Advanced Human Language Technologies

Second Assignment: DDI Classification

Barcelona, May 3rd 2020

Jake Watson

1 Rule Based Classifier

The rule based classifier attempts to make use of various semantic features provided by the Stanford CoreNLP program – in particular, the dependency tree, and the POS tagger.

This information is provided by running the Analyzer function,

```
def analyze(s):
    current_position = 0
    if s:
        s = s.strip()
        dep_graph, = parser.raw_parse(s)
        for i in range(1, len(dep_graph.nodes)):
            node = dep_graph.nodes[i]
            word = node["word"]

            start = current_position + s[current_position:].find(word)
            end = start + len(word) - 1

            current_position = end

            if start != -1:
                node["start"] = start
                node["end"] = end

        return dep_graph
    else:
        return None
```

This function returns a dependency tree for the input sentence. Each node of the tree contains a tokenized word with start and end offsets, it's Part-Of-Speech tag, its relation to the word directly above it in the tree, and any dependent words on it, along with other things.

For each pair of entities in the sentence, the output of this function is passed to the classifier function, which returns one of four types of drug-interaction between the entities, if any interaction exists. If not, it returns `null.`

The classifier uses a set of rules to determine if an interaction exists, and of what type it is. These rules are discussed below.

1.1 Considered Rules:

Four main categories of rules were considered for this classifier:

1. Word Level rules: These rules operate at the single-word level in the sentence. They essentially search an area within the sentence for known keywords, derived from the training set. The areas can be before, between, or after the pair of entities.
2. Phrase Level rules: These rules operate over the words in the path directly between the entities in the tree. They check keywords between in the path, or the POS tags of the words.
3. Sentence Level rules: These rules operate across the whole dependency tree, linking words to the entities although they may be far separated. These check relations and POS tags between words and entities, or check for properties of the tree such as the relative positioning of the entities to each other, or whether or not the entities have the same closest verb in the tree.

Additionally, the rules were divided into a further three categories. Firstly, a set of rules identifies whether or not an interaction could exist between the entities. If they find that one might exist, a second set of rules attempts to weed out false positives. Finally, a third set of rules attempts to identify the DDI type.

For the first category, three rules were tried. The first was testing if the entities shared the same verb, as it was thought that a verb influencing both entities was likely to be describing an interaction. The second was to check if the path between the entities contained a preposition, such as ‘with’, as a phrase like this was thought to be more likely to be ‘joining’ the entities in some way. The third rule was to check if one of the entities was under the other in the dependency tree, as if one entity was directly under the influence of another then it may be likely that they interact. After testing, only the first rule was found to improve the identification score.

For the second category, weeding out false positives, two rules were tried. The first was to test if the path between the entities contained a negation word, such as “don’t” or “not”. The idea behind this is that if such a word exists in a phrase containing two entities, then it describes the consequences of not joining the two entities. The second rule was to check if there is a negative conjunction, such as “or” in the phrase and between the entities in the sentence, as such a phrase may not be describing the joining of these two entities, but rather joining one or both of them independently with a third entity. After testing, the second rule was found to improve the score marginally, and the first was discarded.

For the third category, a series of keyword searches and whole-sentence properties was used. The relations between each word in the path joining both entities was searched for known relations associated to an interaction type, the words before, between and after the entities were searched, and the shared verb for both entities was searched. After testing it was determined that the whole-sentence keyword searches were not effective, but that searching the relations between entities and searching the shared verb for keywords both increased the score. Summarised code (failed tests removed for clarity of report, retained in source code) for the classifier function is shown below.

```
def check_interaction(id_e1, id_e2, entities, entsToNodes, analysis, clues_before,
clues_between, clues_after):

   ddi_type = False

    graph = get_tree_graph(analysis)

    # ---------------------------------------------------------------------------------------
------
    # WORD MANIPULATION TESTS
    # ---------------------------------------------------------------------------------------
------


    shared_verb_keyword = shared_verb_keyword_search(id_e1, id_e2,   entities, entsToNodes,
analysis,
    graph, clues_before, clues_between, clues_after)


    # ---------------------------------------------------------------------------------------
------
    # PHRASE PROPERTIES TESTS
    # ---------------------------------------------------------------------------------------
------

    same_verb = entities_under_same_verb(id_e1, id_e2, entsToNodes,
    analysis, graph)
    _2under1 = second_entity_under_first(id_e1, id_e2, entsToNodes, graph)
    rel_type = check_dependencies_between(id_e1, id_e2, entities, entsToNodes,
    analysis, graph)

    # ---------------------------------------------------------------------------------------
------
    # PHRASE SEARCH TESTS
    # ---------------------------------------------------------------------------------------
------

    preposition = preposition_in_phrase(id_e1, id_e2, entities, entsToNodes,
    analysis, graph)
    negative_conj = negative_conjunction_between_entities(id_e1, id_e2, entities,
    entsToNodes, analysis, graph)

    # First Step: identify whether or not a potential interaction exists.
    # Properties to check:
    # if the entities have a preposition in their shared phrase

    if preposition:

        # Second Step: weed out false positive.
        # False positives: identified by checking for lack of preposition,
        # or presence of negative conjunction between them

        if negative_conj:
            return 0, 'null'

        # Third step: identify type of interaction.
        # checks the relations between the entities for known
        # relations for each ddi type. Also checks the verb that both entities hare
```

```
        # for keywords, and returns the associated ddi type for that keyword.
        # Prioritises the shared_verb keyword over the rel_type, if both are not null.
        if rel_type == shared_verb_keyword and rel_type:
            ddi_type = rel_type
        elif rel_type and not shared_verb_keyword:
            ddi_type = rel_type
        elif shared_verb_keyword and not rel_type:
            ddi_type = shared_verb_keyword
        elif rel_type and shared_verb_keyword:
            ddi_type = shared_verb_keyword

        if ddi_type:
            return 1, ddi_type
        else:
            return 0, 'null'
    else:
        return 0, 'null'
```

1.2 Devel Evaluator Output

```
Partial Evaluation: only detection of DDI (regadless to the type)
tp	fp	fn	total	prec	recall	F1
147	300	337	484	0.3289	0.3037	0.3158


Detection and Classification of DDI
tp	fp	fn	total	prec	recall	F1
105	342	379	484	0.2349	0.2169	0.2256



________________________________________________________________________

SCORES FOR DDI TYPE
Scores for ddi with type mechanism
tp	fp	fn	total	prec	recall	F1
44	141	157	201	0.2378	0.2189	0.228


Scores for ddi with type effect
tp	fp	fn	total	prec	recall	F1
30	74	132	162	0.2885	0.1852	0.2256


Scores for ddi with type advise
tp	fp	fn	total	prec	recall	F1
31	77	88	119	0.287	0.2605	0.2731


Scores for ddi with type int
tp	fp	fn	total	prec	recall	F1
0	50	2	2	0	0	0


MACRO-AVERAGE MEASURES:
	P	R	F1
	0.2033	0.1661	0.1829
________________________________________________________________________
```

1.3 Test Evaluator Output

```
Partial Evaluation: only detection of DDI (regadless to the type)
tp    fp    fn    total prec  recall      F1
448   2220  531   979   0.1679      0.4576      0.2457


Detection and Classification of DDI
tp    fp    fn    total prec  recall      F1
276   2392  703   979   0.1034      0.2819      0.1514


________________________________________________________________________

SCORES FOR DDI TYPE
Scores for ddi with type mechanism
tp    fp    fn    total prec  recall      F1
74    745   228   302   0.0904      0.245 0.132


Scores for ddi with type effect
tp    fp    fn    total prec  recall      F1
109   928   251   360   0.1051      0.3028      0.156


Scores for ddi with type advise
tp    fp    fn    total prec  recall      F1
56    714   165   221   0.0727      0.2534      0.113


Scores for ddi with type int
tp    fp    fn    total prec  recall      F1
37    5     59    96    0.881 0.3854      0.5362


MACRO-AVERAGE MEASURES:
      P     R     F1
      0.2873      0.2967      0.2919
________________________________________________________________________
```

2 Machine Learning Classifier

For the ML approach, the Analyzer function is the same, and much code was reused for the analysis of sentence structure. The features used are discussed below.

2.1 Considered Features

For the ML features, many features were tried. After much testing, three main categories were used:

1. Sentence word and POS: All the nouns and verbs before, between and after the entities were identified and added to the feature vector.

2. Phrase properties: the preposition presence and entity-1-under-entity-2 rules from the previous task were reused and the output added to the feature vector.
3. The relations, lemmas, and POS tags of verbs, conjunctions, prepositions and adjectives in the path between the entities were appended to the feature vector.
4. The relations, lemmas and POS tags of nouns and verbs in the shared path between the entities and the tree root were appended to the feature vector.

Many features were used and discarded. These are described below:

- Drug Types: the lemmas and types of the entities were used, but only decreased accuracy.
- A binary classifier for DDI interaction existence, and a multiclass identifier for type. This attempted to use similar features for the binary classifier as the interaction existence rules in the previous task, but accuracy was extremely poor.
- Dependency triples: triples, for example, (combined, resorcinol, dobj), were added to the features, but this only decreased accuracy.
- Same Verb: the shared verb of the entities, if any, was added to the features, but as in task 3 it only decreased accuracy.
- Keyword searches: these were tried, before it was realized that directly appending the words to the features was more effective, as the ML model will identify common keywords itself.

2.2 Chosen Algorithm

The MegaM maximum entropy learner was used for the final results in this assignment. It was attempted to use Naïve Bayes, from the NLTK classifier set, but final accuracy was poor compared to the MegaM classifications. The ease of using the model files to determine informative feature values was another factor for MegaM.

```
def extract_features(id_e1, id_e2, entsToNodes, entities, analysis):
    graph = get_tree_graph(analysis)

    lemmas_before, lemmas_between, lemmas_after = get_words_in_sentence(
    id_e1, id_e2, entities, analysis)

    _2under1 = second_entity_under_first(id_e1, id_e2, entsToNodes, graph)

    preposition = preposition_in_phrase(id_e1, id_e2, entities, entsToNodes,
    analysis, graph)

    betweenrels, words_between = find_dependencies_between(id_e1, id_e2,
    entities, entsToNodes, analysis, graph)

    rels, VB_NN = find_shared_dependencies(id_e1, id_e2, entsToNodes,
    analysis, graph)

    # Adding to feature vector
    features = []
```

```
    features.append("2under1=" + str(_2under1))
    features.append("preposition=" + str(preposition))

    if lemmas_before:
        for i in range(0, len(lemmas_before)):
            features.append("lemmas_before=" + lemmas_before[i])

    if lemmas_between:
        for i in range(0, len(lemmas_between)):
            features.append("lemmas_between=" + lemmas_between[i])

    if lemmas_after:
        for i in range(0, len(lemmas_after)):
            features.append("lemmas_after=" + lemmas_after[i])

    if rels:
        for rel in rels:
            features.append("rels="+rel)

    if VB_NN:
        for word in VB_NN:
            features.append("shared_word="+word)

    if betweenrels:
        for i in range(0, len(betweenrels)):
            features.append("rels_between=" + betweenrels[i])
    return features

def trainMegamMaxEnt():
    print("Loading features...")
    features = read_features("features.txt")
    features2Megam = open("megam_input.txt", "w+")

    for row in features:
        print(row[0], *row[1], sep=" ", file=features2Megam)
    features2Megam.close()
    print("Training model")
    megam_model = "megam_model.dat"
    os.system("megam -nc -nobias -repeat 10 -maxi 1000 -dpp 0.00000001 multiclass
megam_input.txt > " + megam_model)
```

2.3 Devel Evaluator Output

```
Partial Evaluation: only detection of DDI (regadless to the type)
tp	fp	fn	total	prec	recall	F1
218	135	266	484	0.6176	0.4504	0.5209


Detection and Classification of DDI
tp	fp	fn	total	prec	recall	F1
196	157	288	484	0.5552	0.405	0.4683



________________________________________________________________________

SCORES FOR DDI TYPE
Scores for ddi with type mechanism
tp	fp	fn	total	prec	recall	F1
61	69	140	201	0.4692	0.3035	0.3686


Scores for ddi with type effect
tp	fp	fn	total	prec	recall	F1
87	60	75	162	0.5918	0.537	0.5631


Scores for ddi with type advise
```

```
tp    fp    fn    total prec  recall      F1
46    28    73    119   0.6216      0.3866      0.4767


Scores for ddi with type int
tp    fp    fn    total prec  recall      F1
2     0     0     2     1     1     1


MACRO-AVERAGE MEASURES:
      P     R     F1
      0.6707      0.5568      0.6084
________________________________________________________________________
```

2.4 Test Evaluator Output

```
Partial Evaluation: only detection of DDI (regadless to the type)
tp    fp    fn    total prec  recall      F1
374   324   605   979   0.5358      0.382 0.446


Detection and Classification of DDI
tp    fp    fn    total prec  recall      F1
307   391   672   979   0.4398      0.3136      0.3661



________________________________________________________________________

SCORES FOR DDI TYPE
Scores for ddi with type mechanism
tp    fp    fn    total prec  recall      F1
75    117   227   302   0.3906      0.2483      0.3036


Scores for ddi with type effect
tp    fp    fn    total prec  recall      F1
137   195   223   360   0.4127      0.3806      0.396


Scores for ddi with type advise
tp    fp    fn    total prec  recall      F1
79    62    142   221   0.5603      0.3575      0.4365


Scores for ddi with type int
tp    fp    fn    total prec  recall      F1
16    17    80    96    0.4848      0.1667      0.2481


MACRO-AVERAGE MEASURES:
      P     R     F1
      0.4621      0.2883      0.355
________________________________________________________________________
```