**PAPER REPORT - Synthesis and Machine Learning for Heterogeneous Extraction**

**NOTE: Pages 1-3 contain a summary of the paper, major observations/takeaways are given under “Additional Comments” in Pages 4-5.**

**Objective:** In this paper, I found that the researchers make use of a novel iterative approach involving a combination of Machine Learning, Program Synthesis and a feedback loop. By iterating the process of re-synthesizing the programs and re-training the ML models, it is found that such an iterative process improves the quality of the extraction process. In our fast paced world, data sources are infinite. Extracting structured information from heterogeneous data becomes an essential in data mining and search, and this paper addresses this with a novel approach beating state of the art approaches and vastly improves the extraction process.

**What is introduced?**

1. A novel algorithm that interleaves Machine Learning model training and program synthesis
2. A synthesis algorithm that takes input as the noisy labels produced by the ML model.
3. Type specifications and Domain Specific Languages are used to improve performance.

The Heterogeneous Extraction Framework and related Algorithms are discussed.

There are two types of Machine Learning models, Generative and Discriminative, here we make use of a discriminative classifier called Conditional Random Fields (CRF) that initially generates noisy labels.  
These noisy labels are improved by further steps involving a Program Synthesizer and a feedback loop that aims to improve the working through more iterations.

**Two datasets are used in the paper:**

Political Dataset: Fields and keys, field values can be unstructured text

M2H Flight Email Dataset: Long tail of formats, variation within formats

Data Extraction Framework: Although there might be lesser training examples, the approach tries to generalize across various formats, using ML models. However, input clusters are made such that all inputs in that particular cluster are similar.

By designing domain specific languages and using program synthesis to produce synthesized programs to perform the extraction for each format, we get to have more control over the extraction process.

**Why use program synthesis?** Using programs instead of ML models enables interpretability of the extraction, as programs (as compared to ML models) are human readable.

The framework consists of an ML model, a program synthesizer, and a semi-automated annotator, which completes the feedback loop.

**Machine Learning Algorithms used:**

1. Conditional Random Fields
2. LSTM-CRF

**Benefit of using Conditional Random Fields:** Only a few training examples are needed. CRFs are able to learn that the token next to a "father name key or K-fname" is a "father name or V-fname" with high probability, this is evident from the fact that the political dataset consists of only 25 annotated training inputs, and yet was able to improve vastly on performance.

For M2H flight dataset, LSTM-CRF is used.

**Issues with the synthesis algorithm:** The synthesis algorithm needs to deal with two major issues: noise in the specification and the multiplicity of formats.

As I described a bit earlier, from the paper it can be read that: The DSL only contains programs that can extract equivalent parts of the outputs. As a result, the synthesized disjunctive program produces correct output on almost all inputs of a format if the ML model is correct on a sizable fraction of inputs in the format. This leads to significant accuracy improvements in the extraction algorithm.

**Semi-Automated Annotator:** Programs are ranked on the basis of number of inputs correctly processed by the program. If an ML model output disagrees with output produced by a highly ranked program, the program’s output is declared the correct label.

However, if the program is not highly ranked, user intervenes. How does this user intervention work? Human annotation, and hence “semi-automated”.

**Setting**: Allow for soft field constraints.

**Heterogeneous Data-Extraction Framework Algorithm:**

The NoisyDisjSynth procedure is given all model outputs for the whole data-set and returns a disjunctive program consisting of many programs. Ideally, each of these programs should be made to always produce the correct output on a single format.

**A way to improve performance of the HDEF:** Add more training data. (ChooseInputs, which are outputs of the Disjunctive Program)

The output of a model for a field f is the set of tokens to which the label corresponding to the field is assigned. Below, we discuss two ML models we use, namely, Conditional Random Fields (CRFs) and Bidirectional Long Short-Term Memory augmented with CRFs (LSTM-CRFs)

**More on the Machine Learning Models:**

* CRFS need a good set of feature functions for training. Deep neural nets provide mechanisims to learn feature functions automatically and hence satisfy the requirement of CRFS.
* The key difference between CRFs and LSTM-CRFs is that the features are manually defined in CRFs, and automatically learned from the data in LSTM-CRF.

**What is a good run of the framework?** The definition of “good” includes agreement with ground truth (whenever provided), satisfaction of field constraints, and agreement between ML models and extracted programs.

The example-based synthesis procedure we use is based on the FlashMeta synthesis framework, initialized with specific DSLs for the political and M2H email data-sets.

**Evaluation:**

Score = w ·constraintScore+(1−w)·conformanceScore

**Aim:** Produce a disjunctive program such that Score(P) > t · card(D), and also having lowest possible cardinality.

**The Noisy Disjunctive Synthesis Algorithm**

First, we usually restrict our sample to have very few examples (1 − 4 in practice) as program synthesis techniques can generalize from very few examples. -> **BENEFIT OF HDEF**

Second, we guide the sampling to pick inputs for which none of the already generated candidates produce a valid output that satisfies the field constraints.

**Aim of Candidate Generation:** Produce candidate programs that correspond to different formats in the dataset.

**Tradeoff with the DSL:** If the DSL is very expressive, the synthesizer is able to more closely approximate the machine learning model M—however, this will not “filter the noise” in SoftSpec as desired. On the other hand, if the DSL is not very expressive, it might not be able to approximate M at all.

**Sampling from P-defined clusters:** Each of these clusters approximate one format in the input. Aim is provide a sufficient variety of training data over the clusters.

**Observation:** Why does M2h has more training data in relation to test data whereas political dataset has very few training data compared to test data?

**Transductive Learning**: Made use of in this paper, here the model trained on labelled data is used to label unlabeled data, and the newly labelled points are augmented to the training data.

**Evaluation:**

­Combining ML and Program Synthesis:

In some cases, the improvement is spectacular, because the NoisyDisjSynth algorithm is able to synthesize programs that cover most of the inputs, in spite of the noisy specifications produced by the ML model.

Reasons why combining led to worse performance:

1. Semantic Conflation: Can be fixed with improving the expressivity of programs.
2. Noise Amplification: In the place of birth field of political data-set, the ML model consistently produces only a prefix of the true output. The HDEF procedure consistently amplifies this error, leading to worse results.

**Effect of Feedback Loop:**

In 90% of cases, performance improvement is observed because of the feedback loop. However, due to the same reasons as stated above, performance degrades as well for a few cases.

**Ablation Study:**

Used to understand impact of DSLs and field constraints.

**Takeaways and Questions:**

1. Approximate cover algorithm is a sort of generate and test approach, used in the creation of the disjunctive program.
2. While the HDEF algorithm is able to handle random noise in ML models using sampling, it does not work in cases with systematic noise, and ends up boosting it. One possible solution is to write field constraints that eliminate the systematic noise. The HDEF algorithm also has difficulty handing inputs with multiple semantic contexts.
3. Since there is less training data, is there a possibility of overfitting?
4. Should explore adding more structural constraints to the program synthesizer to improve its performance? Can help filter the noisy output in ML models. However, one tradeoff would be by adding more constraints we would be reducing the search space.
5. To what other domains/problems can combining ML models and program synthesis be used?
6. When adding more training data from the program synthesis output, that could be a noisy example

**Further Work:** See if same process can be used in other tasks, I’m actually curious about this, whether this could be a general mechanism which data scientists can employ to improve extraction processes.

**Additional Comments/Major Observations:**

Interpretability here refers to us actually knowing what is happening.

1. Importance of the Machine Learning model:
   1. Machine Learning models are better at Generalization. This is where it scores over program synthesis.
   2. The base ML model is very important in this scenario, it is able to produce candidate output labels across the entire heterogeneous dataset, and one major benefit is that it can do so even on formats for which there is no training data.
      1. Examples of such cases where there is no training data but still works are given in the paper, such as the “Airport” field values in M2H and “Father’s name” field in the political dataset.
   3. Since the labels generated by the base ML model are noisy (Due to low training data), we improve this using a feedback loop through a combination with outputs from the next Program Synthesis.
   4. Disadvantage of ML models would be interpretability, this is alleviated by PL scripts.
2. Why does the interleaving between program synthesis and Machine Learning work?
   1. For program synthesis as we can see, a different script is generated for each format, you need separate training data for each format, and poses a concern for unseen data. This disadvantage is overcome by using Machine Learning models that, although noisy, generalize far better, across the entire dataset.
   2. Program synthesis scripts are highly interpretable. For eliminating the noise in the labels produced by ML models, we use a set of disjunctive programs as described in the paper for each sample having one format. Although ML models can handle noise well, owing to the low number of training examples, Program Synthesis is needed. Combining with the above (a) point, ML models alleviate some of the concerns of Program Synthesis.
   3. Judging by these two points, the approach to combine the two solves each other’s concerns, and hence works. Both ML models and Program Synthesis aim to learn f(x) between the input and the output.
3. How is the noise handled? Through a combination of ML with Program Synthesis. Soft constraints are thus made possible. Just the ML models (Even though they handle noise well) with a condition of just a few training examples would find it hard to eliminate this noise, hence, a combination of ML and Program Synthesis makes this possible.
4. A note on the Machine Learning models: CRFs need a good set of feature functions for training, but by using LSTM CRFs, the neural network can learn the features extensively, and relations/dependencies between adjacent tokens is learnt by CRFs.
5. The Approximate Cover Algorithm tries to reduce the cardinality of candidate programs in the final set as much as possible, and maximize the agreement between the ML model and the Program Synthesis. In cases where this does not occur, use the Semi-Automated Annotator and human intervention.
6. In order to achieve low cardinality, it is described to pick a small number of programs that work for a large fraction of inputs, essentially, the clusters define themselves.
7. Semi-Automated Annotator: I had described this earlier in my report, wherein if there is disagreement, if the program is highly ranked, program’s output wins, but in other cases, user intervenes.
   1. By adding more training data (Outputs from highly ranked programs), we retrain the ML model and get better results.
8. The ablation study seemed very interesting to me, especially using different DSLs such as L(FE)(Token-based) and L(T)(Domain-specific). Since I’ve been studying ML, I was always taught to look at domain knowledge as a crucial factor in getting good results. Well-designed DSLs particular to domains help to achieve that, as shown by the results.