Adaptive Rule Discovery for Labeling Text Data

Sainyam Galhotra[*]
University of Massachusetts Amherst
sainyam@cs.umass.edu

Behzad Golshan
Megagon Labs
behzad@megagon.ai

Wang-Chiew Tan[*]
Facebook AI
wangchiew@fb.com

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.

ACM Reference Format:
Sainyam Galhotra, Behzad Golshan, and Wang-Chiew Tan. 2021. Adaptive Rule Discovery for Labeling Text Data. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3448016.3457334

1 INTRODUCTION

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 [7, 9, 20].

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 improving the performance of machine learning models [15, 25]. However, obtaining high-quality labeling rules remains a challenging problem. A subset of existing frameworks, with *Snorkel* [25] being the most notable example, rely on domain experts to provide a set of labeling rules which can be a tedious and time-consuming task. In contrast, other frameworks aim to automatically mine useful rules using further supervision. For instance, Snuba [32] circumvents dependence on domain experts by requiring a labeled subset of the data, and then utilizing it to automatically derive labeling rules. *Babble labble* is another example which asks expert to label a few examples and explain their choice. This explanation is used to derive labeling rules. While these approaches have been quite effective in certain settings, we elicit their limitations with the following real-world example.

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 rules for such tasks is a common approach but it has a number of shortcomings:

- It is **time consuming**. Annotators must be familiar with the rule language (e.g., Stanford's Tregex). 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 useful in Example 1. However, annotators may miss it or may not 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. Hence, the number of distinct labels obtained does not always grow linearly with the number of annotators, which is rather inefficient.

[*]Work done while at Megagon Labs.

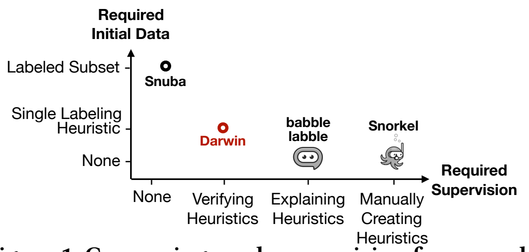


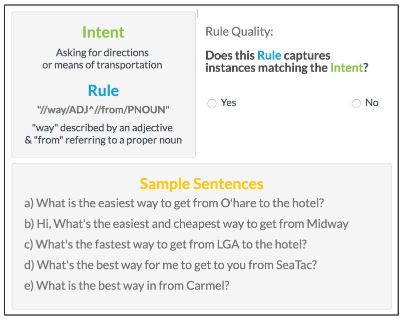
Figure 1: Comparing weak-supervision frameworks

The alternative approach would be to automatically mine useful rules 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.

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 as we show in our experiments, a single labeling rule (or a couple of labeled instances) would be sufficient for Darwin. Compared to Snorkel and Babble labble, Darwin requires less supervision by domain experts. More explicitly, Darwin requires experts to simply verify the suggested rules while Snorkel requires them to manually write rules and Babble labble requires them to provide explanations for why a particular label is assigned to a data point.

Given a corpus and a seed labeling rule, Darwin identifies a set of promising candidate rules.Th e best candidate 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.Th e 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.Th is 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.Th erefore, 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. In fact, we show that verifying 50 rules suggested by Darwin is enough to achieve a F1-score of 0.80. Furthermore, we present theoretical results on approximation guarantees of Darwin.
- Darwin summarizes the rules, shows samples from the coverage setand 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,fi nd sentences that mention particular entity types, and identify sentences that describe certain relationships between entities (i.e., relation extraction).


Intent

Asking for directions
or means of transportation

Rule

"//way/ADJ^//from/PNOUN"

"way" described by an adjective
& "from" referring to a proper noun

Rule Quality:

Does this Rule captures instances matching the Intent?

Yes No

Sample Sentences

a) What is the easiest way to get from O'hare to the hotel?

b) Hi, What's the easiest and cheapest way to get from Midway

c) What's the fastest way to get from LGA to the hotel?

d) What's the best way for me to get to you from SeaTac?

e) What is the best way in from Carmel?

Figure 2: Sample query to annotators.

2 PRELIMINARIES & PROBLEM DEFINITION

In a nutshell, Darwin takes as input an unlabeled corpus of sentences along with an initial seed labeling rule (which is assumed to generate at least two positive instances). Darwin then identifies promising candidate labeling rules leverages an oracle to verify whether a particular candidate rule is effective at capturing positive instances or not. Finally, the set of discovered rules are forwarded to Snorkel [25][1] to train a high precision classifier. Before describing Darwin's rule discovery pipeline in detail, we provide a formal definition of labeling rules along with a description of an oracle.

Rule search space. Naturally, labeling rules can be of different types with distinct semantics. For example, a rule may check for certain phrases in a sentence [32] or it may enforce some conditions on the parse tree [34]. In Darwin, the space of possible rules are specified using a collection of *Rule Grammars*, where each grammar describes a particular type of labeling rules.Th ese concepts are formally defined as follows.

Definition 1 (RuleGrammar). *A Rule Grammar $\mathcal{G}$ is a Context-Free Grammar (CFG) [19]. A CFG consists of derivation rules that can be applied regardless of the context of a non-terminal symbol.*

For a rule grammar $\mathcal{G}$, we now define labeling rules.

Definition 2 (LabelingRules). *A labeling rule r is a derivation of the grammar $\mathcal{G}$. We use C_r to denote the set of sentences that satisfy the rule r, and refer to $|C_r|$ as it's* coverage.*Th e accuracy of a rule r is defined as the fraction of positive instances in the coverage C_r.*

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

Example 2 (TokensRegexGrammar). *Let $\mathcal{V}$ denote the set of all possible words.Th e regular expression grammar on the tokens comprises of the following derivation rules.*

$$\texttt{A} \longrightarrow v\texttt{A} \ \forall\ v \in \mathcal{V}, \qquad \texttt{A} \longrightarrow \texttt{A+A}$$
$$\texttt{A} \longrightarrow \texttt{A*A}, \qquad \texttt{A} \longrightarrow \epsilon$$

This is a CFG that generates regular expression of words as a candidate labeling rule. For example, it generates rules such as 'best way to' as well as less meaningful rules such as 'shuttle is airport' for the task described in Example 1. A sentence satisfies the rule if it contains that phrase.Th e sentences s1, s3 and s6 in Example 1 satisfy the rule r = 'best way to', hence $C_h = \{s1, s3, s6\}$.

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

While TokensRegex captures lexical patterns and phrases, it fails to capture syntactic patterns over parse trees. TreeMatch grammar captures these to identify more complex and generic rules.

Definition 3 (TreeMatchGrammar). *Let $\mathcal{V}$ denote the set of terminals comprising of all the tokens and Part-of-Speech (POS) tags [23] present in the corpus. E.g., NOUN, VERB, etc.*
Derivation Rules: *The grammar has three fundamental operations that make up a rule, namely And (∧), 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:*

$\mathtt{A} \rightarrow \mathtt{/A}$, $\quad \mathtt{A} \rightarrow \mathtt{A \wedge A}$

$\mathtt{A} \rightarrow \mathtt{//A}$, $\quad \mathtt{A} \rightarrow \mathtt{v} \ (\forall \ v \in \mathcal{V})$

As a default setting, Darwin comprises of two different grammars (a) TokensRegex (b) TreeMatch, with the ability to plug in more rule grammars as long as they are context-free. It is important to mention that the complexity of rules 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 using the notion of Oracles as follows.

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

A rule r is considered precise only if it has accuracy > 0.5, i.e. r labels the subset correctly with a probability better than random. Such labelling rules have been shown to be sufficient to train a highly precise classifier in prior literature [25]. We further performed a user study to evaluate the effectiveness of annotators to identify rules that are useful heuristics for the task at hand (User study paragraph). We observed that annotators identified all rules with accuracy more than 0.80 as useful for prediction. Note that many prior techniques have studied the optimization of the accuracy of oracle answers [13, 35]. All such techniques have been abstracted under the notion of an oracle and are orthogonal to this work.

Problem statement. We are now ready to formally define our problem. Given a labeling task, our goal is to find a set R of adequately accurate labeling rules such that the union of their coverage denoted as $P = \bigcup_{r \in R} C_r$, would have a high recall (i.e., to contain majority of the positive instances in the corpus) and all rules $r \in R$ are accurate. We would like to maximize the recall of set P without posing too many queries to the oracle. More details on our experimental setup and labelling accuracy are delegated to [12]. In this work, we do not aim to maximize the accuracy of the identified rules, for which we can also rely on various de-noising techniques from the weak supervision literature [25].

Problem 1 (MaximizeRulesCoverage). *Given a corpus S, a seed labeling rule r_0, an oracle O, and a budget b, find a set R of labeling rules using at most b oracle queries, such that the recall of set P, i.e., the union of the coverage of rules in R, is maximized and all rules $r \in R$ have accuracy > θ.*

Lemma 1. *The Maximize Rules Coverage problem is NP-hard.*

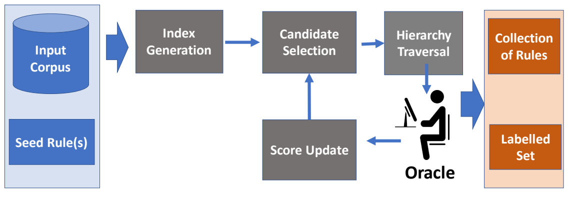


Figure 3: Darwin's architecture.

Algorithm 1 Darwin

```
Input: Input Corpus $\mathcal{S}$, seed rule $r_0$, budget $b$
Output: Collection of rules $R$, Positive instances $P$, Classifier $C$
1: $\mathcal{I} \leftarrow$ generate_index($\mathcal{S}$), $P \leftarrow$ coverage($r_0$)
2: $Q \leftarrow \phi$, $(C,P') \leftarrow$ train_classifier($P$, $\{r_0\}$, $\mathcal{S}$)
3: while $|Q| \leq b$ do
4:     $\mathcal{H} \leftarrow$ generate_hierarchy($\mathcal{S}, P', \mathcal{I}$)
5:     $q \leftarrow$ traversal $(H, P, Q, C)$
6:     if OracleResponse (q) then
7:         $P \leftarrow P \cup$ coverage($q$)
8:         $(C,P'') \leftarrow$ train_classifier($R,P,\mathcal{S}$)
9:     $P' \leftarrow P'' \setminus P'$, $Q \leftarrow Q \cup \{q\}$
10:    $\mathcal{H} \leftarrow$ update_scores($\mathcal{H}$)
11: return $R, P, C$
```

We present proofs and other details in the technical report [12].

User Study. Darwin's performance relies on the quality of responses it receives from the annotators. To study how well human annotators perform, we ran a user study on Appen platform [1] for `directions` dataset (Example 1). Labels were collected for 2 600 rules generated using the PhraseMatch grammar. Each annotator was shown a rule along with a sample of five sentences that satisfy the rule. The annotators were asked to decide if the rule was: (a) Very accurate (b) Somewhat Accurate (c) Confusing (d) Unhelpful. We consider choosing options (a) and (b) as a YES response, and options (c) and (d) as a NO response. Each annotator was paid 2 cents per a single rule evaluation and three evaluations per rule were collected. Each annotator is shown a different sample of sentences from the coverage set. A manual inspection of the results reveals that annotators were able to capture most of the accurate rules 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 positive responses in the 69 positive rules identified by the crowd. These erroneous responses had low agreement between different annotators and the mistakes were due to the fact that the five matching sentences presented to the annotator sometimes can have 3 or 4 positive instances by chance. Presenting more samples lowers the error rate. (no false positive with 10 samples). Interestingly, Darwin rates these rules lower in preference to query as it can analyze the complete coverage set, and mitigate such errors by considering the entire distribution of instances.

3 THE DARWIN SYSTEM

In this section, we describe the architecture of Darwin which is illustrated in Figure 3. 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 it is refined with evolving training data. In order to identify new rules Darwin leverages the following properties. (i) The generalizability of the trained classifier guides the search towards semantically similar rules. For example, on identifying the importance of 'bus' as

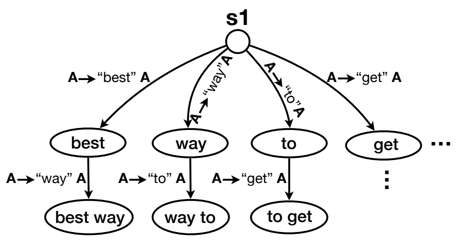


Figure 4: Examples derivation sketch.

a rule, Darwin identifies 'public transport' as another possibility due to their related semantics.(ii) Local structural changes to the already identified rules helps identify new rules eg. consider 'What is the best way to the hotel?' as input seed sentence, Darwin constructs local modifications by dropping and adding tokens and identifying a new rule 'shuttle to the hotel'. Darwin leverages these intuitions to 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 crucial for efficient execution of Darwin. Darwin organizes all candidate rules in the form of a hierarchy that captures subset/-superset relationship among rules (similar to the block hierarchy in [11]). Rules 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 data structure is that if a rule r is identified to capture positives, then any of its subset (descendant in the hierarchy) does not capture any new positive. Additionally, it is helpful for efficient execution of local structural changes to any rule.

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 rule grammar, and one (or more) seed labeling function(s). Alternatively, a set of positive instances can be provided instead of seed labeling rules.Th e output of Darwin is the set of generated rules, the positive instances that are discovered, and a classifier that is trained using the labeled data.

In each iteration, Darwin performs the following steps. First, the *Candidate Generation* component generates a small set of promising candidate rules (from the space of all possible rules), and organizes them in the form of a hierarchy $\mathcal{H}$ (line 4) with the most generic rule at the top and the stricter ones at the bottom. Once the hierarchy is built, the *Hierarchy Traversal* component carefully navigates and evaluates the rules in the hierarchy tofi nd the best candidate (line 5).Th is candidate is then presented to the annotator (line 6) and scores are updated. Finally, the updated classifier and rule scores are sent back to hierarchy generation and 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 (similar to a trie) to provide fast access to sentences that satisfy certain rules. Given a collection of rule grammar $\{G_1, \ldots, G_t\}$ and a sentence s, one can enumerate the set of all possible rules of G_i, generated using afi xed number of derivation rules[2], that s satisfies. For example, using the TokensRegex grammar, all rules that a sentence s satisfies is the set of all regular expressions that correspond to s. We organize these rules into a structure called *Derivation Sketch*, which summarizes the derivations of all rules that match s. Figure 4 shows (parts of) the derivation sketch for sentence $s1$ from Example 1.

[2]The derivation sketch for the TreeMatch grammar is the dependency parse tree of the sentence [34].

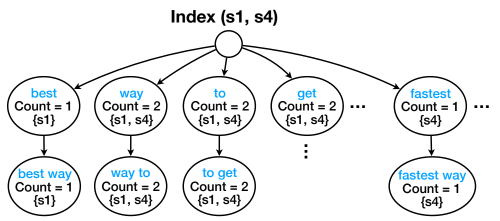


Figure 5: An example of index creation process

Algorithm 2 Candidate-rules Generation

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

An index $\mathcal{I}$ is a compact representation of all rules that are satisfied by at least one sentence in the corpus. Each node in $\mathcal{I}$ represents a rule 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 rule.

The index $\mathcal{I}$ is created by merging the derivation sketch of sentences, one at a time.Th e index is initialized with the derivation sketch of thefi rst sentence.Th ereafter, for every new sentence s, the sketch of s is merged into $\mathcal{I}$ as follows.Th e root node of the sketch and $\mathcal{I}$ are merged, and then all nodes (starting form the merged root) are considered in a breadth-first fashion;Th e children of the node under consideration which are derived using the same derivation rule are merged. 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 5 shows the index built from derivation sketches of $s1$ and $s4$.The index construction has run time linear in the number of sentences.

3.2 Rule-Hierarchy Generation

The number of possible rules under a given grammar $\mathcal{G}$ is often exponential in the dictionary size.Th e hierarchy generation component enumerates a manageable set of promising candidate rules and organizes the generated candidate rules in a hierarchy that captures the subset/superset relationship between them. Specifically, the hierarchy generation process consists of the following steps. First, the *Candidate Generation* step uses the index $\mathcal{I}$ to generate a subset of rules that have high coverage over the set of positive instances discovered so far.Th is algorithm operates in a greedy best-first search mechanism to identify valuable candidates.These rules 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.

Candidate Generation. Algorithm 2 generates candidate rules by exploiting the following property of the index. For a given node in

Algorithm 3 LocalSearch Traversal

```
Input: Rule hierarchy $\mathcal{H}$, Seed rule $r_0$
Output: Collection of positive instances $P$, Collection of rules $R$
1:  $QueryCount \leftarrow 0$
2:  $R \leftarrow \{r_0\},\ P \leftarrow C_{r_0},\ \ C \leftarrow$ TrainClassifier($P$)
3:  $localCandidates \leftarrow \{r_0\}$
4:  while $QueryCount < b$ do
5:      $r \leftarrow$ GetMostBeneficialCandidateRule($localCandidates, C$)
6:      $QueryCount \leftarrow QueryCount + 1$
7:      if OracleResponse($r$) is YES then
8:          $R \leftarrow R \cup r,\ \ P \leftarrow P \cup C_r, C \leftarrow$ TrainClassifier($P$)
9:          $localCandidates \leftarrow (localCandidates \setminus \{r\}) \cup$ Parents($r$)
10:     else
11:         $localCandidates \leftarrow (localCandidates \setminus \{r\}) \cup$ Children($r$)
12: return $P, R$
```

the index, descendant nodes correspond to stricter rules and hence their coverage is less than the coverage of any of its ancestors.The set of candidates is initialized with rule '*' which refers to the root of index $\mathcal{I}$ and matches all possible sentences in the corpus. In each iteration, the algorithm adds the children of the previous iteration's best candidate rule to the candidate list (line 3).Th e candidates are then sorted in decreasing order of coverage over the set P (line 4).Th e candidate with the highest coverage is removed from the candidate list and appended to list offi nal results R (lines 6-7). This process is repeated until there are k rules in R. Note that the time complexity of this greedy algorithm is linear in the number of candidates generated.

3.3 Hierarchy Traversal

This module determines which rule in the hierarchy is the best rule to be submitted to the oracle. We present three traversal techniques: LocalSearch, UniversalSearch, and HybridSearch. At a high level, LocalSearch relies on the hierarchy structure to select the next best candidate from the neighborhood of rules verified by the oracle in the past. In contrast, UniversalSearch ignores locality constraints and selects the rule with maximum benefit globally.

Finally, the HybridSearch traversal combines thefi rst two techniques tofi nd the next best rule.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 rule to be sent to the oracle is based on how *beneficial* the rule is, which we elaborate next.

Benefit of a rule (r) is the expected gain in the positives 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 prediction probability of sentence s being a positive. The ground truth values denote $p_s = 1$ for positive sentences and $p_s = 0$ for negative ones. However, due to lack of ground truth labels, Darwin estimates these probability values by training a classifier using the set of positive instances discovered so far and sampling random instances from the corpus as negatives.Th e probability estimates improve as the system iteratively discovers more rules 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 already identified useful rules by the oracle to identify the next best rule for querying. Specifically, it maintains a set of candidate rules, and selects the most beneficial rule r from the candidates (line 3). If the oracle confirms that r is accurate (line 7), then it adds r's parents into the candidate set as they are generalizations of r (line 9) and might capture more positive instances. However, if the oracle labels r as a noisy rule, LocalSearch adds the children of r to the candidate set (line11) with the hope that a specialized version of r might be less noisy.

LocalSearch is efficient at utilizing the structure of the hierarchy tofi nd promising rules to query the oracle. Since it 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 run for. However, LocalSearch may require many traversal steps in cases where the initial seed rule is quite different from other accurate rules the system aims to discover. Also, it does not exploit the similarity and overlap between the coverage sets of different rules. We now present, UniversalSearch that addresses these shortcomings by utilizing a holistic view of the hierarchy.

Algorithm 4 UniversalSearch Traversal

```
Input: Rule hierarchy $\mathcal{H}$, Seed rules $r_0$
Output: Collection of positive instances $P$, Collection of rules $R$
1:  $QueryCount \leftarrow 0$
2:  $R \leftarrow \{r_0\},\ P \leftarrow C_{r_0},\ \ C \leftarrow$ TrainClassifier($P$)
3:  $universalCandidates \leftarrow \{r : r \in \mathcal{H}\}$
4:  $C \leftarrow$ TrainClassifier($P$)
5:  while $QueryCount < b$ do
6:      $r \leftarrow$ GetMostBeneficialCandidate($universalCandidates, C$)
7:      $QueryCount \leftarrow QueryCount + 1$
8:      if$(\sum_{s \in C_r} p_s)/|C_r| \leq 0.5$ then continue
9:      if OracleResponse($r$) is YES then
10:         $R \leftarrow \{r_0\},\ P \leftarrow C_{r_0}, C \leftarrow$ TrainClassifier($P$)
11:     $universalCandidates \leftarrow universalCandidates \setminus \{r\}$
12: return $P, R$
```

3.5 UniversalSearch

The UniversalSearch algorithm (see Algorithm 4) evaluates all rules in the hierarchy (line 3) as candidates and iteratively query the best rules based on their benefit (lines 5-11). In each iteration, UniversalSearch omits any rule for which the probability of a randomly selected instance from its coverage C_r being negative is more than positive (line 8).Th is condition is evaluated by checking if the expected probability of a sentence in C_r is more likely to be a negative (≤ 0.5). Among the remaining rules, it chooses the rule with maximum benefit (line 6) to query the oracle. Based on oracle's feedback, it re-trains the classifier if new positives were discovered (line 10) or else it continues to query the next best rule.

The strength of UniversalSearch is in its capability to identify semantic similarity between rules and their matching instances. Under reasonable assumptions of the trained classifier, we prove optimality of UniversalSearch to identify rules [12].

Theorem1. *In worst case,* UniversalSearch *provides a constant approximation of Problem 1 with a probability of* $1 - o(1)$

Even though UniversalSearch is proven to achieve constant approximation of the optimal solution, it has the following shortcomings: (1) compared to LocalSearch, it is inefficient as it iterates over all rules 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 accurate rules. In such cases, UniversalSearch fails to exploit the structure of the hierarchy to at leastfi nd structurally similar rules.However, LocalSearch performs such local generalizations of the identified rules. We describe the HybridSearch algorithm next, which combines the strengths of UniversalSearch and LocalSearch.

3.6 HybridSearch

HybridSearch combines the two previous traversal techniques by maintaining lists of local and universal candidates, and imitating the strategy of the both traversal algorithms. Starting from the UniversalSearch strategy, the HybridSearch algorithm queries candidate rules in non-increasing order of benefit to the oracle. If the algorithm fails tofi nd a accurate rule within afi xed number of attempts (τ), then it switches to the LocalSearch strategy. Similarly, if the LocalSearch strategy has no success within afi xed number of attempts, the traversal toggles to the UniversalSearch strategy.Th e switch between the two strategies is decided based by a parameter τ (by default 5) which denotes the number of unsuccessful attempts before the switch. Clearly, higher values of τ discourage switching between the two strategies. In summary, if the trained classifier is noisy (due to lack of positive instances), HybridSearch exploits the structure of the hierarchy to search for rules. Similarly when no accurate rules are found by LocalSearch, it uses UniversalSearch's ability to generalize to other rules.

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 rules 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 rules to be added to the hierarchy.

4 EXPERIMENTS

In this section, we perform empirical evaluation of Darwin along with other baselines to validate the following (For more experiments, please refer to the full version [12]).

- The ability of Darwin to identify majority of the positives, with high precision 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) within 100 queries.
- Darwin is efficient and can generate labels from a corpus of 1M sentences in less than 3 hrs. Darwin's performance is resilient to variations in the seed set.

4.1 Experimental Setup

Datasets. We experimented withfi ve diverse real-world datasets each suitable for one of the following NLP tasks: entity extraction, relationship extraction, and intent classification. All datasets, except for directions, come with ground-truth labels which we use for evaluation and to synthesize the oracle responses. For the directions dataset, we rely on human annotators to generate the gold standard and validate rules.

- cause-effect [30] (10.7K sentences with 12% positives) is a dataset commonly used as a benchmark for extracting cause and effect relationship between two entities. All sentences that contain cause-effect relation between entities are labelled positives.
- tweets [33] dataset (2130 sentences, 11.4% Food tweets) is a benchmark for classifying the intent of tweets into categories such as food, travel and career, etc.
- directions (15.3K sentences, 3.8% positives) dataset is described in Example 1 used to identify directions related sentences.
- musicians dataset (1.5M sentences, 0.1% positives) consists of sentences from Wikipedia articles and the task is to extract the names of musicians.Th e ground-truth is obtained with NELL's knowledge-base [4].
- professions dataset (1M sentences, 1.1% positives) is a collection of sentences from ClueWeb [2].Th e sentences that mention professions are positives.Th e ground truth is generated using NELL's knowledge-base [4].

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 effectiveness of Darwin() and Snuba [32].
- 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 accuracy[3].
- Section 4.4 compares the F-score of the classifier learnt over Darwin's labelled dataset with an *Active Learning* (AL) [28] and a *Keyword Sampling* (KS) technique and the HighP baseline. AL selects the instance with the highest entropy and asks 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 byfi ltering the corpus using relevant keywords, and label the instances in the smaller set. To do so, we asked annotators to provide 10 distinct keywords as a rule to filter the dataset.The KS technique randomly samples instances from thefi ltered dataset and asks for its label. We employ the same deep learning based classifier for all the techniques.

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 algorithms in Python.Th e dependency parse trees are generated with SpaCy [5]. All classifiers are implemented with a 3-layer convolutional neural network followed by two fully connected layers, following the architecture by Kim et al. [17].Th e input to the classifier is a matrix created by stacking word-embedding vectors [3] of the words in the sentence. For generating derivation sketches, the maximum depth is set to 10 and we consider 10K rules in candidate selection. We ran user study over Appen [1] to label rules for directions and tweets datasets. For other datasets, when simulating the responses from an oracle, we respond YES to rule r if its accuracy is more than 0.80.

4.2 Comparison with Snuba

In this experiment, we initialize Snuba and Darwin(HS) with the same set of randomly chosen labeled sentences and compare the

[3]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.

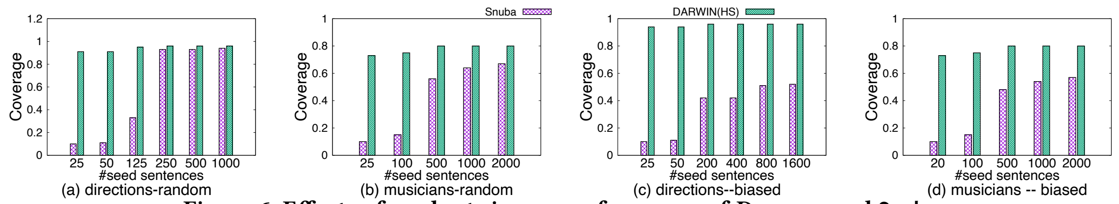


Figure 6: Effects of seed set size on performance of Darwin and Snuba.

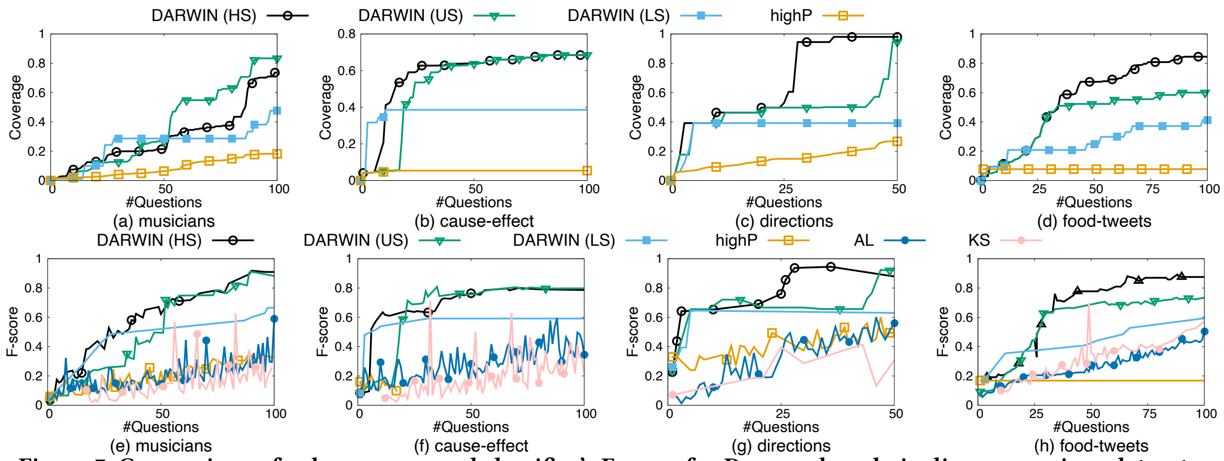


Figure 7: Comparison of rule coverage and classifier's F-score for Darwin based pipelines on various datasets.

coverage of positives identified by each of the techniques[4]. Figures 6a, 6b shows the change in coverage by varying the initial seed set size. Darwin(HS) is able to identify majority of positives even when the pipeline is initialized with less than 25 sentences. However, Snuba requires at least 200 sentences for `directions` and 1000 for `musicians`. All rules identified by Darwin(HS) have accuracy more than 0.80 for both datasets (across all seed inputs).Th e rules identified by Snuba have average accuracy of 0.72 in `directions` and 0.68 in `musicians` for 500 seed sentences. We observe a similar gap in accuracy of identified rules for other seed inputs too.

To further evaluate the generalizability of Snuba and Darwin(HS) to identify rules 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 6c, 6d 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 in two datasets. Darwin(HS) 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` due to the presence of many diverse rules 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.

This experiment validates that Snuba works well when the initial seed set has enough randomly chosen positives and does not generalize to rules that have limited evidence. On the other hand, Darwin(HS) identifies majority of the positives even when the pipeline is initialized with just 25 sentences and has good generalizability. To further evaluate Darwin, the following subsection considers a more challenging scenario where the pipeline is initialized with a single labeling rule or just two positive sentences.

[4]In this experiment, we do not start with a single labeling rule as Snuba achieves a very small coverage as it fails to obtain enough positive instance due to the high degree of imbalance in these datasets.

4.3 Rule Coverage

Figures 7a-7d and 8a illustrate the fraction of positives identified by Darwin and our baselines. 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. Figure 7d demonstrates the behavior for 'Food' intent in the tweets. We observed similar behavior for 'Travel' and 'Career' intents. We can observe that the other baselines do not perform well compared to Darwin; `highP` identifies rules with low coverage as candidates. Also note that the Darwin(LS) algorithm shows a high progressive coverage initially but converges to a 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 practical purposes 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 well when useful rules are present close to each other in the hierarchy and Darwin(US) performs better in the presence of abundant labelled examples.

Figure 9 shows some rules which are queried by the Darwin(HS) algorithm. In the `directions` example, 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.Th e 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

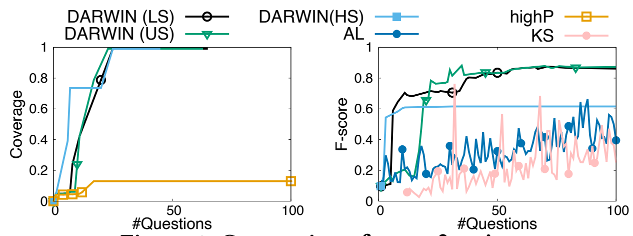


Figure 8: Comparison for `profession`.

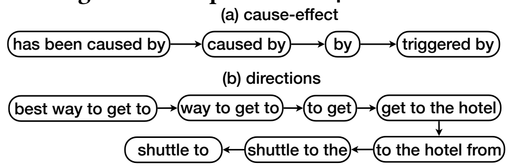


Figure 9: Example of traversals by `HybridSearch` algorithm.

relatively simple as the algorithm generalizes the initial rulefirst and as soon as it reaches the noisy and unhelpful rule '*by*', it again specializes to '*triggered by*' which is again accurate.

4.4Qu ality of the Classifier

This section compares the quality of the classifier generated using the labels identified by DARWIN. Figure 7e-7h and 8b 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.Th e classifier that was trained with the labeled data generated by DARWIN pipelines always maintains a high precision. It is interesting to note that [33] reports the maximum F-score for food intent to be 0.54, as compared to 0.84 by DARWIN.Th e classifier generated by DARWIN achieved an F-score above 0.8 for other intents like 'Travel' and 'Career' too while [33] reports a maximum F-score of 0.58 for these intents.

Efficiency in Label Collection. In terms of running time, DARWIN takes less than 5 minutes to generate the index structure for all datasets.Th e hierarchy generation phase takes less 15 minutes for a corpus of 100K. `HybridSearch` traversal algorithm requires 60-90 minutes on smaller datasets (i.e., `directions`, and `cause-effect`) and about 3 hours on `musicians` and `professions`.Th e major bottleneck in this process is the time taken by the classifier to make a prediction for all instances in the corpus (It takes 25 minutes for one round of training and testing on the `professions` dataset). We implemented a simple optimization where a sentences is evaluated only if it has 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.Th is rule helped reduce the running time from 2 hours and 45 minutes to 65 minutes for `professions`.

5 RELATED WORK

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

Weak Supervision. There are multiple existing approaches for generating weakly supervised labels. Some techniques rely on the notion of *distant supervision* where the labels are inferred using an external knowledge base [6, 21, 36]. One notable example is Snuba [32], which generates labeling rules based on an existing labeled dataset. In contrast, DARWIN is designed for scenarios where no additional source of information is available. In such cases, it is necessary to rely on annotators to write labeling rules. While using rules have proven to be highly effective in many settings [25], there is limited work on how to facilitate the process of discovering high-quality rules. One example is Babble Labble[15], a tool that allows annotators to explain (in natural language) why they have assigned a label to a given data point, which 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 using weakly-supervised labels in an optimal way. Snorkel [25] and Coral [31] are recent examples of systems (based on the data programming paradigm) that de-noise and utilize these labels. Similarly, there are numerous data management problems spanning data fusion [8, 26] and truth discovery [18], which focus on identifying reliable sources of data. Many recent studies in data integration have also explored techniques to handle crowd error [10, 14]. Note that DARWIN is a framework for discovering labeling rules which goes hand-in-hand with these systems as DARWIN's generated rules can be further processed using these de-noising techniques to achieve better results.

Information extraction from text. Prior literature has extensively explored pattern techniques for extracting structured information from unstructured text. Pattern based mining techniques like Hearst patterns [16, 22, 27, 29], NELL [4] have been widely used to understand the expression of language and semantic relationships between entities.Th ese techniques are orthogonal to our study and our framework can be extended to explore different types of patterns by modifying the rule grammar.

Crowdsourcing Frameworks. There have been many studies on devising oracle based abstractions that handle annotations from a crowd and minimize the noise in answers [13, 24, 35]. Perhaps, more relevant, are studies on how rules can be verified with the help of the crowd. One recent example is CrowdGame [37] which validates a rule by showing either the rule or its matching instances to the annotators.Th e authors demonstrate that game-based techniques yield good results for verification. Unlike DARWIN CrowdGrame assumes a pre-existing set of rules from which the best rule should be selected.Th is game-based approach to annotate a rule can be modeled as an Oracle in DARWIN. Optimizing crowd accuracy and crowd allocation strategy are outside the scope of this paper.

6 CONCLUSION

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

REFERENCES

[1] Appen, https://appen.com.

[2] Clueweb, https://lemurproject.org/clueweb09/.

[3] Embeddings, https://spacy.io/models/en#en_core_web_lg.

[4] Nell, http://rtw.ml.cmu.edu/rtw/kbbrowser/.

[5] Spacy, https://spacy.io/.

[6] Enrique Alfonseca, Katja Filippova, Jean-Yves Delort, and Guillermo Garrido. Pattern learning for relation extraction with a hierarchical topic model. In *ACL*, 2012.

[7] Allan Peter Davis,Th omas C Wiegers, Phoebe M Roberts, Benjamin L King, Jean M Lay, Kelley Lennon-Hopkins, Daniela Sciaky, Robin Johnson, Heather Keating, Nigel Greene, et al. A ctd–pfizer collaboration: manual curation of 88 000 scientific articles text mined for drug–disease and drug–phenotype interactions. *Database*, 2013, 2013.

[8] Xin Luna Dong and Divesh Srivastava. Big data integration. *Synthesis Lectures on Data Management*, 7(1), 2015.

[9] L. Eadicicco. Baidu's andrew ng on the future of artificial intelligence, 2017. *Time [Online; posted 11-January-2017]*.

[10] Sainyam Galhotra, Donatella Firmani, Barna Saha, and Divesh Srivastava. Robust entity resolution using random graphs. In *SIGMOD*, 2018.

[11] Sainyam Galhotra, Donatella Firmani, Barna Saha, and Divesh Srivastava. Efficient and effective er with progressive blocking. *The VLDB Journal*, pages 1–21, 2021.

[12] Sainyam Galhotra, Behzad Golshan, and Wang-Chiew Tan. Adaptive rule discovery for labeling text data. *arXiv preprint arXiv:2005.06133*, 2020.

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

[14] Anja Gruenheid, Besmira Nushi, Tim Kraska, Wolfgang Gatterbauer, and Donald Kossmann. Fault-tolerant entity resolution with the crowd. *arXiv preprint arXiv:1512.00537*, 2015.

[15] Braden Hancock, Paroma Varma, Stephanie Wang, Martin Bringmann, Percy Liang, and Christopher Ré. Training classifiers with natural language explanations. *arXiv preprint arXiv:1805.03818*, 2018.

[16] Marti A Hearst. Automatic acquisition of hyponyms from large text corpora. In *Coling 1992 volume 2:Th e 15th international conference on computational linguistics*, 1992.

[17] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

[18] Yaliang Li, Jing Gao, Chuishi Meng, Qi Li, Lu Su, Bo Zhao, Wei Fan, and Jiawei Han. A survey on truth discovery. *ACM SIGKDD Explorations Newsletter*, 17(2), 2016.

[19] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[20] C. Metz. Google's hand-fed ai now gives answers, not just search results, 2016. *Wired [Online; posted 29-November-2016]*.

[21] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL/IJCNLP*, 2009.

[22] Ndapandula Nakashole, Gerhard Weikum, and Fabian Suchanek. Patty: A taxonomy of relational patterns with semantic types. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1135–1145, 2012.

[23] Slav Petrov, Dipanjan Das, and Ryan McDonald. A universal part-of-speech tagset. *arXiv preprint arXiv:1104.2086*, 2011.

[24] Protiva Rahman, Courtney Hebert, and Arnab Nandi. Icarus: minimizing human effort in iterative data completion. In *PVLDB*, volume 11, page 2263. NIH Public Access, 2018.

[25] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. *PVLDB*, 11(3), 2017.

[26] Theodoros Rekatsinas, Manas Joglekar, Hector Garcia-Molina, Aditya Parameswaran, and Christopher Ré. Slimfast: Guaranteed results for data fusion and source reliability. In *SIGMOD*, 2017.

[27] Stephen Roller, Douwe Kiela, and Maximilian Nickel. Hearst patterns revisited: Automatic hypernym detection from large text corpora. *arXiv preprint arXiv:1806.03191*, 2018.

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

[29] Rion Snow, Daniel Jurafsky, and Andrew Y Ng. Learning syntactic patterns for automatic hypernym discovery. *Advances in Neural Information Processing Systems 17*, 2004.

[30] Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. Semantic compositionality through recursive matrix-vector spaces. In *EMNLP-CoNLL*, 2012.

[31] Paroma Varma, Bryan D. He, Payal Bajaj, Nishith Khandwala, Imon Banerjee, Daniel L. Rubin, and Christopher Ré. Inferring generative model structure with static analysis. In *NIPS*, 2017.

[32] Paroma Varma and Christopher Ré. Snuba: Automating weak supervision to label training data. *PVLDB*, 2019.

[33] Jinpeng Wang, Gao Cong, Wayne Xin Zhao, and Xiaoming Li. Mining user intents in twitter: A semi-supervised approach to inferring intent categories for tweets. 2015.

[34] Xiaolan Wang, Aaron Feng, Behzad Golshan, Alon Halevy, George Mihaila, Hidekazu Oiwa, and Wang-Chiew Tan. Scalable semantic querying of text. *PVLDB*, 11(9), 2018.

[35] Peter Welinder, Steve Branson, Serge J. Belongie, and Pietro Perona.Th e multidimensional wisdom of crowds. In *NIPS*, 2010.

[36] Fan Yang, Zhilin Yang, and William W Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *NIPS*, 2017.

[37] Jingru Yang, Ju Fan, Zhewei Wei, Guoliang Li, Tongyu Liu, and Xiaoyong Du. Cost-effective data annotation using game-based crowdsourcing. In *PVLDB*, 2018.