Turning Experimental Procedures into Machine-Readable Recipes

William Spitzer, Menghsuan Sam Pan, Iveel Tsogsuren

(Dated: December 10, 2015)

Large scale manufacturing has been made more efficient with automation and machine procedures centuries ago. Scientific researches, however, still require scientists or technicians to perform the experiments due to the inherent variability and complexity. Battery research, in particular, includes large amount of hand-on experiments. In order to help automate the research, we attempt to create step-by-step machine readable procedures from experimental procedure sections of journal publications using common techniques in natural language processing. To achieve this goal, we built a pipeline process that contained several steps. First, it would take in a list of raw sentences from the experimental procedure text and label them using our custom tags to denote whether or not words were (A)ctions, (I)ngredients, (E)quipment, (PR)oducts, (P)roperties, (R)eferences, or (N)one. It would then group together words based on these labels, and separate out steps such that each step contains exactly one Action. Each step is ordered using a shift based dependency parser to connect the tagged groups with each other. Finally, it orders all of the steps based on keywords in the procedure so that each step follows chronologically from the previous step. The result is outputted as a list of dictionaries that represent each step, with details regarding what is needed to perform each step. In order to evaluate the performance of our model, we have built a small data set as our training data and development data.

I. INTRODUCTION

The late 18th century to early 19th century marks the beginning of automation with Industrial Revolution. Since then, the ongoing advancement in technology replaces hand productions with machine manufacturing lines. Now a day, almost all the commercial products, to certain extents, are manufactured or assembled by machine production lines. In contrast, however, most of the scientific research and experiments are still done hands-on largely due to the complexity and variability these experiments can be. Therefore, the ability to convert complex human written procedures (for human) into machine readable recipes (instructions) represents one of the key barrier to fully automated research activities. Recent developments in natural language processing (NLP) researches have solved or touched upon some of the similar sub-problems in converting the instructions. For instance, identifying different contents in a experimental procedure is very similar to part of speech tagging while sorting the ingredients associated with a given action echos dependency parsing problems.

To reduce the overall scope of this project as well as constrain the variability, this project will only deal with electrochemistry, more specifically, battery, procedures. We choose this field of study because a member of our group is extensively involved with the field. However, the model we built are meant to be generalizable to process experimental procedures in any field of study. The experimental procedures were taken from related published scientific journal such as Nano Letters, ACS Applied Materials & Interfaces, Journal of Electrochemical Society, and etc. These procedures were then processed into our training/development data by tagging with our customized experimental labels and shift-based dependency parser shifts. Then we built our NLP model using variants of well-known NLP techniques such as max-entropy classifier, shift-based dependency parser, rule based grouping algorithm, and so on. We also used widely recognized evaluation functions to determine the effectiveness of our model.

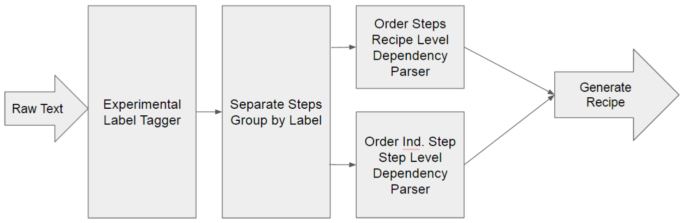

Raw Text
Experimental Label Tagger
Separate Steps Group by Label
Order Steps Recipe Level Dependency Parser
Order Ind. Step Step Level Dependency Parser
Generate Recipe

FIG. 1. Pipeline Process. The raw text input is tagged, grouped, then ordered via dependency parsers by step level and recipe level.

II. DESIGN

II.1. Overall Model

Our overall goal is to take in the raw form of an experimental procedure as text and create a step by step recipe that can be performed by a machine. We defined a step as a single Action performed using some number of Ingredients, Equipments, Products, and References. Each of these can be modified by some number of Properties. We approached the problem of generating these recipes by separating the task into several tasks. The first task involves tagging individual words with a set of experimental labels (A, I, E, PR, E, R, N): (A)ctions, (I)ngredients, (E)quipment, (PR)oducts, (P)roperties, (R)eferences, or (N)one. Following this task, we group up words from the same step together with similar labels into single entities using a rule-based system. During the grouping process,

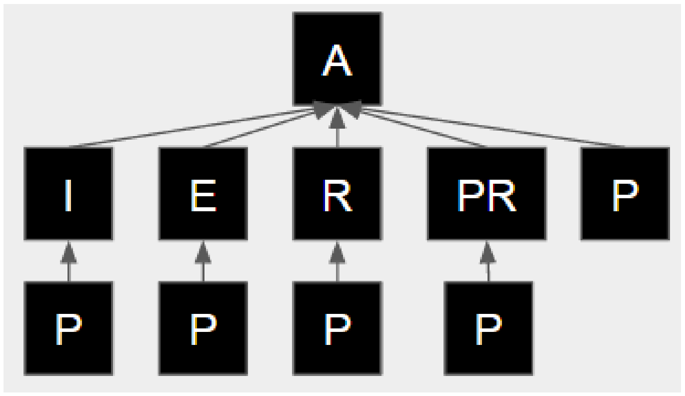
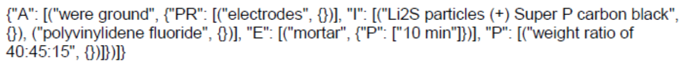


FIG. 2. Structure of a step with the Action as the root. Each action contains some number of Ingredients, Equipments, References, Products, and Properties. Each Ingredient/Equipment/Reference/Product can also be modified by additional Properties.



```
{"A": [("were ground", {"PR": [("electrodes", {})], "I": [("Li2S particles (+) Super P carbon black",
{}), ("polyvinylidene fluoride", {})], "E": [("mortar", {"P": ["10 min"]})], "P": [("weight ratio of
40:45:15", {})]})]}
```

FIG. 3. Structure of a step in dictionary format.

we also extract keywords that are used later to aid in determining the order of the steps such as ”first”, ”second”, ”before”, ”after”, ”then” etc. Once we have the labeled entities, we want to find the relationships between different entities in each step. We applied a transition based dependency parser on these entities to determine the relationships between them. We use this information to build a tree structure that describes the steps in the experiment.

Finally, the steps in the experimental procedure are ordered chronologically via a second dependency parser. The output of our process is a list of dictionary structures, each corresponding to a recipe step. We provide an example of a dictionary that describes a recipe step:

The structure of our dictionary is as follows: The action is at the top. It contains a list of tuples of the form (action string, dict of modifiers). The dict of modifiers contain each possible modifier as shown in the recipe diagram below. Each modifier is again a list of tuples of the form (modifier string, dict of property modifiers). The dict of property modifiers is a list of properties that modify the modifier.

II.2. Experimental Label Tagging and Move Training Set

(A, I, E, PR, E, R, N). These tags are (A)ctions, (I)ngredients, (E)quipment, (PR)oducts, (P)roperties, (R)eferences, or (N)one. An Action is described as **TO DO**

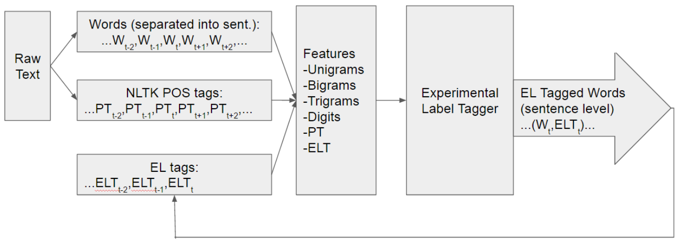

Raw Text
Words (separated into sent.): $...W_{t-2},W_{t-1},W_t,W_{t+1},W_{t+2},...$
NLTK POS tags: $...PT_{t-2},PT_{t-1},PT_t,PT_{t+1},PT_{t+2},...$
EL tags: $...ELT_{t-2},ELT_{t-1},ELT_t$
Features
-Unigrams
-Bigrams
-Trigrams
-Digits
-PT
-ELT
Experimental Label Tagger
EL Tagged Words (sentence level) $...(W_t,ELT_t)...$

FIG. 4. Structure of our Maximum Entropy Tagging Classifier.

II.3. Max Entropy Tagging Classifier

II.3.1. Description

We built a maximum entropy tagging classifier that could tag each input word with an experimental label (EL) tag as discussed in the previous section. We took the basic structure of Honnibal's algorithm [1] and modified it to use additional features and our labels.

The inputs to our classifer were lists of tokenized words from the raw text of the experiment, organized by sentence. Each alphabetical word was converted to lowercase and each numerical word was converted to a ”!NUM” token. In addition, we acquired the NLTK [2] Parts of Speech tag from the NLTK database for each word. Using the tokenized words and POS tags, we generated a wide list of features in order to best classify each word. We used the unigram, bigram, and trigram features for words, POS tags, and EL tags. We also utilized combinations of words and POS tags, and words and EL tags to capture those relationships as well. Once we have generated our features during training, we store them in a dictionary for use in evalution. The dictionary values for each feature is a list of weights corresponding to how likely a label is given that feature. Given a list of features, we can calculate an overall score for each label. Our classifier then chooses the label with the best score given the input features in order to classify each word.

II.3.2. Training

We train our classifier by creating an input feature vector for each word, allowing the classifier to perform a guess for the correct label, then updating the weights for that feature by providing the true label. Each feature that is not yet stored in our dictionary is initialized with a 1 for the correct label, and negative values for the rest, such that the sum of the weights for all labels is 0. If a feature is stored, the classifier compares the guess label with the true label. Given a positive, the weights are kept the same. Given a negative, the weight of the true label is increased by 1 and the other labels are decreased such that the overall sum of the weights for all label is still 0.

We perform training using all the documents in our training set over several iterations (usually around 10). For each iteration, the order of the documents is shuffled to prevent possible overfitting.

II.3.3. *Prediction*

Once we've trained our experimental label tagger, we use it for predictions. We generate the list of feature vectors for each word and apply it to our feature dictionary. If a feature exists in the dictionary, its weights are used. However, if a feature is not found, it is ignored. The final prediction is generated by summing together all the weights across all features and choosing the label with the best weight value.

II.4. Grouper and Step Sorter

II.5. Group-Level Dependency Parser

II.6. Step Level Dependency Parser

III. RESULT AND EVALUATION

IV. DISCUSSION AND FUTURE WORK

Stuff to talk about:

The primary reason that we tag individual words and group later instead of tagging phrases as entities is due to several factors that we do not have a large number of

[1] Honnibal, Matthew. Parsing English in 500 lines of Python. https://spacy.io/blog/parsing-english-in-python
[2] Natural Language Toolkit. http://www.nltk.org/