks accordingly.

Meeting 2: The second meeting was when we began writing our introduction and the various libraries, packages, and functions that we thought would be beneficial in the creation of our model.

Meeting 3: The third meeting we began work on manually annotating random samples used in our model.

Meeting 4: The fourth meeting was a continuation of manual annotations as there was a substantial quantity of random samples.

Meeting 5: The fifth meeting was a finalization of annotations. We also ran our model to return the results which was followed by an in depth discussion of evaluation metrics and what the results show.

8. References

Matthew Honnibal - SpaCy 101: Everything you need to know

https://spacy.io/usage/spacy-101

Pijush Pathak, Linson Thomas Verghese, and Dr. G. Divya - "Sentiment Analysis Text Extraction from Tweets with Spacy NER”

https://ijsret.com/wp-content/uploads/2024/05/IJSRET_V10_issue3_130.pdf

Maxar Blog - “Named Entity Recognition for Twitter”

https://blog.maxar.com/earth-intelligence/2017/named-entity-recognition-for-twitter