Overview

We ran five experiments. See the appendix and the accompanying Jupyter notebook for reproducibility.

1. Develop using GPT3.5 zeroshot

We developed our first model using `gpt3-5-zeroshot.jsonl`, loading it with `db-in` and named it `hmwk_1_zeroshot`. We opted to use `--training patience=800` to reduce the early stopping criteria to improve the time to train. For this initial model, we did not have a dedicated hold out dataset, and used Prodigy's automatic partitioning of 20% the `hmwk_1_zeroshot` as the evaluation dataset. The model attained **32%** F1 score.

2. Create 200 unlabeled for evaluation dataset

For the 2nd experiment, we need dedicated hold-out evaluation dataset. We used a previously annotated `.jsonl` file and loaded it with `db-in` and naming it `hmwk_1_eval`. We then retrain on `hmwk-1-zeroshot` and eval on `hmwk-1-eval`. The model attained **36%** F1 score. We also ran a train-curve, which provided evidence that more annotated data would likely improve the model's performance.

3. Train with Workshop Data with model in loop

For the next two experiments, we'll explore: quantity vs. quality. In experiment 3, we'll use all of the workshop data without correcting it (i.e., quantity). We loaded the data with `db-in`, named it `hmwk-1-workshop`, and merged it with the GPT3.5 Zero Shot as `hmwk_1_train_exp3`. We kept `hmwk_1_eval` as the eval dataset. The model attained **52%** F1 score; therefore adding the workshop dataset adds **+16%** F1 score.

4. Run `model-as-annotator` on Workshop data to correct a sample

In experiment 4, we'll focus on quality, correcting ~200 workshop annotations. We'll use `review` so we can view both the model-predicted and workshop annotations. You could use `ner.manual` (original annotations) or `ner.correct` model predicted alternatively. Merging the corrected annotations with the GPT3.5 Zero Shot, yielded the `hmwk_1_train_exp4`. We'll keep using `hmwk_1_eval` as the evaluation dataset. The model attained **38%** F1 score. This indicates that while the full workshop dataset may have errors, it provides net gain in performance (quantity over quality).

5. Use word vectors (transfer learning) to improve model performance

For experiment 5, we'll explore whether adding word embeddings improves model performance (see Prodigy docs). We used experiment 3 training dataset (best performance) and use `hmwk_1_eval` as the eval dataset. The model attained **59%** F1 score, which is the best performing model/experiment. However, this model is much larger (600MB vs 6MB); therefore, in prod, we may still want to go with model 3.

Appendix

Experiment Performances

#	Training Dataset	Training (n)	Evaluation	Base Model	F1	P	R	Size
1	GPT3.5	400	20% of training	None	0.32	0.44	0.24	6 MB
2	GPT3.5	500	Hold out eval	None	0.36	0.41	0.31	6 MB
3	GPT3.5 + Workshop	1577	Hold out eval	None	0.52	0.56	0.48	6 MB
4	GPT3.5 + Corrected	699	Hold out eval	None	0.38	0.44	0.34	6 MB
5	GPT3.5 + Workshop	1577	Hold out eval	en_core_web_lg	0.59	0.58	0.59	600 MB

Saving Models (wheel)

To export each model, we used `spacy package`, saving it as a Python package to a `models` folder. This saves the model as `wheel` which compressed the model.

To install any of the models, you can the use `pip install` like

```
$ python -m pip install models/en_ner_reddit_cooking-5.0.0.tar.gz
Processing ./models/en_ner_reddit_cooking-5.0.0.tar.gz
  Preparing metadata (setup.py) ... done
...
Successfully built en_ner_reddit_cooking
Installing collected packages: en-ner-reddit-cooking
Successfully installed en-ner-reddit-cooking-5.0.0
```

You can then run the model like:

```
import spacy
nlp = spacy.load("en_ner_reddit_cooking")
doc = nlp("Make a dressing with mayo, plain yogurt, curry powder, mustard powder, mustard,
doc.ents
# (mayo, plain yogurt, curry powder, mustard powder, mustard, ground coriander)
```