Named Entity recognition with spaCy

Course – B9DA109: Machine Learning and Pattern Recognition

Group Members:
20066423 – Anish Rao
20054683 – Adithya Durgapu
20058721 – Phani Sri Pavansai Sharath Chandra Chavali Venkata

---

Dataset - [BioCreative-V CDR Corpus]

Project Overview: Goal & Approach

Goal:

> Our aim is to build a Named Entity Recognition (NER) system for identifying chemical and disease entities in biomedical text By combining rule-based techniques with active learning, we want to achieve good performance while keeping manual annotation to a minimum.

Approach:

> - Build a custom NLP pipeline using spaCy
> - Use spaCy's rule-based matching to create weak labels for for chemical and disease entities
> - Apply uncertainty sampling to identify uncertain predictions, prioritizing them for manual annotation to improve model accuracy.
> - Train a blank spaCy model iteratively using minibatch training, refining entity recognition over multiple active learning cycles.
> - Evaluate model performance using:
>   - Precision (how many detected entities are correct?)
>   - Recall (how many actual entities were successfully identified?)
>   - F1-score (the harmonic mean of precision and recall, ensuring balanced performance evaluation).

Importing Libraries & Initializing spaCy Model

```
# !pip install -U spacy


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")
```

Loading & Parsing PubTator Datag

```
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
                             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


    # Test prints: print a sample from the parsed datasets.
    print("Combined Train+Dev Sample:")
    if train_dev:
        print(train_dev[0])
    else:
        print("No documents found.")
    print("=" * 40)
    print("Test Set Sample (with gold annotations):")
    if test:
        print(test[0])
    else:
        print("No documents found.")
```

```
Combined Train+Dev Sample:
{'text': 'Naloxone reverses the antihypertensive effect of clonidine. In unane
========================================
Test Set Sample (with gold annotations):
{'text': 'Famotidine-associated delirium. A series of six cases. Famotidine is
```

Rule-Based Matching for Entity Annotation

```
matcher = Matcher(nlp.vocab)

chem_patterns = [
    [{"LOWER": {"REGEX": "^[a-z0-9-]+(ol|ine|ide|one|ate|amine|azole|vir|dopa|mab
    [{"LOWER": {"REGEX": "^(?:cyclo|benz|ethyl|methyl|hydro)[a-z0-9-]*$"}}],
    [{"LOWER": {"IN": ["insulin", "penicillin"]}}],
    [
        {"LOWER": {"REGEX": "^[a-z0-9-]+$"}, "OP": "+"},
        {"LOWER": "acid"}
    ],
    [
        {"LOWER": {"REGEX": "^[a-z0-9-]+$"}},
        {"LOWER": {"REGEX": "^(chloride|sulfate|sulphate|oxide|nitrate)$"}}
    ],
    [
        {"LOWER": {"REGEX": "^[a-z0-9-]+$"}, "OP": "+"},
        {"LOWER": {"REGEX": "^(inhibitor|inhibitors|blocker|blockers|antagonist|a
    ]
]

disease_patterns = [
    [{"LOWER": {"REGEX": "^[a-z]+(itis|osis|emia|pathy|oma|algia|penia|dystrophy|
    [{"LOWER": {"IN": [
        "hypertensive", "hypotensive", "cancer", "tumor", "tumour",
        "diabetes", "malaria", "syndrome", "delirium", "migraine"
    ]}}],
    [{"TEXT": {"REGEX": ".*opathy$"}}],
    [
        {"LOWER": {"REGEX": "^[a-z]+$"}, "OP": "+"},
        {"LOWER": {"REGEX": "^(disease|syndrome|cancer|tumor|tumour)$"}}
    ],
    [
        {"LOWER": {"REGEX": "^[a-z]+$"}},
        {"LOWER": "'s", "OP": "?"},
        {"LOWER": {"REGEX": "^(disease|syndrome)$"}}
    ],
    [{"TEXT": {"REGEX": "^cardiac$"}}, {"TEXT": {"REGEX": "^(arrest|asystole)$"}}
    [{"TEXT": {"REGEX": "^heart$"}}, {"TEXT": {"REGEX": "^attack$"}}],
    [{"TEXT": {"REGEX": "^myocardial$"}}, {"TEXT": {"REGEX": "^infarction$"}}],
    [{"LOWER": {"IN": [
        "dystonia", "dyskinesia", "neuropathy", "fasciculation", "fasciculations"
        "hypertension", "myocardial infarction", "chronic renal failure", "crohn'
        "infection", "fever", "headache", "inflammation", "stroke", "sepsis", "an
    ]}}],
    [
        {"LOWER": {"REGEX": "^(acute|chronic|severe|idiopathic)$"}, "OP": "?"},
        {"LOWER": {"IN": ["renal", "cardiac", "pulmonary", "neuromuscular", "audi
        {"LOWER": {"IN": ["failure", "injury", "deficiency", "dysfunction", "toxi
    ],
    [{"LOWER": {"REGEX": "^(neuro|cardio|hepato|nephro|pneumo|encephal)[a-z]+$"}}
]

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)


# Test print: print sample annotated documents.
print("\nSample Annotated Documents from Seed Set:")
num_samples = min(2, len(annotated_seed_docs))
for i in range(num_samples):
    sample = annotated_seed_docs[i]
    print("=" * 40)
    print(f"Document {i+1}:")
    print("Text:", sample["text"])
    print("Auto Entities:", sample.get("auto_entities", []))
```

```
Sample Annotated Documents from Seed Set:
========================================
Document 1:
Text: Naloxone reverses the antihypertensive effect of clonidine. In unanesth
Auto Entities: [(0, 8, 'Chemical'), (49, 58, 'Chemical'), (93, 105, 'Disease'
========================================
Document 2:
Text: Lidocaine-induced cardiac asystole. Intravenous administration of a sin
Auto Entities: [(0, 9, 'Chemical'), (18, 34, 'Disease'), (90, 99, 'Chemical')
```

NER Model training method

```
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_entit
        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_)
        # 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==")
```

Active Learning with Uncertainty Sampling

```
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.
        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]

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": labe
        return ann

    def active_learning_loop(model, labeled_data, full_data, iterations=3, n_samples=
        """
        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_labe

            # 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
                    })

            # 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")
```

Executing Active Learning Training

```
warnings.filterwarnings('ignore')


nlp.initialize()
active_learning_loop(nlp, annotated_seed_docs, train_dev, iterations=7, n_samples
```

```
    === Active Learning Iteration 1 ===
    Selected uncertain samples for annotation:
      - Association of nitric oxide production and apoptosis in a model of experi
      - Absence of acute cerebral vasoconstriction after cocaine-associated subar
      - Chemotherapy of advanced inoperable non-small cell lung cancer with pacli
      - Reduced cardiotoxicity and preserved antitumor efficacy of liposome-encap
      - Neuroactive steroids protect against pilocarpine- and kainic acid-induced
      - Longitudinal assessment of air conduction audiograms in a phase III clini
      - Steroid structure and pharmacological properties determine the anti-amnes
      - Dexrazoxane protects against myelosuppression from the DNA cleavage-enhan
      - Iatrogenic risks of endometrial carcinoma after treatment for breast canc
      - Effects of acute steroid administration on ventilatory and peripheral mus
      - Delayed-onset heparin-induced thrombocytopenia. BACKGROUND: Heparin-induc
      - Increased expression and apical targeting of renal ENaC subunits in purom
      - A cross-sectional evaluation of the effect of risperidone and selective s
      - Effects of verapamil on atrial fibrillation and its electrophysiological
      - Comparison of aqueous and gellan ophthalmic timolol with placebo on the 2
      - Probing peripheral and central cholinergic system responses. OBJECTIVE: T
      - Prednisolone-induced muscle dysfunction is caused more by atrophy than by
      - Dexamethasone-induced ocular hypertension in perfusion-cultured human eye
      - Cyclophosphamide-induced cystitis in freely-moving conscious rats: behavi
      - Nephrotoxic effects in high-risk patients undergoing angiography. BACKGRO
      - Impact of alcohol exposure after pregnancy recognition on ultrasonographi
      - Results of a comparative, phase III, 12-week, multicenter, prospective, r
      - Orthostatic hypotension occurs following alpha 2-adrenoceptor blockade in
      - 22-oxacalcitriol suppresses secondary hyperparathyroidism without inducin
      - Comparison of developmental toxicology of aspirin (acetylsalicylic acid)
    New annotations added. Total labeled data size: 725
    Iteration 1: Loss = 32064.9459
    Iteration 2: Loss = 6796.3936
    Iteration 3: Loss = 3814.2383
    Iteration 4: Loss = 3046.2015
    Iteration 5: Loss = 2681.5259
    Iteration 6: Loss = 2237.3136
    Iteration 7: Loss = 2133.3297
    Iteration 8: Loss = 2110.4924
    Iteration 9: Loss = 2347.1513
    Iteration 10: Loss = 1996.7340

    ==Model Training Completed==

    Finished Active Learning Iteration 1

    === Active Learning Iteration 2 ===
    Selected uncertain samples for annotation:
```

```
- Effect of lithium maintenance therapy on thyroid and parathyroid function
- Phase I trial of 13-cis-retinoic acid in children with neuroblastoma follo
- Force overflow and levodopa-induced dyskinesias in Parkinson's disease. We
- Hepatotoxicity associated with sulfasalazine in inflammatory arthritis: A
- Preliminary efficacy assessment of intrathecal injection of an American fo
- Effect of glyceryl trinitrate on the sphincter of Oddi spasm evoked by pro
- Recent preclinical and clinical studies with the thymidylate synthase inhi
- Does supplemental vitamin C increase cardiovascular disease risk in women
- Acute blood pressure elevations with caffeine in men with borderline syste
- Factors associated with nephrotoxicity and clinical outcome in patients re
- Enhanced stimulus-induced neurotransmitter overflow in epinephrine-induced
- The attenuating effect of carteolol hydrochloride, a beta-adrenoceptor ant
```

Evaluating the NER Model Performance

```
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.ent

        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
```

Execution

```
# 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)
```