Adaptive Rule Discovery for Labeling Text Data

Sainyam Galhotra
UMass Amherst
sainyam@cs.umass.edu

Behzad Golshan
Megagon Labs
behzad@megagon.ai

Wang-Chiew Tan
Megagon Labs
wangchiew@megagon.ai

ABSTRACT

Creating and collecting labeled data is one of the major bottlenecks in machine learning pipelines and the emergence of automated feature generation techniques such as deep learning, which typically requires a lot of training data, has further exacerbated the problem. While weak-supervision techniques have circumvented this bottleneck, existing frameworks either require users to write a set of diverse, high-quality rules to label data (e.g., Snorkel), or require a labeled subset of the data to automatically mine rules (e.g., Snuba). The process of manually writing rules can be tedious and time consuming. At the same time, creating a labeled subset of the data can be costly and even infeasible in imbalanced settings.

To address these shortcomings, we present DARWIN, an interactive system designed to alleviate the task of writing rules for labeling text data in weakly-supervised settings. Given an initial labeling rule, DARWIN automatically generates a set of candidate rules for the labeling task at hand, and utilizes the annotator's feedback to adapt the candidate rules. We describe how DARWIN is scalable and versatile. It can operate over large text corpora (i.e., more than 1 million sentences) and supports a wide range of labeling functions (i.e., any function that can be specified using a context free grammar). Finally, we demonstrate with a suite of experiments over five real-world datasets that DARWIN enables annotators to generate weakly-supervised labels efficiently and with a small cost. In fact, our experiments show that rules discovered by DARWIN on average identify 40% more positive instances compared to Snuba even when it is provided with 1000 labeled instances.

PVLDB Reference Format:
Sainyam Galhotra, Behzad Golshan and Wang-Chiew Tan. Adaptive Rule Discovery for Labeling Text Data. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1. INTRODUCTION

Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

Today, many applications are powered by machine learning techniques. The success of *deep learning* methods in domains such as natural language processing and computer vision is further fuelling this trend. While deep learning (and machine learning in general) can offer superior performance, training such systems typically requires a large set of labeled examples, which is expensive and time-consuming to obtain.

Weak supervision techniques circumvent the above problem to some extent, by leveraging heuristic rules that can generate (noisy) labels for a subset of data. A large volume of labels can be obtained at a low cost this way, and to compensate for the noise, noise-aware techniques can be used for further improving the performance of machine learning models [12, 6]. However, obtaining high-quality labeling heuristics remains a challenging problem. A subset of existing frameworks, with *Snorkel* [12] being the most notable example, rely on domain experts to provide a set of labeling heuristics which can be a tedious and time-consuming task. In contrast, other frameworks aim to automatically mine useful heuristics using further provided supervision. For instance, Snuba [17] circumvents dependence on domain experts by requiring a labeled subset of the data, and then utilizing it to automatically derive labeling heuristics. *Babble labble* is another example which asks expert to label a few examples and explain their choice (in natural language). This explanation is used to derive labeling heuristics. While these approaches have been quite effective in certain settings, we elicit their limitations with the following real-world example on text-data.

EXAMPLE 1. *Consider a corpus that are questions submitted by a hotel's guests to the concierge. Our goal is to build an intent classifier to find (and label) the set of questions asking for directions or means of transportation from one location to another. Below is a sample of messages from the corpus with positive instances marked as green.*

S1	What is the best way to get to SFO airport?	+
S2	Is there a bart from SFO to the hotel?	+
S3	What is the best way to check in there?	-
S4	Is Uber the fastest way to get to the airport?	+
S5	Would Uber Eats be the fastest way to order?	-
S6	What is the best way to order food from you?	-

Relying on domain experts to provide labeling heuristics for tasks such as the one presented in Example 1 is a common approach but it has a number of shortcomings:

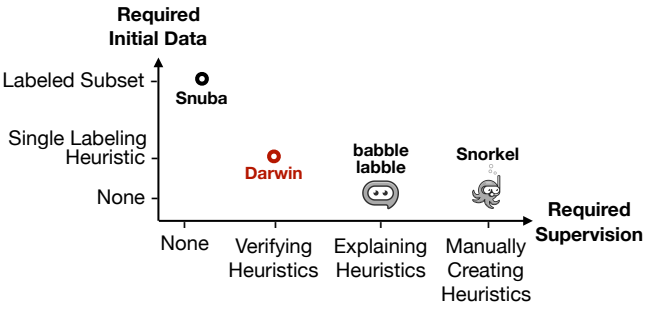


Figure 1: Comparing weak-supervision frameworks

- It is **time consuming**. Annotators must be familiar with the rule language (e.g., Stanford's Tregex or AI2's IKE language). Moreover, they need to be acquainted with the dataset to specify useful rules, i.e., rules that label a reasonable number of instances with a small amount of noise. This is normally done with trial-and-error and fine-tuning of the rules on a sample of the corpus, which can be quite tedious.
- Oftentimes, some **useful rules remain undiscovered**. This is because annotators may miss important keywords or possess limited domain knowledge. For example, the word '*bart*' (which refers to a transportation system in California) is clearly useful for the task in Example 1. However, annotators may miss the important keyword '*bart*' or they may not even know what '*bart*' is (especially those who are not from the area).
- It yields rules with **overlapping results**. If multiple annotators work on writing rules independently, they are likely to end up with identical or similar rules. since Hence, the number of distinct labels obtained does not always grow linearly with the number of annotators, which is rather inefficient.

The alternative approach would be to automatically mine useful heuristics with systems such as Snuba or Babble labble. Both systems require a set of labeled instances (accompanied by natural language explanations in case of Babble labble) which can be costly and oftentimes infeasible to collect in imbalanced settings. For instance while Example 1 shows a balanced number of positive and negative instances, in practice, the positive instances often make up only a tiny fraction of the entire corpus. Hence, labeling a random sample would not be sufficient to obtain enough positive instances. Consequently, automatically inferring heuristic rules is not feasible using the few discovered positive instances.

To mitigate the above issues, we present DARWIN, an adaptive rule discovery system for text data. Figure 1 highlights how DARWIN compares with other state-of-the-art weak-supervision frameworks. Compared to Snuba, DARWIN requires far less labeled instances. In fact, a single labeling rule (or a couple of labeled instances) would be sufficient for DARWIN. Compared to Snorkel and Babble labble, DARWIN requires a lower degree of supervision by domain experts. More explicitly, DARWIN requires experts to simply verify the suggested heuristics while Snorkel requires them to manually write such rules and Babble labble requires them to provide explanations for why a particular label is assigned to a given data point.

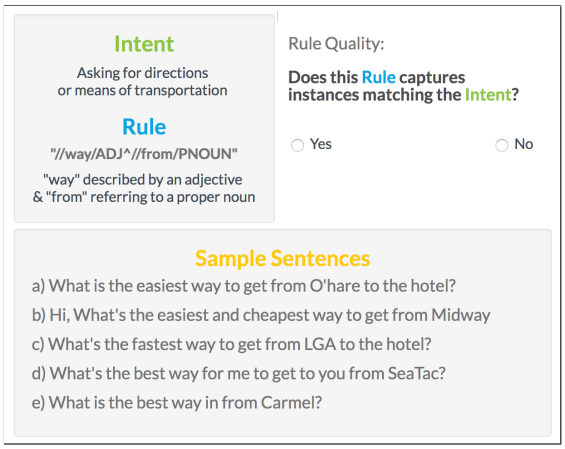


Figure 2: Sample query to annotators.

Given a corpus and a seed labeling rule, DARWIN identifies a set of promising candidate rules. The best candidate rule (along with a few example instances matching the rule) is then presented to the annotator to confirm whether it is useful for capturing the positive instances or not. Figure 2 presents an example of this step for the intent described in Example 1. The annotator is presented with examples that satisfy the rule and asked to answer whether the rule is useful for the intent (a YES/NO question). Based on the response, DARWIN adaptively identifies the next set of promising candidate rules. This interactive process, where rules are illustrated with examples, facilitates annotators to identify the most effective set of rules without the need to fully understand the corpus or the rule language. Our contributions are as follows.

- DARWIN supports any rule language that can be specified using a context-free grammar. Therefore, it can generate a wide range of rules, from simple phrase matching to complex conditions over the dependency parse trees of the sentences.
- DARWIN can effectively identify rules over a large text corpora, even when the number of candidate rules is exponentially large. This is achieved by building indices over the input corpus and we theoretically prove the approximation guarantee of DARWIN.
- DARWIN does not require annotators to be familiar with the rule language. By analyzing the similarity and the overlap between the set of sentences matching different rules, DARWIN automatically surface patterns in data and also supports parallel discovery of rules by asking different annotators to evaluate different rules.
- We demonstrate how DARWIN can be used for a variety of labeling tasks: classify intents, find sentences that mention particular entity types, and identify sentences that describe certain relationships between entities (i.e., relation extraction).

In the following sections, we define our problem, describe DARWIN, and demonstrate its effectiveness and efficiency through a suite of experiments. Specifically, we show that DARWIN outperforms other baseline approaches in its ability to generate a larger set of labeled examples by asking a limited number of questions.

2. PRELIMINARIES & PROBLEM DEFINITION

In a nutshell, DARWIN takes as input an unlabeled corpus of sentences along with an initial seed labeling heuristic (which is assumed to generate at least two positive instances). DARWIN then identifies promising candidate labeling heuristics. DARWIN leverages an oracle to verify whether a particular candidate heuristic is effective at capturing positive instances or not. Finally, the set of discovered heuristics are forwarded to Snorkel [12][1] to train a high precision classifier. Before describing DARWIN's rule discovery pipeline in detail, we provide a formal definition of labeling heuristics along with a description of an oracle.

Heuristic search space. Naturally, labeling heuristics can be of different types with distinct semantics. For example, a heuristic may check for certain phrases in a sentence [17] or it may enforce some conditions on the parse tree [19] and POS tags of a sentence. In DARWIN, the space of possible heuristics are specified using a collection of *Heuristic Grammars*, where each grammar describes a particular type of labeling heuristics. These concepts are formally defined as follows.

DEFINITION 1 (HEURISTIC GRAMMAR). *A Heuristic Grammar $\mathcal{G}$ is a Context-Free Grammar (CFG). Recall that a CFG consists of a collection of derivation rules.*

For a given heuristic grammar $\mathcal{G}$, we define labeling heuristics as follows.

DEFINITION 2 (LABELING HEURISTIC). *A labeling heuristic r is a derivation of the grammar $\mathcal{G}$. We use C_r to denote the set of sentences in the corpus that satisfy the heuristic r, and refer to $|C_r|$ as it's* coverage.

To further clarify the above definitions, let us consider a simplified regular expression grammar called TokensRegex. TokensRegex captures all regular expressions over tokens considering '+' and '*' operators[2]. This grammar can be formally writen using a CFG grammer as shown below.

EXAMPLE 2 (TOKENSREGEX GRAMMAR). *Let $\mathcal{V}$ denote the set of all possible words. The regular expression grammar on the tokens comprises of the following derivation rules.*

$A \rightarrow vA \quad (\forall v \in \mathcal{V})$
$A \rightarrow A + A$
$A \rightarrow A * A$
$A \rightarrow \epsilon$

The above TokensRegex grammar allows for a regular expression of words as a candidate labeling heuristic. For example, this grammar generates heuristics such as 'best way to' or 'shuttle' as well as less meaningful heuristics such as 'shuttle is airport' as candidates for the task described in Example 1. A sentence satisfies the heuristic if it contains that phrase. The sentences $s1$, $s3$ and $s6$ in Example 1 satisfy the heuristic r = 'best way to', hence $C_h = \{s1, s3, s6\}$.

[1]Note that Snorkel both provides a framework for writing labeling rules as well as tools for training noise-aware models. Here we refer to the latter.

[2]We use TokenRegex to explain DARWIN's pipeline. DARWIN functionality is not restricted to this grammar and we discuss more complex grammar.

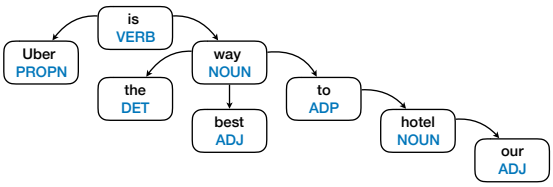


Figure 3: An example of a parse tree.

As a default setting, DARWIN comprises of two different grammars (a) TokensRegex (b) TreeMatch, with the ability to plug in more heuristic grammars as long as they are context-free. While TokensRegex is capable of capturing lexical patterns and phrases, it fails to capture syntactic patterns and pattern over parse trees. TreeMatch grammar captures such patterns to identify more complex and generic heuristic functions.

DEFINITION 3 (TREEMATCH GRAMMAR). *Let $\mathcal{V}$ denote the set of terminals comprising of all the tokens and Part-of-Speech (POS) tags [11] present in the corpus. E.g., NOUN, VERB, etc.*
Derivation Rules: *The grammar has three fundamental operations that make up a heuristic, namely And ($\wedge$), Child (/), and Descendant (//). The symbol 'a/b' implies that terminal 'b' should be a child of terminal 'a' in the dependency parse tree. The symbol 'a//b' implies that terminal 'b' should be a descendant of terminal 'a' in the parse tree. Given that, the derivation rules of the grammar are:*

$\texttt{A} \rightarrow \texttt{/A}$
$\texttt{A} \rightarrow \texttt{A} \wedge \texttt{A}$
$\texttt{A} \rightarrow \texttt{//A}$
$\texttt{A} \rightarrow \texttt{v} \ (\forall \ v \in \mathcal{V})$

It is important to mention that the complexity of heuristics that can be specified using the TreeMatch grammar exceeds what rule-mining frameworks such as Snuba or Babble labble can capture.

Oracle Abstraction. Finally, we formalize the feedback that we may either obtain from a single annotator, a group of annotators, or a crowd-sourcing platform using the notion of Oracles as follows.

DEFINITION 4 (ORACLE). *An Oracle O is a function which given a heuristic r and a few samples from its coverage set C_h outputs a YES/NO answer indicating whether or not r is adequately precise.*

An Oracle plays the role of a perfect annotator who always answers the questions correctly. In practice, annotators may provide incorrect answers (as we show in our experiments), but the notion of an oracle enables us to provide insights into the theoretical aspects of our problem.

Problem statement. We are now ready to formally define our problem. Given a labeling task, our goal is to find a set R of labeling heuristics such that the union of the coverage of the heuristics in R, denoted as $P = \bigcup_{r \in R} C_r$, would have a high recall (i.e., to contain a high ratio of the positive instances in the corpus). We would like to maximize the recall of set P without posing too many queries to the oracle. We empirically observed that users label a heuristic as precise only when the heuristic has precision at least 0.8. Hence,

in this paper, we do not focus on optimizing the precision of heuristics, which we can also rely on various de-noising techniques from the weak supervision literature [12].

PROBLEM 1 (MAXIMIZE HEURISTICS COVERAGE). *Given a corpus S, a seed labeling function r_0, an oracle O, and a budget b, find a set R of labeling heuristics using at most b queries to the oracle, such that the recall of set P, i.e., the union of the coverage of heuristics in R, is maximized.*

LEMMA 1. *The* MAXIMIZE HEURISTICS COVERAGE *problem is NP-hard.*

PROOF. We show the hardness of our problem by reducing the maximum-coverage problem to an instance of our problem. Given a collection of sets $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ and a budget b, the maximum-coverage problem aims to find b sets from $\mathcal{A}$ such that the size of their union is maximized. Given an instance of the maximum-coverage problem, we create an instance of our problem as follows. For each set A_i, we define a heuristic r_i with coverage set $C_{r_i} = A_i$ and mark all the instances as positives. Consequently, the oracle O would always respond with a Yes as all the heuristic are perfectly precise. Now, it is easy to see that the coverage of set P in our setting is equivalent to the coverage of selected sets in the maximum-coverage problem. As a result, if our heuristic discovery problem can be solved in polynomial time, then the corresponding sets would form the optimal maximum-coverage solution. Hence, our problem is also NP-hard. □

Note that while we focus on maximizing the recall, it is also useful to report the performance of the classifier that is trained using our weakly-supervised labels. Therefore, in our experiments, we also record the F-score of our trained classifier to provide a better evaluation of DARWIN.

3. THE DARWIN SYSTEM

In this section, we describe the architecture of DARWIN which is illustrated in Figure 4. DARWIN operates in multiple phases that aim to identify diverse set of heuristics used to identify positives. The pipeline is initialized with a seed labelling function or a couple of positive sentences. DARWIN learns a rudimentary classifier using these positive sentences and the classifier is refined with evolving training data. In order to identify new heuristics DARWIN leverages the following properties. (i) The generalizability of the trained classifier helps guide the search towards semantically similar heuristics. For example, on identifying the importance of 'bus' as a heuristic, DARWIN identifies 'public transport' as another possibility due to their related semantics[3]. (ii) Local structural changes to the already identified heuristics helps identify new heuristics eg. consider 'What is the best way to the hotel?' as input seed sentence, DARWIN constructs local modifications by dropping and adding tokens (derivation rules in general) and identifying a new heuristic 'shuttle to the hotel'. DARWIN leverages these intuitions to adaptively refine the search space and simultaneously learn a precise classifier with high coverage over the positives.

Before describing the architecture, we define a data structure that is critical for efficient execution of DARWIN. All candidate heuristics that are considered by DARWIN are organized in the form of a hierarchy. This hierarchy captures subset/superset relationship among heuristics. Heuristics with higher coverage are placed closer to the root and the ones with lower coverage are closer to the leaves. For example. 'best way to the hotel' is a subset of 'best way to' and will be its descendant. One of the key properties of this hierarchical structure is that if a heuristic r is identified to capture positives, then any of its subset (descendant in the hierarchy) does not capture any new positive. This data structure has $O(1)$ update time to identify the subsets of a heuristic. Additionally, it is helpful for efficient execution of local structural changes to any heuristic. All these benefits will be discussed in detail in the later sections.

[3]This generalization is possible via word embeddings which are provided as an input to the classifier. We provide more details of our classifier in the experiments.

Algorithm 1 DARWIN

Input: Input Corpus $\mathcal{S}$, seed heuristic r_0, budget b
Output: Collection of heuristics R, Set of positive instances P, Classifier C

```
1: I ← generate_index(S)
2: Q ← φ
3: P ← coverage(r_0)
4: C, P' ← train_classifier(P, {r_0}, S)
5: while |Q| ≤ b do
6:    H ← generate_hierarchy(S, P', I)
7:    q ← traversal (H, P, Q, C)
8:    if oracle_query(q) then
9:       P ← P ∪ coverage(q)
10:      C, P'' ← train_classifier(R, P, S)
11:   P' ← P'' \ P'
12:   H ← update_scores(H)
13:   Q ← Q ∪ {q}
14: return R, P, C
```

Algorithm 1 presents the pseudo code of the end-to-end DARWIN architecture. DARWIN's input consists of the corpus to be labeled, a collection of heuristic grammar, and one (or more) seed labeling function(s). Alternatively, a set of positive instances can be provided instead of seed labeling heuristics. The output of DARWIN is the set of generated heuristics, the positive instances that are discovered, and a classifier that is trained using the labeled data.

Before the heuristic-discovery phase begins, DARWIN creates an index over the input corpus for fast access to the sentences that satisfy a given heuristic (more details in Section 3.1). The heuristic-discovery phase is an iterative process where DARWIN interacts with annotators and uses their feedback to identify new candidates and ask further queries. In a nutshell, each iteration consists of the following steps. First, the *Candidate Generation* component generates a small set of promising candidate heuristics (from the space of all possible heuristic functions), and organizes them in the form of a hierarchy $\mathcal{H}$ (line 6) with the most generic functions at the top and the stricter ones at the bottom. We will describe shortly how $\mathcal{H}$ is generated and used to prune less effective heuristics. Once the hierarchy is built, DARWIN's *Hierarchy Traversal* component carefully navigates and evaluates the heuristics in the hierarchy to find the best candidate (line 7). The best candidate is then presented to the annotator (line 8). Finally, the updated classifier and scores of heuristics are sent back to hierarchy generation to identify new candidates and perform traversal for the next iteration. We describe the details of these components next.

3.1 Indexing the Input Corpus

DARWIN creates an index for the input corpus to provide

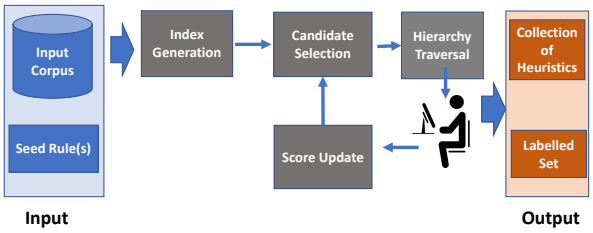


Figure 4: DARWIN's architecture.

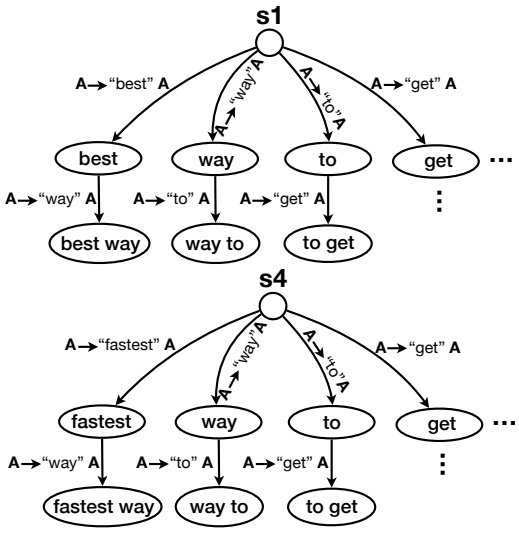


Figure 5: Examples of derivation sketches

fast access to sentences that satisfy certain heuristics. This index aims at constructing a space efficient representation of each sentence in the corpus for efficient execution of subsequent steps involving traversal through the various candidate heuristics. This hierarchical structure of this index is very similar to that of a trie.

Given a collection of heuristic grammar $\{G_1, \ldots, G_t\}$ and a sentence s, one can enumerate the set of all possible heuristics of G_i, generated using a fixed number of derivation rules, that s satisfies. For example, using the TokensRegex grammar, the set of all heuristics that a sentence s satisfies is the set of all regular expressions that correspond to s. We organize the set of heuristics matching a sentence s into a structure called the *Derivation Sketch*, which summarizes the derivations of all heuristics that match s. Figure 5 shows (parts of) the derivation sketch for sentences $s1$ and $s4$ from Example 1 based on the TokensRegex grammar.

A derivation sketch is built for each sentence in the corpus. After this, we create an index $\mathcal{I}$ which is a compact representation of all heuristics that are satisfied by at least one sentence in the corpus. Each node in $\mathcal{I}$ represents a heuristic labeling function and stores the number of sentences that satisfy it, pointers to the children in the index, and an inverted list that points to sentences that satisfy the heuristic.

The index $\mathcal{I}$ is created by merging the derivation sketch of sentences, one at a time. The index is first initialized with the derivation sketch of the first sentence. Thereafter, for every new sentence s, the derivation sketch of s is merged into $\mathcal{I}$ as follows. The root node of the sketch and the root node of $\mathcal{I}$ is merged, and then all nodes (starting form the merged root) are considered in a breadth-first fashion; The children of the node under consideration which are derived

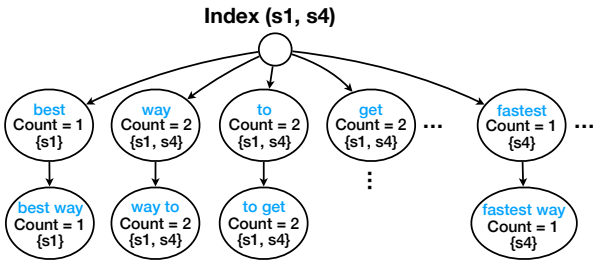


Figure 6: An example of index creation process

using the same derivation rule are merged together. For every node that gets merged, the count of the merged node is increased by one. Also, the inverted list at that node gets updated to include the new sentence. Figure 6 shows the index built from derivation sketches of $s1$ and $s4$ in Figure 5. Note that the time taken to construct the index is linear in the number of sentences since we have limited the number of derivation rules possible to generate a heuristic. The process is also highly parallelizable as index structures for different parts of the corpus can be created independently and then merged. This index also has a linear update time complexity for adding the derivation sketch of a new sentence.

TreeMatch Grammar: This grammar has more operators as compared to TokensRegex and can generate exponentially more candidate heuristics. The derivation sketch can be created as explained by enumerating all sequence of derivation rules up to a fixed number of steps. However, a more compact derivation sketch for the TreeMatch grammar is simply the dependency parse tree of the sentence, as we can use it to quickly check whether a heuristic matches the parse tree or not [19]. Figure 3 shows the dependency parse tree of a sentence which can serve as its derivation sketch as well. Given the exponentially many candidate heuristics generated by this grammar, the candidate generation step is crucial for ignoring useless heuristics and thereby helping the subsequent stages to focus on meaningful heuristics. We evaluate the performance of DARWIN with this grammar in the next section.

3.2 Heuristic-Hierarchy Generation

As mentioned earlier, the number of possible heuristics under a given grammar $\mathcal{G}$ is often exponential in the size of dictionary. The task of the heuristics-hierarchy generation component is to generate a manageable set of promising candidate heuristics from the space of all possible heuristics and organize the generated candidate heuristics in a hierarchy that captures the subset/superset relationship between the heuristics. Specifically, the hierarchy generation process consists of the following steps. First, the *Candidate Generation* step generates a subset of possible heuristics that have high coverage over the set of positive instances discovered so far. This algorithm operates in a greedy best-first search mechanism to identify valuable candidates. These heuristics are promising as they already have some overlap with the existing positive instances. Next, these candidates are arranged in the form of a hierarchy along with subset-superset edges between them. We describe these steps in detail next.

3.2.1 Candidate Generation

The candidate generation step uses the index $\mathcal{I}$ to generate a set of heuristic labeling functions with high coverage

Algorithm 2 Candidate-heuristic Generation

Input: Index $\mathcal{I}$, Set of positive instances P, Number of desired heuristics k
Output: Collection of heuristics R
1: $R \leftarrow \{*\}$, $recentHeuristic \leftarrow *$, $candidates \leftarrow \phi$
2: **while** $|R| \leq k$ **do**
3: $candidates \leftarrow candidates \cup$ Children($recentHeuristic, \mathcal{I}$)
4: $sortedCandidates \leftarrow$ CoverageSort($candidates, P$)
5: $recentHeuristic \leftarrow sortedCandidates[0]$
6: $candidates \leftarrow candidates$.remove($recentHeuristic$)
7: $R \leftarrow R \cup recentHeuristic$
8: **return** R

over the set of positive instances P that are discovered so far by DARWIN. Note that heuristics that (at least partially) cover the set of discovered positive instances, are likely to be good heuristics and help detect more positive instances. To efficiently find such heuristics, we rely on one of the interesting properties of index $\mathcal{I}$; Recall that the count of a node $u \in \mathcal{I}$ refers to the total number of sentences that contain the tokens along the path from the root to u in their derivation sketches. As descendant nodes correspond to stricter heuristics, the coverage of a heuristic corresponding to a node is never less than the coverage of any of its descendants in the index. Thus we use a greedy algorithm to identify a collection of diverse heuristics that have high coverage over the set of positive instances P.

Algorithm 2 generates candidate heuristics by exploiting the property described above. The set of candidate heuristics is initialized with heuristic '*' which refers to the root of index $\mathcal{I}$. This heuristic matches all possible sentences in the corpus. In each iteration, the algorithm adds the children of the previous iteration's best candidate heuristic to the candidate list (line 3). The candidates are then sorted in decreasing order of coverage over the set P (line 4). The candidate with the highest coverage is removed from the candidate list and appended to list of final results R (lines 6-7). This process is repeated until there are k heuristics in R. Note that the time complexity of this greedy algorithm is linear in the number of candidates generated.

Other constraints can also be added to the candidate-heuristic generation phase to ensure that the generated heuristics satisfy those criteria. For example, DARWIN can apply heuristics to ensure that the candidate heuristics are diverse in terms of the set of derivation rules used to derive the heuristic, their level in the index $\mathcal{I}$, and the set of instances they cover. Some of these heuristics help DARWIN avoid having to evaluate many similar candidate heuristics.

Hierarchical Arrangement and edge discovery. The candidates returned by Algorithm 2 have high coverage over the positives (discovered so far). This component iterates over the generated heuristics to arrange them into a hierarchy $\mathcal{H}$ following the same parent/child relationship that index $\mathcal{I}$ captures[4] and an edge is added between them.

This hierarchical arrangement of heuristics is followed by a cleanup to get rid of heuristics that do not add any new positive sentences than the ones already identified. The goal of cleanup is to improve the efficiency and space complexity of DARWIN as the traversal component will never query a heuristic that does not add any new positives.

3.3 Hierarchy Traversal

[4]heuristic r_1 is a child of r_2 if it can be obtained by applying a derivation rule to r_2.

The result of the heuristic-hierarchy generation is a hierarchy $\mathcal{H}$ of promising heuristics. The hierarchy traversal module determines which heuristic in the hierarchy is the best heuristic to be submitted to the oracle.

We present three hierarchy traversal techniques: `LocalSearch`, `UniversalSearch`, and `HybridSearch`. At a high level, `LocalSearch` relies on the hierarchy structure to select the next best candidate from the immediate neighborhood of heuristics verified by the oracle in the past. In contrast, `UniversalSearch` ignores locality constraints and selects the heuristic with maximum benefit globally.

Finally, the `HybridSearch` traversal combines the first two techniques to find the next best heuristic. The `HybridSearch` traversal is more robust than `LocalSearch` and `UniversalSearch`. All three techniques work in an iterative fashion, and in each iteration, the criteria for selecting a heuristic to be sent to the oracle is based on how *beneficial* the heuristic is, which we elaborate next.

Benefit of a heuristic (r): The benefit of a heuristic r is the expected gain in the positive set P upon choosing r. More formally, the benefit is quantified as $\sum_{s \in C_r \setminus P} p_s$, where p_s is the probability of sentence s being a positive instance. In DARWIN, these probability values are estimated by training a classifier[5] using the set of positive instances discovered so far and sampling random instances from the corpus as negatives. The probability estimates improve as the system iteratively discovers more heuristics and the classifier is re-trained with more positive training examples.

We describe our three traversal techniques next.

3.4 `LocalSearch`

`LocalSearch` traversal algorithm (Algorithm 3) benefits from the local hierarchy structure around the heuristics already identified as useful by the oracle to identify the next best heuristic for querying. Specifically, `LocalSearch` maintains a set of candidate heuristics, and selects the most beneficial heuristic r from the candidates. If the oracle confirms that r is adequately precise, then it adds r's parents into the candidate set as they are generalizations of r and might be helpful at capturing more positive instances. However, if the oracle labels r as a noisy heuristic, `LocalSearch` adds the children of r to the candidate set instead with the hope that a specialized version of heuristic r might be less noisy.

`LocalSearch` is simple and efficient at utilizing the structure of the hierarchy to find promising heuristics to submit to the oracle. Since the algorithm only explores the local neighborhood of the queried candidates, it has a time complexity of $O(dt)$, where d is the maximum degree of an internal node and t is the number of iterations the algorithm is running for. However, a disadvantage of `LocalSearch` is that it may require many traversal steps in cases where the initial seed heuristic is quite different from other precise heuristics the system aims to discover. Also, it does not exploit the similarity and the overlap between the coverage sets of different heuristics. The `UniversalSearch` algorithm, which we describe shortly, addresses these shortcomings by utilizing a holistic view of the hierarchy.

Efficient Implementation. Since the `LocalSearch` traversal only explores a node's immediate parents/children, it

[5]Any short text classifier would be ideal for this task.

Algorithm 3 `LocalSearch` Traversal

Input: heuristic hierarchy $\mathcal{H}$, Seed heuristic r_0
Output: Collection of positive instances P, Collection of heuristics R

$QueryCount \leftarrow 0$
$R \leftarrow \{r_0\}, \quad P \leftarrow C_{r_0}, \quad \mathcal{C} \leftarrow \text{TrainClassifier}(P)$
$localCandidates \leftarrow \{r_0\}$
while $QueryCount < b$ **do**
  $r \leftarrow \text{GetMostBeneficialCandidateheuristic}(localCandidates, \mathcal{C})$
  $QueryCount \leftarrow QueryCount + 1$
  if OracleResponse(r) is YES **then**
    $R \leftarrow R \cup r, \quad P \leftarrow P \cup C_r$
    $localCandidates \leftarrow (localCandidates \setminus \{r\}) \cup \text{Parents}(r)$
    $\mathcal{C} \leftarrow \text{TrainClassifier}(P)$
  else
    $localCandidates \leftarrow (localCandidates \setminus \{r\}) \cup \text{Children}(r)$
return P, R

does not require the entire hierarchy apriori. Hence, in its implementation, we can skip the heuristic-generation component and expand the hierarchy on the fly based on the oracle's feedback.

3.5 `UniversalSearch`

The `UniversalSearch` algorithm (see Algorithm 4) evaluates all heuristics present in the hierarchy to identify the best heuristic. In each iteration, `UniversalSearch` omits any heuristic for which the benefit per instance is smaller than 0.5, i.e. majority of the instances in C_r are expected to be negatives. Among the remaining heuristics, it chooses the heuristic with maximum benefit to submit to the oracle. Based on oracle's feedback, it re-trains the classifier if new positives were discovered or else it continues to query the next best heuristic to the oracle. Note that `UniversalSearch` captures the best candidates irrespective of the hierarchy structure.

The strength of `UniversalSearch` is in its capability to identify semantic similarity between heuristics and their matching instances even if they are far apart in the hierarchy. However, it has the following shortcomings: (1) compared to `LocalSearch`, it is inefficient as it iterates over all heuristics in the hierarchy to identify the best candidate, and (2) in absence of enough positive instances, the trained classifier is likely to overfit and not generalize well to other precise heuristics. In such cases, `UniversalSearch` fails to exploit the structure of the hierarchy to at least find heuristics that are structurally similar to the seed heuristics.

We describe the `HybridSearch` algorithm next, which combines the strengths of `UniversalSearch` and `LocalSearch`.

3.6 `HybridSearch`

`HybridSearch` (See Algorithm 5) combines the two previous traversal techniques by maintaining a list of local candidates and a list of universal candidates, and imitating the strategy of the both traversal algorithms. Starting from the `UniversalSearch` strategy, the `HybridSearch` algorithm queries candidate heuristics (with a benefit per instance above 0.5) to the oracle. If the algorithm fails to find a precise heuristic within a fixed number of attempts, then it switches to the `LocalSearch` strategy. Similarly, if the `LocalSearch` strategy has no success within a fixed number of attempts, the traversal toggles to the `UniversalSearch` strategy. The switch between the two strategies is decided based by a parameter τ (by default 5) which denotes the number of unsuccessful attempt before the switch happens.

Algorithm 4 `UniversalSearch` Traversal

Input: heuristic hierarchy $\mathcal{H}$, Seed heuristic r_0
Output: Collection of positive instances P, Collection of heuristics R

$QueryCount \leftarrow 0$
$R \leftarrow \{r_0\}, \quad P \leftarrow C_{r_0}, \quad \mathcal{C} \leftarrow \text{TrainClassifier}(P)$
$universalCandidates \leftarrow \{r : r \in \mathcal{H}\}$
$\mathcal{C} \leftarrow \text{TrainClassifier}(P)$
while $QueryCount < b$ **do**
  $r \leftarrow \text{GetMostBeneficialCandidate}(universalCandidates, \mathcal{C})$
  $QueryCount \leftarrow QueryCount + 1$
  if AvgBenefit(r) ≤ 0.5 **then** continue
  if OracleResponse(r) is YES **then**
    $R \leftarrow \{r_0\}, \quad P \leftarrow C_{r_0}$
    $\mathcal{C} \leftarrow \text{TrainClassifier}(P)$
  $universalCandidates \leftarrow universalCandidates \setminus \{r\}$
return P, R

Algorithm 5 `HybridSearch` Traversal

Input: heuristic hierarchy $\mathcal{H}$, Seed heuristic r_0
Output: Collection of positive instances P, Collection of heuristics R

$universalMode \leftarrow \text{True}, \quad attempt \leftarrow 0$
$R \leftarrow \{r_0\}, \quad P \leftarrow C_{r_0}, \quad \mathcal{C} \leftarrow \text{TrainClassifier}(P)$
$localCands \leftarrow \{r_0\}, \quad universalCands \leftarrow \{r : r \in \mathcal{H}\}$
$QueryCount \leftarrow 0$
while $QueryCount < k$ **do**
  if $attempt \geq \tau$ **then**
    $universalMode \leftarrow$ **not** $universalMode$
    $attempt \leftarrow 0$
  $attempt \leftarrow attempt + 1$
  $candidates = universalCands$ **if** $universalMode$ **else** $localCands$
  $QueryCount \leftarrow QueryCount + 1$
  $r \leftarrow \text{GetMostBeneficialCandidateheuristic}(candidates, \mathcal{C})$
  if $universalMode$ and AvgBenefit(r) ≤ 0.5 **then** continue
  if OracleResponse(r) is YES **then**
    $R \leftarrow R \cup r, \quad P \leftarrow P \cup C_r$
    $\mathcal{C} \leftarrow \text{TrainClassifier}(P)$
    $localCands \leftarrow localCands \setminus \{r\} \cup \text{Parents}(r)$
  else
    $localCandidates \leftarrow localCandidates \setminus \{r\} \cup \text{Children}(r)$
  $universalCands \leftarrow universalCands \setminus \{r\}$
return P, R

Clearly, higher values of τ discourage switching between the two strategies.

Our empirical evaluation shows that `HybridSearch` formed by combining `UniversalSearch` and `LocalSearch` strategies, runs well on all types of datasets even when the other two traversal algorithms struggle to discover high-quality heuristics. In short, if the trained classifier is noisy (due to lack of positive instances), `HybridSearch` exploits the structure of the hierarchy to search for precise heuristics. Similarly when no precise heuristics are found by `LocalSearch`, it uses `UniversalSearch`'s ability to generalize to other heuristics.

3.7 Score Update

After a query is submitted to the oracle, Darwin passes the feedback to the *score update* component to (1) re-train the classifier, (2) re-evaluate the scores of all heuristics in the hierarchy, and (3) update the set of positive instances (if feedback is positive) and signal the hierarchy generation component to generate new candidate heuristics to be added to the hierarchy.

3.8 Theoretical Analysis

In this section we analyze the ability of `UniversalSearch` hierarchy traversal to identify majority of the positive sentences within the query budget b. For this analysis, we consider a noise model to define the probability of a sentence to be positive as assigned by a classifier. In ideal, setting the classifier assigns a score of 1 to positives and 0 to negatives. We consider a noise model that estimates the probability of a sentence s to be positive. Let P^* denote the collection of positive sentences in the corpus of sentences S. According to our model, the score of a positive sentence $s \in P^*$ is more than $\theta \geq 0.5$ with a probability of β and less than $1-\theta$ otherwise. The score of a negative sentence $s \in S \setminus P^*$ has the same distribution with a probability of β' in place of β. Under this noise model, we want to show that the set of heuristics R and the corresponding positive set P identified by `UniversalSearch` are constant approximation of the optimal solution. It is safe to assume that the classifier is better than random i.e. $\beta > \beta'$ and $\theta > 0.5$.

For this analysis, we assume that the heuristics considered for exploration have a coverage of $\Omega(\log n)$ and there are polynomial number of total heuristics in the hierarchy $\mathcal{H}$, say $O(n)$. This is a realistic assumption as we want our algorithm to run in polynomial time and our algorithm is aimed at identifying heuristics with large fraction of positives. Under this assumption, we show that at any iteration, a heuristic r chosen by `UniversalSearch` has coverage $|C_r| > \alpha \max_{r' \in \mathcal{H}} |C_{r'}|$, where α is a constant. This guarantees that `UniversalSearch` identifies atleast αP^* positives within a query budget of b, where P^* is the total number of positives identified by an ideal algorithm. To bound the estimated coverage of a heuristic r, we use the Hoeffding's inequality [7].

Notation. We define a random variable X_s which refers to the score assigned to a sentence s and let μ_s denote its expected value. The benefit score of a heuristic r is $\sum_{s \in C} X_s$.

LEMMA 2. *Given a heuristic function r with coverage of C_r and precision p, the expected score of the heuristic function is atleast $\theta\beta'|C_r|$*

PROOF. Expected score of the heuristic function is

$$\begin{aligned} E\left[\sum_{s \in C_r} X_s\right] &= \sum_{s \in C_r} \mu_s = \sum_{s \in C_r \cap P} \mu_s + \sum_{s \in C_r \setminus P} \mu_r \\ &\geq \sum_{s \in C_r \cap P} (\theta\beta) + \sum_{s \in C_r \setminus P} \left(\theta\beta'\right) \\ &= (\theta\beta)\, p|C_r| + (1-p)|C_r| \left(\theta\beta'\right) \\ &\geq \theta\beta'|C_r| \end{aligned}$$

□

We use this calculation to bound the score of a heuristic r that has more than $\log n$ sentences.

LEMMA 3. *Consider a heuristic function r with coverage C_r such that $|C_r| = c \log n$ sentences, where $c \geq \frac{2}{\epsilon^2 \theta^2 \beta'^2}$ is a constant. The score of the heuristic is atleast $(1-\epsilon)\theta\beta'|C_r|$ with a probability of $1 - 2/n^4$.*

PROOF. The score of heuristic function r is $\sum_{s \in C_r} X_s$. The expected value of the score is calculated in lemma 2. Using Hoeffding's inequality,

$$\begin{aligned} Pr\left[\frac{1}{|C_r|}\sum_{s \in C_r} X_s \leq (1-\epsilon)\mu_r/|C_r|\right] &\leq 2e^{-2\epsilon^2\mu_r^2/|C_r|} \\ &= 2e^{-2\epsilon^2\theta^2\beta'^2|C_r|} \\ &\leq 2e^{-4\log n} = \frac{2}{n^4} \end{aligned}$$

This shows that $\frac{1}{|C_r|}\sum_{s \in C_r} X_s$ is greater than $(1-\epsilon)\theta\beta'|C_r|$ with a probability more than $1 - \frac{2}{n^4}$ □

Using a similar analysis, we identify an upper bound of the heuristic score. Due to space constraint, we defer the proof to full version[6].

LEMMA 4. *Given a heuristic function r with a coverage of C_r and precision p, the expected score of the heuristic function is atmost $(\beta + (1-\theta)(1-\beta))\,|C_r|$.*

LEMMA 5. *Consider a heuristic r with coverage C_r such that $|C_r| = c \log n$ sentences, where $c \geq \frac{2}{\epsilon^2(\beta+(1-\theta)(1-\beta))^2}$ is a constant. The score of the heuristic is atmost $(1+\epsilon)\,(\beta + (1-\theta)(1-\beta))\,|C_r|$ with a probability of $1 - 2/n^4$*

Using the calculated bounds on score of a heuristic, we evaluate the condition when a particular heuristic is preferred over the other.

LEMMA 6. *Given a pair of heuristic functions r_1 and r_2 with respective coverage C_1 and C_2. If C_1 has more positives than C_2, the `UniversalSearch` score of r_1 is higher than that of r_2 whenever $\frac{|C_1|}{|C_2|} \geq \alpha$ with a probability of $1 - \frac{4}{n^4}$, where α is a constant.*

Using a similar analysis, we can calculate the estimated average probability of a heuristic. For a heuristic r with precision p_r, we can show that it is considered for benefit calculation only when $p_r > \gamma$, where γ is a constant.

THEOREM 1. *In worst case, `UniversalSearch` provides a constant approximation of Problem 1 with a probability of $1 - o(1)$*

PROOF. In each iteration, `UniversalSearch` algorithm sorts each of the candidate heuristic based on estimated average probability of a randomly chosen sentence from C_r to be positive. All these candidates have true precision $p_r > \gamma$. Given a pair of heuristics r_1 and r_2, using Lemma 6, the benefit score of a block r_1 is higher than that of r_2 whenever $\frac{|C_{r_1}|}{|C_{r_2}|} > \alpha$ with a probability of $1 - \frac{4}{n^4}$. Let r_{OPT} be the heuristic chosen by optimal algorithm. Using union bound over $\binom{n}{2}$pairs of heuristics, with a probability of atleast $1 - \frac{4}{n^2}$, the estimated benefit of r_{OPT} is higher than that of any r' whenever $|C_{r'}| \leq |C_{r_{OPT}}|\alpha$. Therefore, `UniversalSearch` never chooses any block heuristic with coverage smaller than $|C_{r_{OPT}}|\alpha$ with a probability of $1 - o(1)$. This shows that the total number of positives identified by `UniversalSearch`are atleast $|C_{r_{OPT}}|\alpha\gamma$, which is a constant approximation of $|C_{r_{OPT}}|$ with a probability of $1 - o(1)$. □

[6] `https://people.cs.umass.edu/~sainyam/darwin_fullversion.pdf`

dataset	# Sentences	% Positives	Labeling
cause-effect	10.7K	12.2	Relations
musicians	15.8K	10	Entities
directions	15.3K	3.8	Intents
profession	1M	1.1	Entities
tweets	2130	11.4 (Food)	Intents

Table 1: Dataset statistics

Notice that this proof assumes access to a classifier that scores each sentence s with a probability sampled according to this error model. In the initial iterations of DARWIN the recall of the classifier is low and hence, the values of β and θ are lower. As DARWIN identifies new heuristic functions, the increase in training data pushes these values to a higher value, thereby improving the approximation factor of our algorithm. Even when the classifier is not ideal, our only analysis assumes that the classifier is better than random and positive sentences have higher likelihood of getting a higher score than negative ones.

Discussion. We proposed three techniques for hierarchy traversal. `UniversalSearch` approach is useful to capture holistic information about the different candidate labeling heuristics and is proven to achieve constant approximation of the optimal solution under reasonable assumptions of the trained classifier. However, due to lack of training data in initial iterations of the pipeline, this assumption may not hold and `UniversalSearch` does not perform optimally. However, `LocalSearch` performs local generalization of identified heuristics to quickly increase the number of identified positives. `HybridSearch` is a robust amalgamation of these two techniques and is recommendation. Since `HybridSearch` is a generalization of `LocalSearch` and `UniversalSearch`, it is slightly less efficient than either of these.

4. EXPERIMENTS

In this section, we perform empirical evaluation of DARWIN along with other baselines to validate the following.

- The ability of DARWIN to identify majority of the positives even when initialized with a small seed set.
- The positives identified by DARWIN outperform other baseline techniques that use active learning, a human annotator or any other automated techniques. We show that DARWIN can uncover most of positive instances (i.e, 80% or more) with roughly 100 queries.
- The heuristics identified by DARWIN have high precision and help train a classifier with superior F-score (≥ 0.8).
- DARWIN is highly efficient and can generate labels from a corpus of 1M sentences in less than 3 hrs. DARWIN performance is resilient to variations in the seed set.

4.1 Experimental Setup

Here, we describe the datasets, the baselines, and our overall experimental setup.

Datasets. We experimented with five diverse real-world datasets each suitable for one of the following NLP tasks: entity extraction, relationship extraction, and intent classification. Table 1 summarizes the statistics of these datasets. All datasets, except for `directions`, come with ground-truth labels which we use for evaluation and to synthesize the responses from an oracle. For the `directions` datasets, we rely on human annotators to generate the gold standard and validate the heuristics. We describe each of these dataset below.

- `cause-effect` [15] is a dataset commonly used as a benchmark for relationship extraction between pairs of entities. We focus on the task of finding sentences that describe a cause and effect relationship between two entities.
- `directions` is an internal dataset described in Example 1. For this dataset, we leveraged Figure-eight[7] crowd workers to verify the heuristics generated by DARWIN.
- `musicians` dataset consists of sentences from Wikipedia articles. The task is entity extraction with the goal to extract the names of musicians. The ground-truth is obtained with the help of NELL's knowledge-base[8].
- `professions` dataset is a collection of sentences from ClueWeb[9]. The sentences that mention various professions (e.g., scientist, teacher, etc.) are positives. The ground truth is generated using NELL's knowledge-base.
- `tweets` [18] data set is a benchmark for classifying the intent of tweets into predefined categories such as food, travel and career, etc.

Baselines. We evaluate our framework on two fronts: (1) the ratio of positive instances it discovers (i.e. coverage) and (2) the performance of the classifier trained using our weakly-supervised labels. Our baselines for these two evaluation criteria are listed below.

- Section 4.2 compares the fraction of positives identified with `Snuba`[17]. In this experiment, we consider a small sample of positives chosen randomly from the dataset.
- Section 4.3 compares the coverage obtained by DARWIN against two baselines, namely `HighP` and `HighC`. `HighP` is a simpler version of DARWIN which selects the rule which is expected to have a high precision (according to the classifier) and submits it to the oracle. On the other hand, `HighC` selects rules with the maximum coverage, irrespective of their expected precision[10].
- Section 4.4 compares the F-score of the classifier generated by DARWIN with an *Active Learning* (`AL`) [14] and a *Keyword Sampling* (`KS`) technique as well as the `HighP` baseline mentioned earlier. `AL` improves its performance by selecting the instance with the highest entropy and asking the oracle for its label. It then re-trains the classifier using the new label. The `KS` approach is designed to check if we can quickly obtain a small set of promising instances by filtering the corpus using a set of relevant keywords, and label the instances in the smaller set. To do so, we asked annotators to provide 10 distinct keywords as a heuristic to filter the dataset. The `KS` technique randomly samples instances from the filtered dataset and asks for its label. We employ the same deep learning based classifier for all the techniques.

Finally, note that DARWIN can use different traversal algorithms: `LocalSearch`, `UniversalSearch`, and `HybridSearch`, which we refer to as DARWIN(LS), DARWIN(US), and DARWIN(HS) respectively.

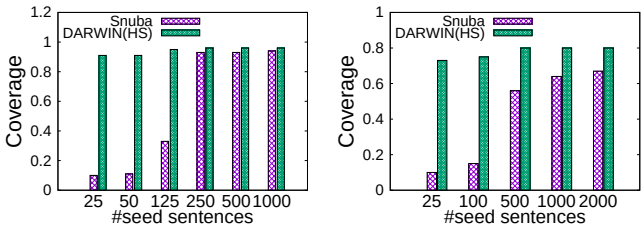
Settings. We implemented all proposed algorithms and baselines in Python and ran the experiments on a server with a 500GB RAM and 64 core 2.10GHz x 2 processors.

[7] https://figure-eight.com

[8] `http://rtw.ml.cmu.edu/rtw/kbbrowser/`

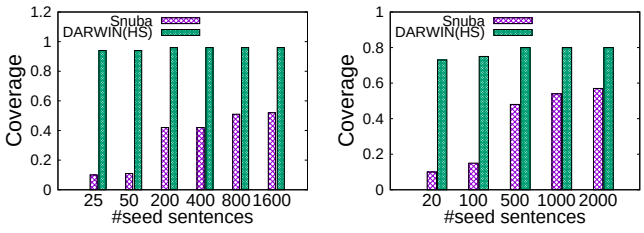
[9] `https://lemurproject.org/clueweb09/`

[10] `HighC`'s performance was quite poor as most of its suggested rules are rejected by the oracle. As a result, we omit HighC from the plots for the sake of clarity.



(a) `directions` (b) `musicians`

Figure 7: Effect of varying initial seed set.



(a) `directions` (b) `musicians`

Figure 8: Effect of varying initial seed set.

The dependency parse trees and the POS tags are generated with SpaCy[11]. All text classifiers trained in our experiments (whether used by DARWIN or other baselines) are implemented with a 3-layer convolutional neural network followed by two fully connected layers, following the architecture described by Kim et al [8]. The input to the classifier is a matrix created by stacking the word-embedding vectors of the words appearing in the sentence. We also used SpaCy's word-embeddings for English[12]. For generating derivation sketches, the maximum depth is set to 10 and we consider 10K heuristics in candidate selection. When simulating the responses from an oracle (using the ground-truth data), we respond YES to heuristic h if at least 80% of its coverage set consist of positive instances.

4.2 Comparison with Snuba

In this experiment, we initialize `Snuba` and DARWIN(HS) with the same set of randomly chosen sentences and compare the total number of positives identified by each of the techniques[13]. The `Snuba` technique does not query the oracle and infers heuristics based on the input labelled instances. For the sake of fairness, we do not compare the accuracy of identified heuristics. Figure 7 shows the change in fraction of identified positives with the change in initial seed set. DARWIN(HS) is able to identify majority of positives even when the pipeline is initialized with less than 25 sentences. However, `Snuba` requires atleast 200 randomly chosen sentences for `directions` and 1000 for `musicians`. If we employ expert to sample positives, `Snuba` requires atleast 100 positive samples in `musicians`.

To evaluate the generalizability of `Snuba` and DARWIN(HS) to identify heuristics that have limited or no evidence in the initial seed set, we construct a biased sample of seed positives. In this experiment, we choose sentences randomly from the corpus after ignoring the ones that contain the token 'shuttle' in `directions` dataset and 'composer' in `musicians`. Figure 8 shows the fraction of positives identified with varying size of the seed set. `Snuba` is not able to identify the positives that contain the token 'shuttle' in `directions` and 'composer' in `musicians`. Henceforth, it achieves poor coverage over the positives in two datasets. DARWINHS is able to identify majority of positives irrespective of the number of sentences used to initialize the pipeline. `Snuba` requires considerably more labelled sentences in `musicians` as compared to `directions` because of presence of many diverse heuristics in the dataset, most of which have limited evidence in the seed subset. We observe similar performance gap between `Snuba` and DARWIN(HS) for other datasets. Due to lack of space, we omit other plots.

This experiment validates that `Snuba` works well when the initial seed set has enough randomly chosen positives and lacks the ability to generalize to heuristics that have limited evidence. On the other hand, DARWIN(HS) is able to identify majority of the positives even when the pipeline is initialized with just 25 sentences and has good generalizability. To further evaluate DARWIN's ability to identify positives, the following subsection considers a more realistic scenario where the pipeline is initialized with just two positive sentences or a single labelling heuristic.

4.3 Rule Coverage

Figures 6a-6d and 10a illustrate the fraction of positives identified by DARWIN and our baselines. We can observe that DARWIN(HS) has the most stable performance and outperforms other techniques. While DARWIN(US) occasionally outperforms DARWIN(HS), we observe that it fails to perform well on all datasets. In most cases (with an exception of `cause-effect`), the DARWIN(HS) achieves a coverage of 0.8 using less than 120 queries to the oracle. The `cause-effect` is known to be a tough benchmark in the NLP community as the best F-score reported by [15] is 82% given complete access to the training set. Assuming that the oracle considers a majority vote by querying three crowd members and each query costs 2 cents[14], the DARWIN(HS) pipeline generates more than 80% of the positive labels with only $7.20. Figure 6d demonstrates the behavior for 'Food' intent in the tweets. We observed similar behavior for 'Travel' and 'Career' intents on this data set. We can observe that the other baselines do not perform well compared to DARWIN; The `highP` identifies heuristics with very small coverage as its candidates. Also note that the DARWIN(LS) algorithm shows a high progressive coverage initially but it converges to a very low coverage value because it is unable to identify rules that are semantically similar, but far away in the hierarchy. Overall, we recommend DARWIN(HS) for any practical application as it is more robust and works better than most of the techniques. On the other hand, DARWIN(LS) and DARWIN(US) variants work well in specific settings. DARWIN(LS) performs better than the other techniques when precise rules are present close to each other in the hierarchy and DARWIN(US) performs well in the presence of abundant labelled examples.

Figure 11 shows some the heuristics which are queried by the DARWIN(HS) algorithm. In the `directions` example,

[11] `https://spacy.io/`

[12] `https://spacy.io/models/en\#en_core_web_lg`

[13] In this experiment, we do not start with a single labeling heuristic because `Snuba` does not get more than 0.4 coverage in that case. This is expected because `Snuba` is suited for an unbiased labeled subset of the dataset. Due to high imbalance, the datasets require a lot of samples to contain sufficient representation of positives.

[14] These are standard assumptions in crowdsourcing platforms eg. figure-eight. We used the same cost model to collect labels for `directions`.

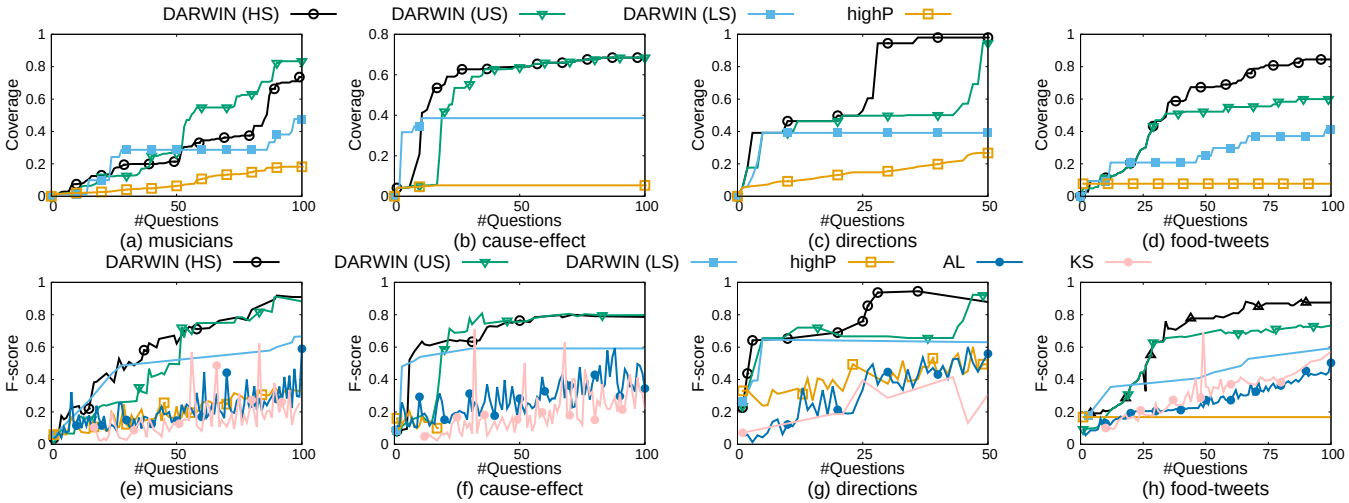


Figure 9: Comparison of rule coverage and classifier's F-score for DARWIN based pipelines on various datasets.

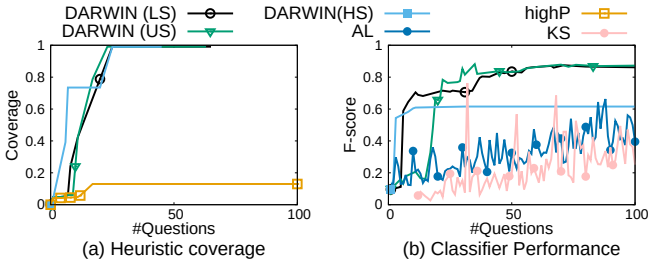


Figure 10: Comparison for `profession`.

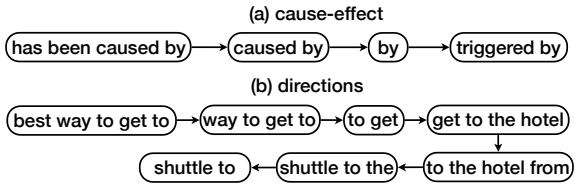


Figure 11: Example of traversals by `HybridSearch` algorithm on two datasets.

DARWIN(HS) started with '*best way to get to*' and was able to traverse to '*shuttle to*', which is quite distinct from the initial seed rule. The choice of '*to the hotel from*' by the algorithm provides some evidence that '*shuttle to*' is also a good rule since the phrases often co-occur together in positive instances. In the `cause-effect` example, the traversal is relatively simple as the algorithm generalizes the initial rule first and as soon as it reaches the noisy and unhelpful rule '*by*', it again specializes to '*triggered by*' which is again a precise rule. In addition to these simpler heuristics, DARWIN identified more complex heuristics for `professions` like '/is/NOUN ∧ job', among others.

4.4 Quality of the Classifier

This section compares the quality of the classifier generated using the labels identified by DARWIN. Figure 6e-6h and 10b show that DARWIN(HS) dominates other techniques over all the datasets. The active learning technique suffers from poor F-score initially and improves gradually. Since AL generates very few training examples, the trained classifier is highly unstable and shows jittery F-score. The KS approach shows similar performance and performs comparable to AL. On the other hand, DARWIN based pipelines are much more stable in terms of F-score. The classifier that was trained with the labeled data generated by DARWIN pipelines always maintains a high precision. It is interesting to note that [18] reports the maximum F-score for food intent to be 0.54, as compared to 0.84 by DARWIN. The classifier generated by DARWIN achieved an F-score above 0.8 for other intents like 'Travel' and 'Career' too while [18] reports a maximum F-score of 0.58 for these intents.

4.5 Additional Experiments

To provide better insights into DARWIN's performance, we have conducted a series of experiments to evaluate (1) how efficient the framework is in terms of the time required to obtain labels, (2) how much noise-aware models (trained by Snorkel) can improve the classification results, (3) how well human annotators approximate our notation oracles. Due to space limitations, we present the effect of varying seed rules and parameters in the full version.

Efficiency in Label Collection. As we demonstrated, DARWIN identifies majority of the positive instances in all the datasets using roughly 100 queries. The time taken to generate the index structure for all the datasets was less than 5 minutes. The hierarchy generation phase then iterates over the index to identify the candidate rules. This phase takes less 15 minutes for a corpus of 100K.

Since the `LocalSearch` algorithm does not require the index to be pre-computed and generates candidates on the fly, it runs in less than 45 minutes for all datasets. `HybridSearch` and `UniversalSearch` traversal algorithms require 60-90 minutes on smaller datasets (i.e., `directions`, `musicians` and `cause-effect`) and about 2 hour and 45 minutes on `professions`. The major bottleneck in this process is the time taken by the classifier to make a prediction for all instances in the corpus (It takes roughly 25 minutes for one round of training and testing on the `professions` dataset). We implemented a simple optimization where we evaluated a sentences only if it had a confidence score more than 0.3 in the previous iteration and only evaluated instances that didn't satisfy this constraint once every three iterations. This heuristic helped us reduce the running time from 2 hours and 45 minutes to 65 minutes for the `professions` dataset. The total running time does not grow linearly with the size of dataset because

	M	C	D	F
Darwin	0.91	0.79	0.89	0.87
Darwin+Snorkel	0.82	0.78	0.97	0.87

Table 2: Performance of Darwin with Snorkel (`M=musicians,C=cause-effect,D=directions,F=food-tweet`)

most of the components use the classifier to identify the positives. These candidate positives are used for hierarchy generation and traversal. Hence, the running time grows linearly with positive set size but not with dataset size (after using the above mentioned optimization).

Training noise-aware classifiers. One of the recent developments in weak-supervision paradigms has been emergence of frameworks such as *Snorkel* [12] which are designed to de-noise the generated labels and train noise-aware classifiers. In this experiment, we direct the set of rules identified by Darwin to Snorkel, and compare the quality of the noise-aware classifier against a classifier trained directly on the labels generated by Darwin. Table 2 summarizes the F-score that two classifiers have obtained on our datasets. We can observe that in most cases, using Snorkel does not yield any improvements. This is mainly because in many of these datasets, the rules generated by Darwin already exhibit a low degree of noise and a good coverage and thus there is almost no room for improvements. Nevertheless, we can see that on some datasets such as `directions` using Snorkel can be quite beneficial.

Performance of human annotators. Clearly, Darwin's performance heavily relies on the quality of responses it receives from the annotators. To study how well human annotators perform, we ran an experimental study on Figure-eight crowd-sourcing platform for `directions` dataset. Labels were collected for 2 600 heuristics. Each annotator was paid 2 cents per a single rule evaluation and three evaluations per rule were collected. A manual inspection of the results reveals that annotators were able to capture most of the precise heuristics such as 'best way to get there', 'shuttle from', 'across the street from', 'airport to hotel', and etc. Overall, we found less than 10 false positives responses in the 69 positive heuristics identified by the crowd labels. These erroneous responses were due to the fact that the 5 matching sentences presented to the annotator sometimes can have 3 or 4 positive instances by chance which confuses the annotators. Presenting more samples lowers the error rate. Interestingly, Darwin often rates these heuristics lower in preference to query as it can analyze the complete coverage set, and mitigate such errors by considering the entire distribution of instances. The annotators took 23 sec on average to label a heuristic query. For 100 queries, Darwin generates all the labels in less than 40 min of human effort. This time can further be reduced by asking various questions in parallel to different crowd members.

5. RELATED WORK

To the best of our knowledge, Darwin is the first system that assists annotators to discover rules under any desired rule grammar for rapid labeling of text data. Our work is related to studies in areas of weak supervision, crowdsourcing, and the intersection of the two which we discuss next.

Weak Supervision. There are multiple existing approaches for generating labels in weakly supervised settings. Some techniques rely on the notion of *distant supervision* where the labels are inferred using an external knowledge base [10, 1, 21]. One notable example is a system named Snuba [17] which generates labeling rules based on an existing labeled dataset. In contrast to these systems, Darwin is designed for scenarios where no additional sources of information are available. In such cases, it is necessary to rely on annotators to write labeling rules. While using expert-written rules have proven to be highly effective in many settings [12], there is limited work on how to facilitate the process of writing or discovering high-quality rules. One interesting example is Babble Labble[6], a labeling tool that allows annotators to explain (in natural language form) why they have assigned a label to a given data point. These explanations are then transformed into labeling rules. While Babble Labble simplifies the rule writing process, it only handles a single internal rule language. On the other hand, Darwin allows experts to pick their desired rule language depending on the complexity and the dynamics of the task at hand.

There have been several studies on utilizing the weakly-supervised labels in an optimal way. Snorkel [12] and Coral [16] are recent examples of systems (based on the data programming paradigm) that de-noise and utilize the labels collected via weak supervision. Similarly, there are numerous data management problems spanning data fusion [2, 13] and truth discovery [9], which focus on identifying reliable sources of data. Many recent studies in data integration have also explored techniques that handle error in crowd answers [3, 5]. Note that Darwin is a framework for discovering labeling rules which goes hand-in-hand with the aforementioned systems since Darwin's generated rules can be further processed using these de-noising techniques to achieve better results.

Crowdsourcing Frameworks. There has been many studies on devising oracle based abstractions that handle annotations from a crowd and minimize the noise in answers [20, 4]. Perhaps, more relevant to our work, are existing studies on how labeling rules can be verified with the help of the crowd. One recent example is a system named CrowdGame [22] which validates a rule by showing either the rule or its matching instances to the annotators. The authors demonstrate that their proposed game-based techniques yields the best results for rule verification. Unlike Darwin CrowdGrame assumes a pre-existing (manageable) set of possible rules from which the best rule should be selected. Darwin, on the other hand, has no such assumption and has to create a promising set of rules from the rule grammar. Additionally, the game-based approach to annotate a rule can be modeled as an Oracle in Darwin.

6. CONCLUSION

We present Darwin, an interactive end-to-end system that enables annotators to rapidly label text datasets by identifying precise labeling rules for the task at hand. Darwin compiles the semantic and syntactic patterns in the corpus to generate a set of candidate heuristics that are highly likely to capture the positives instances in the corpus. The set of candidate heuristics are organized into a hierarchy which enables Darwin to quickly determine which heuristic should be presented to the annotators for verification. Our experiments demonstrate the superior performance of Darwin in wide range of labeling tasks spanning intent classification, entity extraction and relationship extraction.

7. REFERENCES

[1] E. Alfonseca, K. Filippova, J.-Y. Delort, and G. Garrido. Pattern learning for relation extraction with a hierarchical topic model. In *ACL*, 2012.

[2] X. L. Dong and D. Srivastava. Big data integration. *Synthesis Lectures on Data Management*, 7(1), 2015.

[3] S. Galhotra, D. Firmani, B. Saha, and D. Srivastava. Robust entity resolution using random graphs. In *SIGMOD*, 2018.

[4] A. Ghosh, S. Kale, and P. McAfee. Who moderates the moderators?: crowdsourcing abuse detection in user-generated content. In *Proceedings of the 12th ACM conference on Electronic commerce*, 2011.

[5] A. Gruenheid, B. Nushi, T. Kraska, W. Gatterbauer, and D. Kossmann. Fault-tolerant entity resolution with the crowd. *arXiv preprint arXiv:1512.00537*, 2015.

[6] B. Hancock, P. Varma, S. Wang, M. Bringmann, P. Liang, and C. Ré. Training classifiers with natural language explanations. *arXiv preprint arXiv:1805.03818*, 2018.

[7] W. Hoeffding. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[8] Y. Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

[9] Y. Li, J. Gao, C. Meng, Q. Li, L. Su, B. Zhao, W. Fan, and J. Han. A survey on truth discovery. *ACM SIGKDD Explorations Newsletter*, 17(2), 2016.

[10] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL/IJCNLP*, 2009.

[11] S. Petrov, D. Das, and R. McDonald. A universal part-of-speech tagset. *arXiv preprint arXiv:1104.2086*, 2011.

[12] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *PVLDB*, 11(3), 2017.

[13] T. Rekatsinas, M. Joglekar, H. Garcia-Molina, A. Parameswaran, and C. Ré. Slimfast: Guaranteed results for data fusion and source reliability. In *SIGMOD*, 2017.

[14] B. Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2012.

[15] R. Socher, B. Huval, C. D. Manning, and A. Y. Ng. Semantic compositionality through recursive matrix-vector spaces. In *EMNLP-CoNLL*, 2012.

[16] P. Varma, B. D. He, P. Bajaj, N. Khandwala, I. Banerjee, D. L. Rubin, and C. Ré. Inferring generative model structure with static analysis. In *NIPS*, 2017.

[17] P. Varma and C. Ré. Snuba: Automating weak supervision to label training data. *PVLDB*, 2019.

[18] J. Wang, G. Cong, W. X. Zhao, and X. Li. Mining user intents in twitter: A semi-supervised approach to inferring intent categories for tweets. 2015.

[19] X. Wang, A. Feng, B. Golshan, A. Halevy, G. Mihaila, H. Oiwa, and W.-C. Tan. Scalable semantic querying of text. *PVLDB*, 11(9), 2018.

[20] P. Welinder, S. Branson, S. J. Belongie, and P. Perona. The multidimensional wisdom of crowds. In *NIPS*, 2010.

[21] F. Yang, Z. Yang, and W. W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *NIPS*, 2017.

[22] J. Yang, J. Fan, Z. Wei, G. Li, T. Liu, and X. Du. Cost-effective data annotation using game-based crowdsourcing. 2018.

APPENDIX

A. PROOF OF LEMMA 4

PROOF. Expected score of the heuristic function is

$$\begin{aligned} E[\sum_{s\in C_r} X_s] &= \sum_{s\in C_r} \mu_s \\ &= \sum_{s\in C_r\cap P} \mu_s + \sum_{s\in C_r\setminus P} \mu_s \\ &\leq \sum_{s\in C_r\cap P} (\beta + (1-\theta)(1-\beta)) \\ &\quad + \sum_{s\in C_r\setminus P} \left(\beta' + (1-\theta)(1-\beta')\right) \\ &= (\beta + (1-\theta)(1-\beta))\, p|C_r| \\ &\quad +(1-p)|C_r|\left(\beta' + (1-\theta)(1-\beta')\right) \\ &\leq (\beta + (1-\theta)(1-\beta))\,|C_r| \end{aligned}$$

□

B. PROOF OF LEMMA 5

PROOF. The score of heuristic function r is $\sum_{s\in C_r} X_s$. The expected value of the score is calculated in lemma 4. Using Hoeffding's inequality,

$$\begin{aligned} Pr[\frac{1}{|C_r|}\sum_{s\in C_r} X_s \leq (1+\epsilon)\mu_r/|C_r|] &\leq 2e^{-2\epsilon^2\mu_r^2/|C_r|} \\ &= 2e^{-2\epsilon^2(\beta+(1-\theta)(1-\beta))^2|C_r|} \\ &= 2e^{-4\log n} = \frac{2}{n^4} \end{aligned}$$

This shows that $\frac{1}{|C_r|}\sum_{s\in C_r} X_s$ is smaller than $(1+\epsilon)(\beta + (1-\theta)(1-\beta))|C_r|$ with a probability more than $1-\frac{2}{n^4}$ □

C. PROOF OF LEMMA 6

PROOF. Using Lemma 3 and 5, we can observe that the calculated benefit of heuristic h_1 is atleast $(1-\epsilon)\theta\beta'|C_{r_1}|$ with a probability of $1-\frac{2}{n^4}$. Similarly, the score of r_2 is atmost $(1+\epsilon)\left(\beta+(1-\theta)(1-\beta)\right)|C_{r_2}|$ with a probability of $1-\frac{2}{n^4}$. This shows that

$$score(r_1) > score(r_2) \quad (1)$$

$$(1-\epsilon)\theta\beta'|C_{r_1}| > (1+\epsilon)\left(\beta+(1-\theta)(1-\beta)\right)|C_{r_2}| \quad (2)$$

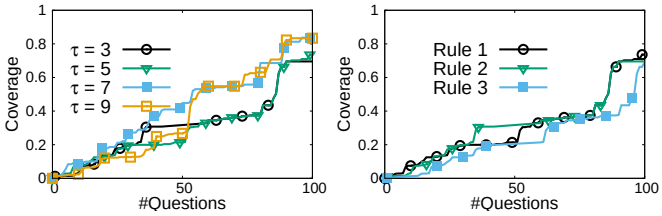
$$\frac{|C_{r_1}|}{|C_{r_2}|} > \frac{(1+\epsilon)\left(\beta+(1-\theta)(1-\beta)\right)}{(1-\epsilon)\theta\beta'} \quad (3)$$

$$\frac{|C_{r_1}|}{|C_{r_2}|} > \alpha \quad (4)$$

where α is a constant. □

D. ADDITIONAL EXPERIMENTS

Sensitivity to HybridSearch's traversal parameters. Here, we study to what extent DARWIN's performance is sensitive to parameter τ in the HybridSearchtraversal algorithm. Recall that parameter τ determines how often the HybridSearch algorithm switches between exploiting the local structure of the hierarchy as opposed to evaluating all possible candidates using the classifier. Figure 12a shows



(a) musicians (b) musicians

Figure 12: Sensitivity of DARWIN to τ and seed rules.

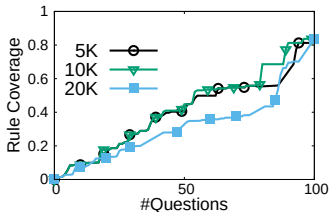


Figure 13: Sensitivity of DARWIN(HS) to the number of candidates generated.

that DARWIN(HS) performs very similar on varying τ. The solution quality tends to improve slightly on increasing τ because the effective rules for the musicians dataset are not close to each other in the hierarchy. However, note that choosing large values of τ can affect the efficiency of the pipeline. More precisely, large values of τ force the HybridSearch system to rely on its internal classifier to evaluate all rule candidates for too many steps which can be quite time consuming.

Sensitivity to seed rule. This experiment establishes that DARWIN has a robust performance given different types of input seed rule. Focusing on the musicians dataset, we initialize DARWIN with the following seed rules. Rule 1 is the keyword '*composer*' stating that any sentence containing this word mentions a musician. Rule 2 is the keyword '*piano*' and finally Rule 3 is the sentence '*Beethoven taught piano to the daughters of Hungarian Countess Anna Brunsvik.*'. Note that Rule 2 is an extremely generalized version of Rule 3. Figure 12b compares the performance of DARWIN(HS) for all three seed rules. DARWIN performs equally well on three different types of input seed rules. We can observe that for Rule 3, DARWIN requires the initial 8 queries to generalize the seed rule, and as soon as it identifies a rule with high coverage, it performs very similar to the other seed rules.

Sensitivity to number of generated candidates. One of the parameters of the DARWIN framework is the number of candidates that gets generated by the candidate-rule generation component. In our experiments, DARWIN generates 10K candidates rules with high coverage and organizes them into an index. The goal is to make sure the set of generated candidates contain some (if not all) of the precise rules. Choosing a large value for the index size would satisfy this objective but affects the efficiency and increases the number of candidates that UniversalSearchand HybridSearch algorithms need to consider. We observed that generating 10K candidates per iteration helped DARWIN identify precise candidate rules. Figure 13 shows that the performance of DARWIN(HS) algorithm is consistently similar for different number of candidate rules generated.

Sensitivity to classifier quality. Figure 14 compares the performance of HybridSearch strategy on musicians

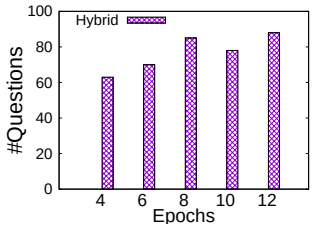
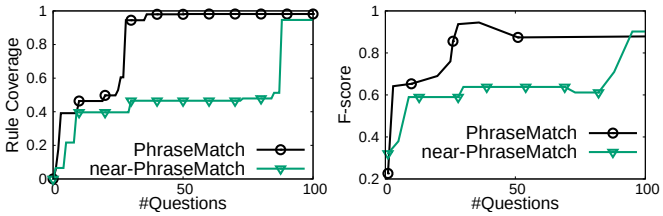


Figure 14: Effect of classifier quality on DARWIN(HS) (on `musicians` dataset)



(a) `directions` (b) `directions`

Figure 15: Comparison of DARWIN(HS) on two different but very similar grammars.

dataset by varying the number of epochs for which the neural network classifier is trained. With more epochs, the classifier tends to overfit more to the training data. We measure the number of questions from DARWIN(HS) to the oracle in order to label at least 75% of the positive sentences. It is evident that the DARWIN performance is robust to change in behavior of the classifier.

Minor Grammar variations. We experiment with a variant of PhraseMatch grammar which allows for the tokens to appear close to each other (within a distance of 2) and not necessarily in a sequence. For instance, 'way hotel' is a valid rule which satisfies the sentence 'what is the best way to our hotel?'. Fig 15 compares the performance of DARWIN(HS) on these two different grammars. It is evident that DARWIN performs slightly worse on near-PhraseMatch grammar. This is due to the large number of noisy candidate rules that have high coverage. The experiment on this grammar leveraged the same index as that of PhraseMatch grammar. The only difference was in the coverage set of the rules and the edges in the hierarchy. This validates DARWIN's flexibility to handle different types of grammar with minimal modifications.

Comparing Rule Grammars In this experiment, we take a deeper look at DARWIN's performance using the two grammars separately. Figure 16 compares the performance of the DARWIN(HS) algorithm over PhraseMatch grammar and TreeMatch grammar for the `professions` dataset. Clearly, the TreeMatch grammar shows superior performance as compared to PhraseMatch grammar on this dataset.

We explain one of the the reasons behind poor performance of PhraseMatch grammar with an example. Suppose the algorithm starts with '*mom is the best job*' as the seed rule[15] and generalizes it to '*is the best job*'. However, '*is the best job*' is labeled as a noisy rule as positive instances have a NOUN preceding this phrase, while this phrase also captures sentences such as 'what is the best job I can get?' and 'doing what you love is the best job.'. Because the rule

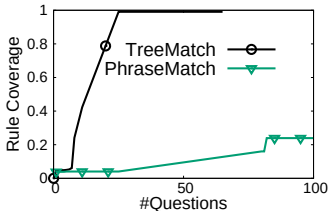


Figure 16: Comparings performance of rule grammars (on `professions` dataset).

is too noisy, the `HybridSearch` algorithm attempts to specialize this rule to identify various prefixes to this phrase. However, since there are many possible phrases that can be generated by adding a prefix to this phrase (e.g., '*teaching is the best job*', '*coding is the best job*', and so on), each such rule tends to have a low coverage. As a result, using the PhraseMatch grammar DARWIN struggles to identify a small set of highly precise rules with a high coverage.

In contrast, the TreeMatch grammar can capture the set of positives succinctly with a rule such as '/is/NOUN ∧ job', which is one of the ancestors of '*mom is the best job*' in the rule hierarchy. This example demonstrates the usefulness of TreeMatch grammars to capture the complex relationships of tokens in sentences. The expressiveness of the TreeMatch grammar can concisely describe rules that are difficult to capture with PhraseMatch grammar. On the other hand, the PhraseMatch grammar performs better compared to the TreeMatch grammar on the `musicians` dataset as most of the sentences are very coherent and the precise rules are very short and straightforward. '*orchestra*', '*piano*', and '*nyman band*' are examples of such rules.

[15]This is a real example from ClueWeb.