B9DA109 Machine Learning and Pattern Recognition:

CA_ONE

Named Entity Recognition Using spaCy

January 2025

Submitted by:
Anish Rao: 20066423

Lecturer: Devesh Jawla

Index

1. Introduction

Named Entity Recognition (NER) is an important part of Natural Language Processing (NLP) which helps in identifying and determining specific pieces of information known as entities inside of a text or abstract into set categories that we can define, such as persons, locations (GPE), diseases, organizations, and more.

In the biomedical domain, NER plays an important role for getting key insights from research papers and medical documents, where accurate identification of entities (e.g., drug names, medical conditions) can help in improving research, drug discovery, and better making better medical decisions.

In this project, we trained a custom NER system using spaCy to recognize chemical and disease entities in biomedical abstracts. SpaCy is a well-known open-source NLP library that has inbuilt tools for building or training models from scratch. It can also be used to build our own model upon pretrained models. We use the BioCreative-V CDR corpus is used as the dataset for building the model. It contains abstracts which are manually annotated, making it perfect for testing and validating our model.

Deep learning models such as BioBERT have successfully worked on achieving impressive results in biomedical NER, but they still do require significant resources for computation. Therefore, we have chosen to train a blank model from scratch so that it explores a hybrid approach that combines Active Learning as well as rule-based-matching to improve the efficiency and performance of the model. This decision has helped provide us with a deeper knowledge of the learning processes, thereby allowing us to experiment with rule-based matching, while offering insights into the challenges and benefits of an active learning framework.

The dataset consists of three PubTator-formatted files:

- **Training Set:** CDR_TrainingSet.PubTator.txt – 500 biomedical abstracts
- **Development Set:** CDR_DevelopmentSet.PubTator.txt – 500 biomedical abstracts
- **Test Set:** CDR_TestSet.PubTator.txt – 500 biomedical abstracts

Each document contains two components, a title and an abstract, along with the chemical and disease entities that are manually annotated, including character offsets in the text. The datasets used for training and development (merged into 1,000 samples) are being used for active learning and model training, while the test dataset with gold-standard annotations is reserved for final the evaluation.

2. Literature Review

Named Entity Recognition (NER) is an important task in Natural Language Processing (NLP), more specifically in the biomedical field, where a huge amount of unstructured text contains critically important information about topics such as diseases, chemicals, and treatments. NER has improved and evolved massively over time. Earlier systems used to rely on approaches like rule-based and dictionary methods which further relied on medical terminologies that are predefined. While this may be effective for structured data with high precision, they often struggle with newer terms and require constant updates. Currently with advancements in sectors such as machine learning and deep learning, models that use architectures such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks often see improved NER by capturing dependencies in text that are contextual. The introduction of transformer models like BERT, BioBERT are capable of now capturing complex patterns, achieving results that are considered state-of-the-art especially in NER tasks. However, these approaches often require large, annotated datasets.

Active Learning (AL): has recently come up as a promising strategy which can be used to address limitations in data. AL techniques allow the model to ask a human annotator for examples that are most informative, which helps reducing the labeling effort while maintaining competitive performance.

SpaCy and Annotation Tools: For this project we use the spaCy library. It is a powerful NLP framework that provides fast, efficient and scalable NER pipelines. While spaCy doesn't have any support for direct active learning, but it provides us with the tools to train a model incrementally by implementing our own active learning loop to improve over multiple iterations.

Our approach combines both rule-based matching and active learning. We initially generate weak labels to get some train data, which is the further refined by focusing on the uncertain cases using active learning. While pretrained models typically offer higher performance, we specifically chose to train from scratch to better understand how an NER model learns in a low-resource setting.

Challenges and Future Directions: Data quantity and quality are critical for NER performance. Key challenges in biomedical NER include ambiguous terminology, data scarcity, and complex sentence structures. Future research will focus on semi-supervised learning, knowledge integration, and multimodal approaches to enhance entity extraction for biomedical applications.

3. Methodology

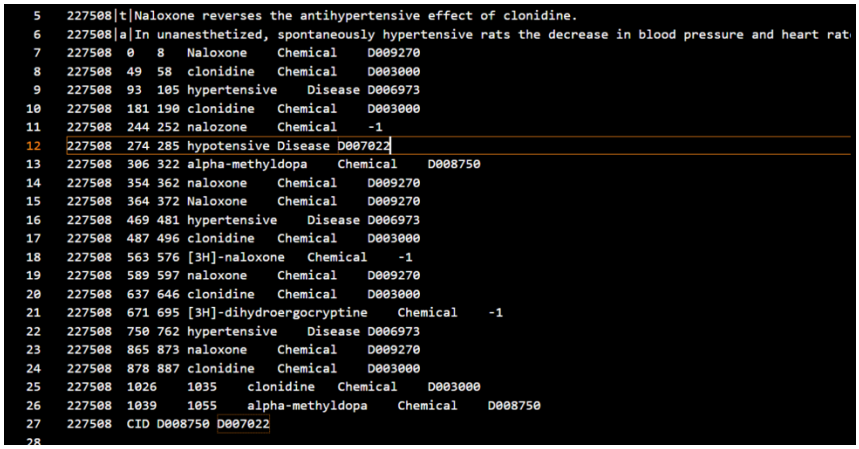
In our project, we develop a custom NER system for biomedical text using spaCy. Our goal was to improve our model's accuracy and make it more efficient in identifying chemicals and diseases. We achieved this by combining rule-based matching and active learning.

Our approach included:

- Data Preparation – Parsing and structuring the dataset for training.
- Rule-Based Matching – Using predefined rules to generate weak labels for entities.
- Active Learning – Iteratively selecting the most uncertain examples for annotation.
- Model Training – Training a blank spaCy model with incremental updates over multiple iterations.

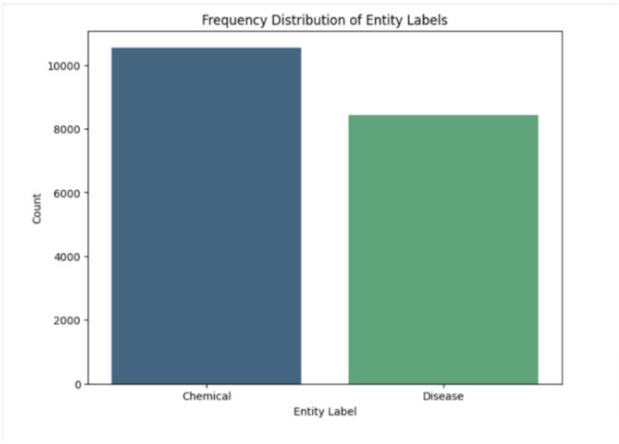
3.1 Data Preparation

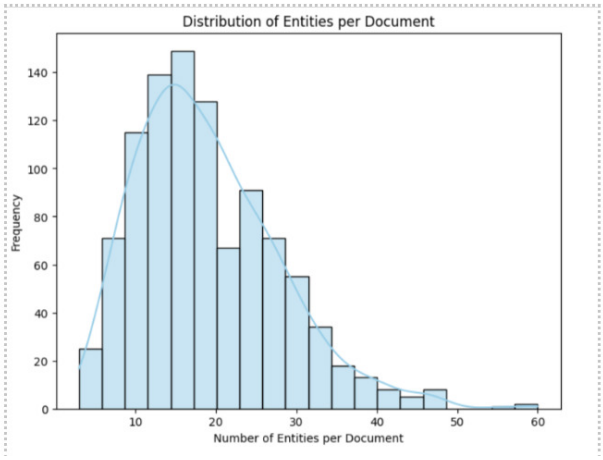
- **Dataset Structure:**
  We used the *BioCreative-V CDR Corpus* dataset for our project. The sets are formatted in PubTator style; a training set, a development set, and a test set, each containing 500 samples.
  Each file contains biomedical articles with structured annotations:
  - A title and an abstract.
  - Manually annotated chemical and disease entities, with exact character offsets.



Sample Data

- **Dataset Statistics and Visualizations:**
  - The combined training set (Train + Dev sets) has a total of 1000 documents, with an average of 18.98 annotations per document.
  - Total entity counts:
    - Chemical Entities: 10,550
    - Disease Entities: 8,426
  - Most of the documents have 10-30 entities, with some documents with up to 50+ entities.





- **Parsing Process:**
  We implemented a custom *parse_data* function which reads the file contents and goes through each line to identify titles and abstracts. These are combined into a single text field to ensure the complete document is available for processing.
  For the training and development sets, annotations are ignored (to simulate unlabeled data). We later generate the labels via rule-based matching and active learning. For the testing set, the function extracts the start and end positions of entities along with their respective labels, ensuring that the data is in the required format for evaluating.

3.2 Rule-Based Matching

We used spaCy's Matcher for creating the initial training labels, which detects patterns using regex rules. For example, chemical names often end with suffixes like "-ine" or "-ol," while many diseases would end with "-itis" or "-osis." These rule-based matches provide us with the initial training data. While this method achieves high precision, its recall is very low which is improved by the active learning.

3.3 Active Learning Loop

Active learning helps in reducing the manual annotation effort by focusing only on the most uncertain examples instead of labeling everything randomly.

- **Initial Labeling:**
  We start of by labeling about 70% of the data using our rule-based matcher. This creates “weakly labeled” dataset for initial training
- **Uncertainty Sampling:**
  The model then checks from the remaining unlabeled data and assign them some uncertainty score. The samples that have the highest score are picked for further labeling.
- **Simulated Human Annotation:**
  In a real-world setup, a human expert would label these uncertain samples manually. For our project, we simulate this step by making use of our rule-based matcher as we already had some accurate rules from examining the datasets.
- **Incremental Model Update:**
  The newly annotated samples are then added to the training set, and the model is trained again over this new set. This process is repeated until we get some good results.

For our project we ran the active learning loop for seven iterations, selecting twenty-five uncertain samples to label/annotate per iteration. By doing this we could ensure model learns slowly and prevent any overfitting

3.4 Model Training & Optimization

Instead of using a pretrained NER model, we started with a blank spaCy model *(spacy.blank("en"))*. This would make sure that our model would learn everything only from our dataset, without any extra noise. For training we used mini batching with:

- Batch size - 16
- Dropout rate - 0.2 (to prevent/reduce any overfitting)
- Epochs - 10

We repeat this process over seven iterations of the active learning loop to update the model gradually.

3.5 Evaluation

To assess how well our model performs, we test It over a separate test (500 documents). These documents contains gold-standard entity annotations, which means they have been manually labeled. We then compare our models result to these annotations

We used three evaluation metrics - Precision, Recall, and F1-score, which are commonly used for NER tasks. These are then calculated using the below components of classification evaluation:

- True Positives (TP) - Entities that the model predicts correctly.
- False Positives (FP) - Entities predicted by the model, but they don't actually exist (incorrect predictions).
- False Negatives (FN) - Entities present in the ground truth but missed by the model.

The three metrics are calculated as follows:

- Precision - Measures how many of the entities predicted by the model were actually correct.
- Recall - Measures how many true entities were successfully identified by the model.
- F1-Score - The harmonic mean of precision and recall, measuring overall performance.

		Predicted	
		0	1
Actual	0	TN	FP
	1	FN	TP

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

4. Results

The training process for our model started with a baseline model using only rule-based annotations. Since these annotations were initially generated using patterns that are predefined, the initial model worked well with recognizing common entity types but was struggling with cases that seemed less familiar or were more complex. This then led to a high precision but very low recall model, which basically meant that it could confidently recognize only a small subset of entities but often missed many others.

In order to improve this, we had to apply active learning over seven different iterations, in which the model was continuously improving itself by focusing on the examples that it was most uncertain about. This process helped our model to generalize beyond the fixed rule-based patterns and allowed for better understanding of entity variations in a variety of contexts.

Finally, our last performance metrics was calculated using *precision_recall_fscore_support* from scikit-learn:

Metric	Score (%)
Precision	58.2%
Recall	35.8%
F1-Score	44.3%

```
[ ]  # Prepare test data.
     TEST_DATA = [{"text": doc["text"], "entities": doc["entities"]} for doc in test]
     evaluate_model(nlp, TEST_DATA)
```

```
=== Evaluation on Test Set ===
Precision: 0.5819, Recall: 0.3580, F1-Score: 0.4433
(0.5819386909693455, 0.35803853603833213, 0.4433223933350164)
```

Our model was successfully exposed to cases that were more complex, slowly improved its ability to generalize. Although the final metrics are average when compared to the pretrained models that are considered state-of-art, they validate our approach of combining rule-based weak labeling with active learning. A deeper analysis of these results and potential improvements is discussed in the Conclusion section.

5. Conclusion

With this project we successfully developed a custom Named Entity Recognition (NER) system that can be used for biomedical texts using a blank spaCy model, rule-based matching, and active learning. Training from scratch helped provide more in-depth insights into our model's behavior, highlighting the trade-offs that happen between precision and recall in a low-resource setting.

The active learning approach proved to be effective, allowing the model to improve slowly in recall while not losing much precision, making it a good substitute to the traditional way of manual annotation.

Before and After Active Learning: Performance Comparison

Model Version	Precision (%)	Recall (%)	F1-Score (%)
Baseline Model	63%	18%	28%
Final Model	58%	36%	44%

- **Precision** remained stable (~58%) due to the rule-based annotations, confirming appropriate entity detection.
- **Recall** doubled (~36%), showing the model improved from active learning and uncertain sample selection.
- The **F1-score** (~44%) showed decent improvement

Challenges & Limitations:

Despite the improvements, some challenges still remain:

- Recall still needs a lot of improvement – While active learning helped, the model still had issues with rare and context-dependent names.
- Limited size of dataset – Since we had only 1,000 training/development samples, our model did not have much to learn from.
- Strict entity character matching – The model had to predict an entity with exact character length, or it would be considered incorrect.

Although the NER model which we built does not outperform the models that are pretrained, such as BioBERT, it can show a scalable, more economic method for training entity recognition models from scratch with less manual annotation required. The combination of rule-based weak labeling and active learning provides a favorable framework that can help in improving entity recognition in low-resource environments.

Future Work:
To further improve our models' performance future research could discover areas such as:

- Different approaches to uncertainty sampling, for example - margin-based selection which could improve active learning.
- Adding transformer-based models (e.g., BioBERT, SciSpaCy) to improve and give more context-based understanding.
- Getting more dataset to include a larger variety of biomedical samples for better generalization and covering more examples.

6. Code

6.1 Dataset & Colab Notebook

For this project we used the **BioCreative-V CDR Corpus**. The dataset consists of three text files in PubTator format, each containing a title, abstract, and manually annotated chemical and disease entities.

Dataset Link: BioCreative-V CDR Corpus
Google Colab Notebook for Code Execution: Project Implementation

Dataset Breakdown:

File Name	Purpose	Number of Documents
CDR_TrainingSet.PubTator.txt	Training set	500
CDR_DevelopmentSet.PubTator.txt	Development set	500
CDR_TestSet.PubTator.txt	Test set	500

6.2 Imports and Model Initialization

We import the required libraries and initialize a blank spaCy model to use.

```
import spacy
from spacy.matcher import Matcher
from spacy.training import Example
import random
import warnings
import spacy.util
from sklearn.metrics import precision_recall_fscore_support
import matplotlib.pyplot as plt
import numpy as np

# load blank spacy model
nlp = spacy.blank("en")
```

The blank model makes sure that the model learns completely from our dataset, rather than relying on any other external knowledge.

6.3 Data Parsing

As the dataset is in a PubTator format, we needed to extract, clean and structure the set which is accepted by spaCy.

```
def parse_data(file_path, remove_gold=False):
    """
    Parses a PubTator file to extract the combined text (title + abstract)
    Optionally, the entity annotations.

    Args:
        file_path (str): Path to the PubTator file.
        remove_gold (bool): If True, ignore entity annotations
                            If False, keep entity annotations

    Returns:
        list: A list of dictionaries, each containing:
              -> "text": The combined document text.
              -> "entities": A list of tuples (start, end, label)
                            if remove_gold is False, otherwise an empty list.
    """
    with open(file_path, 'r', encoding='utf-8') as f:
        data = f.read()

    docs = []
    current = {"text": "", "entities": []}

    for line in data.split("\n"):
        if line.startswith("#####"):
            continue
        if "|t|" in line:
            # Title
            current["text"] = line.split("|t|")[-1].strip()
        elif "|a|" in line:
            # Append Abstract
            current["text"] += " " + line.split("|a|")[-1].strip()
        elif "\t" in line and not remove_gold:
            # Parse annotations only for Test set.
            parts = line.split("\t")
            if len(parts) >= 5:
                try:
                    start, end, label = int(parts[1]), int(parts[2]), parts[4]
                except ValueError:
                    continue
                current["entities"].append((start, end, label))
        elif line.strip() == "" and current["text"]:
            # End of a document; add it to our list.
            docs.append(current)
            current = {"text": "", "entities": []}

    if current["text"]:
        docs.append(current)
    return docs

train_path = "CDR_TrainingSet.PubTator.txt"
dev_path   = "CDR_DevelopmentSet.PubTator.txt"
test_path  = "CDR_TestSet.PubTator.txt"

# Parse the files:
train = parse_data(train_path, remove_gold=True)
dev = parse_data(dev_path, remove_gold=True)
test = parse_data(test_path, remove_gold=False)

train_dev = train + dev
```

6.4 Rule-Based Matching

We came up with a set of regex rules with spaCy's Matcher to create weak chemical and disease labels.

```
matcher = Matcher(nlp.vocab)

chem_patterns = [
    [{"LOWER": {"REGEX": "^[a-z0-9-]+(ol|ine|ide|one|ate|amine|azole|vir|dopa|mab|statin|pril|sartan|rin)$"}}],
    [{"LOWER": {"REGEX": "^(?:cyclo|benz|ethyl|methyl|hydro)[a-z0-9-]*$"}}],
    [{"LOWER": {"IN": ["insulin", "penicillin"]}}],
    [...]
    [...]
    [...]
]

disease_patterns = [
    [{"LOWER": {"REGEX": "^[a-z]+(itis|osis|emia|pathy|oma|algia|penia|dystrophy|sclerosis|necrosis|lepsy)$"}}],
    [{"LOWER": {"IN": [
        "hypertensive", "hypotensive", "cancer", "tumor", "tumour",
        "diabetes", "malaria", "syndrome", "delirium", "migraine"
    ]}}],
    [{"TEXT": {"REGEX": ".*opathy$"}}],
    [...],
    [...],
    [{"TEXT": {"REGEX": "^cardiac$"}}, {"TEXT": {"REGEX": "^(arrest|asystole)$"}}],
    [{"TEXT": {"REGEX": "^heart$"}}, {"TEXT": {"REGEX": "^attack$"}}],
    [{"TEXT": {"REGEX": "^myocardial$"}}, {"TEXT": {"REGEX": "^infarction$"}}],
    [...],
]
```

```
matcher.add("DISEASE", disease_patterns)
matcher.add("CHEMICAL", chem_patterns)

def annotate_docs_with_rules(docs):
    """
    Applies our rule-based matcher to each document to generate weak labels

    Args:
        docs (list): List of document dictionaries

    Returns:
        list: List of document dictionaries with an added key [auto_entities]
              with a list of weak entity annotations as tuples (start, end, label).
    """
    annotated_docs = []
    for doc in docs:
        text = doc["text"]
        spacy_doc = nlp(text)
        matches = matcher(spacy_doc)
        auto_entities = []
        for match_id, start, end in matches:
            span = spacy_doc[start:end]
            label = nlp.vocab.strings[match_id].title()
            auto_entities.append((span.start_char, span.end_char, label))
        new_doc = doc.copy()
        new_doc["auto_entities"] = auto_entities
        annotated_docs.append(new_doc)
    return annotated_docs

# We use a subset of the total data
seed_size = int(0.7 * len(train_dev))
seed_docs = train_dev[:seed_size]

# Apply the rule-based matcher
annotated_seed_docs = annotate_docs_with_rules(seed_docs)
```

6.5 Model Training

In the next step we train our model over few iterations, updating in batches.

We set the dropout rate to 0.2 to prevent overfitting, with 10 training epochs/iterations.

```
def train_ner_model(model, data, n_iter):
    """
    Trains a blank NER model using spaCy with the given data.

    Args:
        model: The blank spaCy model.
        data: List of training examples (dictionaries with "text" and "auto_entities").
        n_iter: Number of training iterations/Epochs.

    Returns:
        None
    """
    # Add NER component if not present.
    if "ner" not in model.pipe_names:
        ner = model.add_pipe("ner", last=True)
    else:
        ner = model.get_pipe("ner")

    # Add weak labels from the training data.
    for doc_data in data:
        ents = doc_data.get("auto_entities", [])
        for ent in ents:
            label = ent[2]
            try:
                ner.add_label(label)
            except Exception:
                pass
```

```
    # Prepare training examples.
    training_examples = []
    for doc_data in data:
        text = doc_data["text"]
        raw_ents = doc_data.get("auto_entities", [])
        doc = model.make_doc(text)
        spans = []
        # Convert entities to Span objects.
        for ent in raw_ents:
            span = doc.char_span(ent[0], ent[1], label=ent[2])
            if span is not None:
                spans.append(span)
        #remove overlapping spans.
        filtered_spans = spacy.util.filter_spans(spans)
        # Convert filtered spans back to the (start, end, label) format.
        annotations = {"entities": [(span.start_char, span.end_char, span.label_)
        for span in filtered_spans]}
        # Only add example if there's at least one entity.
        if annotations["entities"]:
            example = Example.from_dict(doc, annotations)
            training_examples.append(example)

    # Initialize optimizer.
    optimizer = model.begin_training()
    for itn in range(n_iter):
        random.shuffle(training_examples)
        losses = {}
        batches = spacy.util.minibatch(training_examples, size=16)
        for batch in batches:
            model.update(batch, drop=0.2, sgd=optimizer, losses=losses)
        print(f"Iteration {itn+1}: Loss = {losses.get('ner', 0):.4f}")
    print("\n==Model Training Completed==")
```

6.6 Active Learning Loop

The loop consists of:

- **Uncertainty Sampling:**
  The model selects the samples which it is least confident about .

```python
def uncertainty_sampling(model, docs, n_samples, length_constant=200):
    """
    Selects n_samples documents based on model uncertainty.

    Args:
        model: The spaCy model.
        docs: List of document dictionaries (each with a 'text' key).
        n_samples: Number of documents to select.
        length_constant: Constant to scale uncertainty based on document length.

    Returns:
        List of document texts with the highest uncertainty.
    """
    scores = []
    for doc in docs:
        text = doc["text"]
        spacy_doc = model(text)
        confidences = [ent._.confidence for ent in spacy_doc.ents
                       if hasattr(ent._, "confidence")]
        if confidences:
            uncertainty = 1 - max(confidences)
        else:
            uncertainty = len(text) / (len(text) + length_constant)
        scores.append(uncertainty)

    indices = np.argsort(scores)[-n_samples:]
    return [docs[i]["text"] for i in indices]
```

- **Annotation Simulation:**
  The uncertain samples are then labeled using our rule-based matcher

```python
def manual_annotation(text):
    """
    Simulates manual annotation by reapplying the rule-based matcher.

    Args:
        text: The input text.

    Returns:
        list: List of dictionaries with "start", "end", "label".
    """
    doc = nlp(text)
    matches = matcher(doc)
    ann = []
    for match_id, start, end in matches:
        span = doc[start:end]
        label = nlp.vocab.strings[match_id].title()
        ann.append({"start": span.start_char, "end": span.end_char, "label": label})
    return ann
```

- **Model Update:**

  Finally, we retrain our model with new samples. This loop then runs for a set number of times.

```
def active_learning_loop(model, labeled_data, full_data, iterations=3, n_samples=5):
    """
    Simple active learning loop to iteratively update the labeled dataset
    and retrain the model.

    Args:
        model: The spaCy model.
        labeled_data: The initial labeled set
        full_data: The complete set of documents
        optimizer: The optimizer.
        iterations: Number of active learning iterations.
        n_samples: Number of uncertain samples to select per iteration.

    Returns:
        None
    """
    for i in range(iterations):
        print(f"\n=== Active Learning Iteration {i+1} ===")
        # Exclude documents already in labeled_data.
        already_labeled = {doc["text"] for doc in labeled_data}
        candidates = [doc for doc in full_data if doc["text"] not in already_labeled]

        # Use uncertainty sampling on remaining candidates.
        # selected_texts = uncertainty_sampling(model, candidates, n_samples)
        selected_texts = uncertainty_sampling(model, candidates, n_samples)
        print("Selected uncertain samples for annotation:")
        for text in selected_texts:
            print("  -", text[:100] + "...")
```

```
        # manual annotation on selected samples.
        new_annotations = []
        for doc in candidates:
            if doc["text"] in selected_texts:
                anns = manual_annotation(doc["text"])
                new_annotations.append({
                    "text": doc["text"],
                    "entities": [(ann["start"], ann["end"], ann["label"]) for ann in anns]
                })

        # Add the newly annotated examples to the labeled set.
        labeled_data.extend(new_annotations)
        print(f"New annotations added. Total labeled data size: {len(labeled_data)}")

        # Retrain the model on the updated labeled set.
        train_ner_model(model, labeled_data, n_iter=10)
        model.to_disk(f"model_iter_{i+1}")
        print(f"\nFinished Active Learning Iteration {i+1}")

    print("\nActive Learning Loop Completed")
nlp.initialize()
active_learning_loop(nlp, annotated_seed_docs, train_dev, iterations=7, n_samples=25)
```

6.7 Evaluation

We use scikit-learn's precision_recall_fscore_support to check our models' performance.

```python
def evaluate_model(model, test_data):
    """
    Evaluates the trained NER model on test data using micro-averaged metrics.

    Args:
        model: The trained spaCy model.
        test_data: List of test document dictionaries with "text" and "entities".

    Returns:
        tuple: (Precision, Recall, F1-score)
    """
    y_true, y_pred = [], []
    for doc_data in test_data:
        text = doc_data["text"]
        gold = set(doc_data["entities"])
        doc = model(text)
        pred = set([(ent.start_char, ent.end_char, ent.label_) for ent in doc.ents])

        all_labels = set([e[2] for e in gold]).union(set([e[2] for e in pred]))
        for label in all_labels:
            tp = len([e for e in gold if e in pred and e[2] == label])
            fp = len([e for e in pred if e not in gold and e[2] == label])
            fn = len([e for e in gold if e not in pred and e[2] == label])
            y_true.extend([label] * tp)
            y_pred.extend([label] * tp)
            y_true.extend([label] * fn)
            y_pred.extend(["O"] * fn)
            y_true.extend(["O"] * fp)
            y_pred.extend([label] * fp)

    precision, recall, f1, _ = precision_recall_fscore_support(
        y_true, y_pred, average="micro", labels=list(set(y_true) - {"O"})
    )
    print(f"\n=== Evaluation on Test Set ===")
    print(f"Precision: {precision:.4f}, Recall: {recall:.4f}, F1-Score: {f1:.4f}")
    return precision, recall, f1

# Prepare test data.
TEST_DATA = [{"text": doc["text"], "entities": doc["entities"]} for doc in test]
evaluate_model(nlp, TEST_DATA)
```

The evaluation output:

```
=== Evaluation on Test Set ===
Precision: 0.5819, Recall: 0.3580, F1-Score: 0.4433
(0.5819386909693455, 0.35803853603833213, 0.4433223933350164)
```

Appendix: Meeting Notes & Group Member Contributions

Group Meeting Notes

Meeting 1 (Week 1) –

- **Goal:** Understand the dataset and come up with the overall plan.
- **Discussion:**
    - Went through the PubTator format and discussed best way to parse and merged train and dev sets.
    - We agreed upon the project structure and plan

Meeting 2 (Week 1) –

- **Goal:** Parse the data and decide rules for weak labeling.
- **Result:** Came up with a collection of rules/patterns after going through dataset.

Meeting 3 (Week 2) –

- **Goal:** Setup the active learning loop and model training function.
- **Result:** We successfully wrote all required functions

Meeting 4 (Week 3) –

- **Goal:** Decide best evaluation metrics and parameters.
- **Result:** After reviewing the results, we experimented with different parameters to improve our model.

Meeting 5 (Week 4) –

- **Goal:** Finalize the code and report.
- **Result:** We finished the report and restructured the code for final submission.

Individual Contributions

- **Anish (Model Development & Implementation Lead):**
  - Worked on the active learning, evaluation metrics and parameter tuning
  - Estimated time: ~30 hours over 4 weeks.
- **Sharath (Research & Documentation Lead):**
  - Completed the research, literature review and restructuring.
  - Estimated time: ~30 hours over 4 weeks.
- **Adithya (Data Preparation & Rule-Based Matching Lead):**
  - Handled data cleanup, parsing and applied rule-based matching.
  - Estimated time: ~25 hours over 4 weeks.