Information Extraction Phase 3 Write-Up

Jacob Johnson

1 Information Extraction Task:

My system performs a relation extraction task to recognize Drug-Drug interactions in literature from the biomedical domain of Pharmacovigilance (Herrera-Zazo et al). My system takes as input raw documents (either within the command line or from a file). Optional inputs include the set of gold entities within the document (If gold entities are not provided, then my system uses a trained spaCy Entity Extraction model) and the set of features to use in prediction (or, just the best-ranked set of features from an exhaustive ablation test on the devset. My system then outputs the set of extracted relation tuples, along with which type of relation it is labeled as. In accordance with the corpus labeling schemes of Herrera-Zazo et al, there are four labels for relations: "mechanism", which describes how drugs interact at a biological level, "effect", which describes what the interaction does to the patient, "advise", which is just a written encouragement not to allow the drugs the chance to interact, or "int", short for interaction, which simply describes that an interaction is observed between the drugs, without further detail.

2 Resources List:

All external python packages were already available on the CADE lab machines and can be easily installed using Python's pip package installer, but links to their documentation websites are included here.

1. sklearn (`https://scikit-learn.org`): the LogisticRegression class was the backbone of the machine learning for my system. I also used the associated DictVectorizer class to handle non-numeric features (dependency labels in the filter classifier).

2. spaCy (`https://spacy.io`): en_core_web_lg (spaCy's large web-trained English language model) was used for dependency parsing, and vectorization. These vectors, in turn, came from GloVe Common Crawl. I also trained its NER capabilities to recognize Drug Entities in the case where gold Entities are not input with the document, where it achieved an F-score of .797.

3. numPy (`https://numpy.org`): the ndarray class was used to hold vectors produced by spaCy.

4. pandas (`https://pandas.pydata.org/`): the DataFrame class was also used to hold vectors produced by spaCy and given to sklearn models as training and input data.

5. From the python standard library, the pickle package was used to save and load machine learning models, pre-calculated vectors, and forms of the training, dev, and test data that were preloaded from their XML format, the xml package was used to parse the corpus annotations, and the re package was used for regular expressions.

6. The corpus was no longer available at the link provided in Herrera-Zazo et al, but it can be downloaded from `https://github.com/isegura/DDICorpus`

7. I included a code snippet I found on stack overflow to display a progress bar while extracting vectors from documents, so I could monitor its progress.

3 Technical Description:

My system relies on a vector-semantics evaluation of the syntactic context (dependency path) that connects pairs of entity mentions within a sentence to each other and to the sentence root. Words along these dependency paths were recorded, vectorized, weighted, and aggregated. With prompting from the Phase 1 suggestions, I included several features derived from the dependency labels along the path, as well as several other features, of which selected ones were concatenated together, resulting in a vector of up to 3305 dimensions:

1. path1: This feature, along with peak and path2, were derived from the original context path feature used in my Phase 2 system. Instead of fully summing the vectors for all the words in the full dependency path into one aggregated vector, I aggregated the vectors for one side of the dependency path as its own feature. Path items closer to the peak were weighted higher, such that if there were 3 words along the dependency path from an entity to the peak, the one nearest to the entity mention would have 1/4 of its weight, and the one nearest to the peak would have 3/4 of its weight. Because this was based on spaCy's GloVe vectors, this feature was a 300-dimensional vector.

2. peak: This feature is simply the 300-dimensional vector of the "peak" between the two entity mentions, retrieved from spaCy/GloVe.

3. path2: This feature is the aggregation of the vectors in the path for the second entity mention, and was also a 300-dimensional vector; see path1 for more details.

4. path1len: This feature is the number of words along the dependency path from the first entity mention to the peak, a single value.

5. path2len: Likewise, this feature is the number of words along the dependency path from the second entity mention to the peak, a single value.

6. dep1: This feature and dep2 were motivated by a need to include information about the dependencies between along the path, in addition to the words themselves. I generated an embedding-like representation for each dependency label in spaCy (which are based on the clearNLP guidelines) by training an sklearn LogisticRegression classifier to predict the dependency relation between a word and its dependency head based on the concatenation of the word's vector and the vector of its dependency head. This was trained on all words in the train partition that I used for my phase 2 system (866 documents, all of which ended up in my train-test partition for phase3), then the weights used for each label were retrieved from the classifier and assigned to the label they correspond to. They were 600-dimensional vectors, because they were learned based on the concatenation of two 300-dimensional vectors. Because this is simply used to generate a vector representation for dependency labels that can be aggregated across a dependency path, I do not expect this to result in cross-contamination of the data, however, it would be interesting to follow up on this concern. These embedding-like representations were aggregated in similar fashion to the words along the path: dependency labels closer to the "peak" of the context were weighted more heavily than those closer to the entity mention, such that if there were 4 dependency labels along the path from an entity to the peak, the one nearest to the entity mention

would have 1/4 of its weight and the one nearest to the peak would have its full weight. dep1 is the aggregation for the labels along the dependency path from the first entity to the peak.

7. dep2: This is the aggregation for the labels along the dependency path from the second entity mention to the peak; see dep1 for details.

8. root: This is the 300-dimensional vector for the root of the sentence. The motivation for including this, and other root-based features was the observation that some sentences take the form "ABC and DEF...", in which case the direct dependency path gives very little insight into whether the sentence reports an interaction (DEF - conj - ABC), as that sentence could be completed with something as simple as "...interact.", suggesting an interaction was observed, or something such as "...have not been observed to interact.", suggesting no interaction, or even "...were both tested for interactions with GHI.", suggesting that an interaction between ABC and DEF was never even in question.

9. rootPath: This is the 300-dimensional aggregation of vectors from the words along the dependency path from the peak to the sentence root. Words closer to the sentence root were given higher weight, such that if there were 3 words along the path, the one nearest to the peak would have 1/4 its weight and the one nearest to the root would have 3/4 its weight.

10. rootDep: This is the 600-dimensional aggregation of dependency label vectors along the dependency path from the peak to the sentence root. Words closer to the sentence root were given higher weight, such that if there were 4 dependency labels along the path, the one nearest to the peak would have 1/4 its weight and the one nearest to the root would have its full weight.

11. rootLen: This is the number of words whose vectors were aggregated into the rootPath, a single value.

12. swapped: This is a non-syntactic feature that went along with a tweak to my system that ended up not helping much; see the first two elements of the next list (especially "Swapping") for more details. If swapping was activated, this feature was a single value of 1 if a swap was performed, or 0 if a swap was not performed.

13. surfDist: This is a non-syntactic feature that reports the surface distance between the roots of the two entity mentions, a single value.

Based on the results of my Phase 2 experimentation, a single LogisticRegression multiclassifier was trained on the five labels, "mechanism", "effect", "advise", "int", and "none", using all gold vectors from the training documents, plus as many negative vectors produced from unrelated entities as there were gold vectors (the five cross-validation folds contain 4009, 3449, 3388, 3725, and 3957 gold relations, respectively, which means each fold's classifier was trained on 14800 positive and 14800 negative vectors on average).

Additionally, I attempted three changes to the system beyond concatenating additional features into the Classifier input:

1. Clustering Contexts: Because mentions of the same pairs of entities can occur multiple times within a document, my system now collects all of the pairs' context vectors as it reads through the document, and then, based on every within-sentence co-occurrence of entity mentions in a document, decides whether the entities are related, and assigns a label based on the predictions of the classifier for the contexts. If multiple labels are predicted, it selects the most-predicted label. Only if several labels are tied as the most highly-predicted across the document does it ignore potential labels. Because this is

no longer a multibinary classification system, this approach is not susceptible to the danger pointed out in the review of my Phase 2 system; the number of predictions is determined by the number of co-occurring pairs of mentions of the two drugs, rather than by the number of binary classifiers. It is still possible for labels to be ignored, but it is much less likely, as it requires there to be the same number of predictions made with the same label.

2. Swapping: Because entities could be mentioned in either order, and because I was concerned that the classifier could stumble over comparing the clustered contexts ABC-DEF and DEF-ABC, I attempted to canonize which entity mention was considered the "first" (associated with path1, path1len, and dep1) and which was the second, by determining that the alphabetically-earlier entity would be considered first. Before I got to the exhaustive ablation test (see below), preliminary experiments and reason suggested that this was not helpful, so I deactivated it, but, as noted again in the evaluation section, the code to extract this feature was still present when the code for the ablation test was run.

3. Filter for cross-phrase entity relation rejection: The motivation for this filter is based on the observation that entities mentioned in some configurations relative to each other are unlikely to be related, such as if one is in a direct object and the other is within an adjunct phrase. I set up a script to train a classifier based on the two dependency labels connecting the two entity mention paths to the peak. It retrains on the fly based on the test partition that produced by cross validation. It looks at all pairs of dependency labels that connect a pair of entity paths to their peak (for both related and unrelated entities) and all pairs that occur for unrelated entities but never for related entities are marked "bad", while all pairs that occur for related entities are marked "good" (trying this the other way around - labeling dependency label pairs as "bad" if they ever occurred with unrelated entities - resulted in too few pairs that never occurred with unrelated entities). An evaluation of the dependency filter alone based on the same cross-validation folds yielded Precision of .184 and Recall of .970 for identifying related entity mention pairs, suggesting that, while it does allow a lot of unrelated entity pairs through, it blocks very few related entity pairs, while filtering out 18.6% of all entity mention pairs very efficiently.

4 Phase 3 vs Phase 2

In addition to the changes described in the above technical description of my phase 3 system, I made several minor adjustments to the code, most of which were intended to accomplish one of two objectives:

1. Making slight improvements to the efficiency of the system (mainly in the way it computes vector aggregations).

2. Making the system more modular, so that classifiers can be retrained on they fly, in support of Cross-Validation, which I used for the Phase 3 system (less use of pretrained/pickled models).

I also made adjustments to how strictly I evaluate outputs. See under the Evaluation section for more details.

5 Evaluation:

During the development phase, I performed an overnight exhaustive ablation testing of 12 features ("swapping" was already turned off by this point, so even though experiments were performed that varied only in whether this feature was included, that data is not meaningful). The outputs of all 8191 systems were

recorded and ranked by their overall f-score, and individual features are ranked here by the average ranking of all systems that use that feature: path2, peak, root, dep1, path1, dep2, rootPath, rootDep, path2len, path1len, surfdist, rootLen (see the comments on other/ranks/rank.py for a more detailed description). This is just one training set used on the dev set (not cross-validated), but I think it does, in a limited way, give some insight into the relative information a feature gives the system about whether a pair of entities are related. The path and peak features are all relatively highly ranked, while surface distances and lengths are ranked lower. I find it interesting that the path from the second entity to the peak (path2) was more highly ranked than the path from the first entity (path1). Possibly this has to do with the nature of English syntax, since more syntactically-complex elements of the sentence often occur further to the right (English syntax produces right-heavy trees), but it is also possible that this is simply an artifact of spaCy's parsing model. It would be interesting to follow up with this to see whether this is still observed when other parsing algorithms are used.

Starting with the evaluation of my Phase 2 system, I became suspicious of the results of my system's evaluation. In particular, the F-score jumped noticeably between development and testing. In response, I planned to use Cross-Validation for Phase 3 to combat the possibility that my test-set, though randomly selected, might just be "lucky". I reassigned documents either into a devset partition (169 documents) or a cross-validation partition (1000 documents). For experiments on the devset, I used the old train partition (a 866-document subset of the cross-validation partition) with this devset, while for the experiments reported below I used 5-fold cross validation across the 100-document partition (200 documents per fold). This brings the earlier systems' evaluations down a few points from what was previously reported, but these numbers likely capture the system's effectiveness more meaningfullly.

Additionally, I found that, since some pairs of related entities are mentioned multiple times within a sentence or document ("ABC and DEF...but ABC..."), sometimes one ordering of a Mention Pair will be extracted ("ABC", "DEF"), and sometimes the other ("DEF", "ABC"). To account for this, I created a custom wrapper class for a python tuple to compare extracted mention pairs regardless of order. Really, this should have been done from the beginning; my assumption that a sentence would not mention an entity more than once was a bad call.

To facilitate comparison between all three phases of my system, I modified a copy of my previous two systems to work with my current evaluation script in using this metric for entity mention pair equality and also to use cross-validation.

First, an ablation:

1. from2: uses only path1, peak, and path2, all of which contain only information that was aggregated into the vectors used in Phase2
2. +dep: this uses all features in from2, plus the dependency-related features from the same dependency path, namely dep1, dep2, path1len, and path2len
3. +root: this uses all the features in +dep, plus the root-related features, namely root, rootPath, rootDep, and rootLen
4. all: this uses all the features for the system
5. p+r: this uses only peak and root, which are highly meaningful from a syntactic standpoint

6. dev: this uses the set of features used by the highest-scoring system from the ablation test in development, namely path1len, path1, dep1, peak, path2, path2len, root, rootPath, and rootDep

	mechanism			effect			advise			int			total		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
from2	0.504	0.428	0.463	0.548	0.510	0.528	0.435	0.383	0.407	**0.618**	0.399	**0.485**	0.514	0.450	0.480
+dep	**0.521**	0.429	**0.471**	**0.574**	0.514	0.543	0.439	0.395	0.416	0.512	0.385	0.440	**0.525**	0.454	0.487
+root	0.432	0.337	0.378	0.504	0.465	0.484	0.384	**0.512**	0.439	0.874	0.288	0.433	0.456	0.424	0.440
all	0.414	0.422	0.418	0.567	0.418	0.481	0.420	0.393	0.406	0.800	0.264	0.397	0.480	0.404	0.439
p+r	0.267	0.304	0.284	0.382	0.426	0.403	0.374	0.501	0.428	0.337	**0.451**	0.386	0.343	0.406	0.372
dev	0.504	**0.442**	**0.471**	0.528	**0.595**	**0.560**	**0.495**	**0.512**	**0.503**	0.523	0.431	0.472	0.514	**0.520**	**0.517**

Notice that adding the dependency-label features did improve precision and recall for most categories and overall, but the adding the root and other features worsened performance. For all my human intuition, I couldn't have predicted the set of features suggested by the devset experiments would work as well as it did.

Next, a comparison of Phase 1, 2, and 3 systems:

	mechanism			effect			advise			int			total		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
P1	0.349	0.374	0.361	0.471	0.169	0.249	0.178	0.473	0.259	0.129	0.403	0.195	0.251	0.313	0.279
P2	0.405	**0.671**	**0.505**	0.395	**0.657**	0.494	0.272	**0.545**	0.362	0.296	0.368	0.328	0.363	**0.620**	0.457
P3	**0.504**	0.442	0.471	**0.528**	0.595	**0.560**	**0.495**	0.512	**0.503**	**0.523**	**0.431**	**0.472**	**0.514**	0.520	**0.517**

Though my Phase 2 system often outperforms my phase3 system in terms of Recall, my Phase 3 system's substantial increase in Precision brings it to the top ranking. Meanwhile, my Phase 1 system is obsolete in comparison to the other two systems.

And finally, a comparison between using Gold Entities using spaCy's NER to extract the entities:

	mechanism			effect			advise			int			total		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
spaCy	0.472	0.435	0.453	0.493	0.557	0.523	0.466	0.459	0.463	0.550	0.403	0.465	0.484	0.489	0.486
gold	0.504	0.442	0.471	0.528	0.595	0.560	0.495	0.512	0.503	0.523	0.431	0.472	0.514	0.520	0.517

As expected, it takes a few points off for not having the gold entities, but it shows that spaCy's NER does a decent job at extracting the entities.

6 Remaining work:

Below I highlight a few approaches I could take with my system if I continue to work on it beyond the scope of this class/project:

1. I suspect that, in trying to create aggregatable, embedding-like representations of dependency labels, I needlessly increased dimensionality of the vectors, and inhibited my system's performance (and efficiency). Since the "filter" (see Phase 3 vs Phase 2 section) seems to effectively help in determining

whether a pair of mentioned entities are related, it seems likely that just including the actual dependency label as a feature (and letting sklearn transform it using its own methods) might have been a better approach.

2. It seems like this is starting to bump into problems with the reliability of the syntax parser. For example, I have noticed that in constructions taking the form of "NOUN of ABC and DEF" are consistently parsed as "(NOUN of ABC) and DEF", rather than "NOUN of (ABC and DEF)". It may be that this is more common in the English web-text that this spaCy model was trained on, but it was the incorrect parse for many constructions within the DDI corpus. In order to get better results, it would probably be useful to train a parsing system on a medical/drug corpus or at least one with a more similar syntactic profile.

3. Additionally, there are important cues to drug interactions that are still not captured by my system's representation of an entity-mention pair: consider the example sentence "ABC was tested with DEF for interactions and...". Nothing beyond the word "interactions" would be accessed by my system, since "tested" is both the peak of ABC-DEF, and the root of the sentence. Meanwhile, the actual information is completely left out of this region. A similar scenario occurs for negation words, which can be easily overlooked but could change the meaning of the sentence.

4. Over this project, I have generally been targeting to improve F-score. In a real situation where this were actually going to be used to detect drug-drug interactions, it would probably be better practice to optimize Precision and acquire a large corpus of documents, similar to the web-SLI paradigm. After all, according to the Herrero-Zazo paper this work is based on, there are 10,000-20,000 medical articles published per week on MedLine, one of the sources for the corpus used, and a very precise system with low recall would be more reliable than my current system, for which each extracted interaction only has about a 50% chance of being real.

7 References:

Herrero-Zazo, M., Segura-Bedmar, I., Martínez, P., & Declerck, T. (2013). The DDI corpus: An annotated corpus with pharmacological substances and drug–drug interactions. Journal of biomedical informatics, 46(5), 914-920.