WITAN: Unsupervised Labelling Function Generation for Assisted Data Programming

Benjamin Denham
Auckland University of Technology
Auckland, New Zealand
ben.denham@aut.ac.nz

Edmund M-K. Lai
Auckland University of Technology
Auckland, New Zealand
edmund.lai@aut.ac.nz

Roopak Sinha
Auckland University of Technology
Auckland, New Zealand
roopak.sinha@aut.ac.nz

M. Asif Naeem
National University of Computer & Emerging Sciences
Islamabad, Pakistan
asif.naeem@nu.edu.pk

ABSTRACT

Effective supervised training of modern machine learning models often requires large labelled training datasets, which could be prohibitively costly to acquire for many practical applications. Research addressing this problem has sought ways to leverage *weak supervision* sources, such as the user-defined heuristic labelling functions used in the *data programming* paradigm, which are cheaper and easier to acquire. Automatic generation of these functions can make data programming even more efficient and effective. However, existing approaches rely on initial supervision in the form of small labelled datasets or interactive user feedback. In this paper, we propose WITAN, an algorithm for generating labelling functions without any initial supervision. This flexibility affords many interaction modes, including unsupervised dataset exploration before the user even defines a set of classes. Experiments in binary and multi-class classification demonstrate the efficiency and classification accuracy of WITAN compared to alternative labelling approaches.

PVLDB Reference Format:
Benjamin Denham, Edmund M-K. Lai, Roopak Sinha, and M. Asif Naeem. WITAN: Unsupervised Labelling Function Generation for Assisted Data Programming. PVLDB, 15(11): 2334 - 2347, 2022.
doi:10.14778/3551793.3551797

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ben-denham/witan.

1 INTRODUCTION

The benefits of leveraging large datasets for supervised training of classifiers have been clearly demonstrated in recent years. In particular, deep learning algorithms are able to deliver impressive results in text classification and image recognition tasks but are well-known for their reliance on massive training datasets [38]. Unfortunately, acquiring training labels in such large quantities is often prohibitively expensive, especially if labelling requires the assessment of training instances by a domain expert. The scarcity of training labels has led to the development of *weak supervision* methods that leverage supervision sources that are noisier but cheaper to acquire than traditional ground truth training labels. Such weak supervision sources include crowd-sourced labels [47], existing knowledge-bases (in so-called *distant supervision*) [4], and even class names alone [55].

Data programming has emerged as a popular weak supervision paradigm, accepting supervision in the form of user-defined heuristic *labelling functions* (LFs) that assign class labels to instances, typically based on conditions of instance features [50]. For example, an LF for text sentiment classification might label any instance containing the word "wonderful" with the "positive" class. Not only can data programming save effort by enabling the user to label potentially hundreds of instances with each LF, but LFs can also explicitly capture the user's domain knowledge. A set of LFs can serve as high-level documentation of labelling decisions in a way that a large set of labelled instances cannot. This also means that labelling decisions can be re-evaluated and changed in light of new requirements by simply updating the LFs. These attributes make data programming a natural paradigm for users to systematically manage and improve their training data in line with the recent trend toward data-centric AI, where emphasis is placed on improving training data over models [44].

Despite these benefits, users have reported [37] the need for guidance in designing LFs that 1) are accurate, 2) cover many instances, and 3) are not overly biased towards certain classes. While a wide variety of *assisted data programming* approaches have been proposed to help users design LFs (see Section 2.2), almost all of these approaches assume that the user already knows at least the set of classes they wish to identify within their dataset. However, there are many situations where this will not be the case; a user seeking to classify articles by topic or warranty claims by fault type may not have a fixed set of topics or faults in mind before exploring the dataset. Such users would benefit from an unsupervised analysis of their dataset to identify a variety of potential LFs for them to choose from. Even if the user has some idea of their desired classification scheme, such an unsupervised analysis could help them avoid human bias in manual LF design and identify other relevant concepts within the dataset, such as natural sub-classes of their target classes (e.g. subtopics or special cases of faults).

Proceedings of the VLDB Endowment, Vol. 15, No. 11 ISSN 2150-8097.
doi:10.14778/3551793.3551797

To address this need, we propose Witan, an algorithm for unsupervised LF discovery. As illustrated in step ① of Figure 1, Witan takes an unlabelled dataset and uses an information-theoretic approach to generate a set of potential LFs that are diverse not only in the instances they cover but also in the dimensions of the dataset they abstract. In step ②, a domain expert reviews the entire set of generated LFs holistically, selecting LFs relevant to their classification task and deciding which class labels those LFs should assign. Finally, the optional step ③ demonstrates how Witan can extend an existing set of LFs with complementary LFs that increase diversity, whether the existing LFs were generated by Witan or provided by the user as *seed* LFs. Analogous to the historical royal advisors for which it is named [8], Witan aims to elicit new insights and assist in classifier training as much as possible while still empowering users with full control over the set of LFs – a user can easily interpret the behaviour of a proposed LF and decide whether or not to use it to assign a class label, and whether to manually amend the decision criteria to better fit their classification task. In addition to these benefits, we demonstrate that Witan-generated LFs result in competitive classification performance when compared to alternative LF generation and instance labelling approaches.

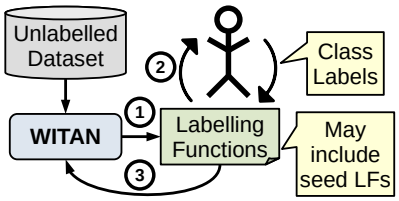

Figure 1: Logical view of Witan, demonstrating ① initial unsupervised generation of LFs, ② user review and assignment of class labels to LFs, and ③ extending an existing set of LFs.

The rest of this paper is organised as follows. Section 2 presents the current state of the data programming approach to weak supervision with an emphasis on assisted data programming. Our novel algorithm (Witan) for unsupervised LF discovery is described in Section 3, which includes the core algorithm as well as optional components to construct LFs based on conjunctions and disjunctions of feature conditions and to guide the selection of additional LFs based on user approval of previously generated LFs, seed LFs, or both. In Section 4, results of extensive experimental evaluation of Witan against other LF generation and labelling approaches for both binary and multi-class classification tasks are presented. They demonstrate the ability of Witan to approach peak classification performance with less user effort than other methods. Finally, Section 5 concludes the paper with suggestions for future work.

2 BACKGROUND AND RELATED WORK

2.1 Data Programming for Weak Supervision

Data programming is a recently proposed paradigm for weak supervision from user-defined labelling functions (LFs) [50]. In the context of a classification task to predict the class label $y \in \mathcal{Y}$ for an instance with features $x \in \mathcal{X}$, a labelling function λ uses an instance's features to either assign a predicted class label or abstain ($\emptyset$), such that $\lambda : \mathcal{X} \rightarrow \mathcal{Y} \cup \{\emptyset\}$ [49]. While LFs can potentially use complex decision criteria based on distantly supervising knowledge bases [49] or arbitrary supervised classifiers [49, 53], we focus on LFs that can be expressed as conditions on instance features, such that a user would be able to interpret and potentially amend the decision criteria of a machine-generated LF.

As LFs essentially act as imperfect classifiers, one or more LFs may assign different class labels to a given instance. To account for this, arguably the most important component in a data programming system is a *labelling model* to aggregate LF outputs for training instances into a consistent set of training class labels. The seminal Snorkel system [49] learns a generative model that estimates the accuracies of LFs based solely on their agreements and disagreements on training instances, without the need for any ground truth labels. The generative model is used to produce *probabilistic training labels*, where each of m possible class labels is assigned a numeric probability: $\tilde{y} = (\tilde{y}_1, \ldots, \tilde{y}_m), \tilde{y}_i \in [0, 1]$. These probabilistic training labels can then be used to train a classifier using a noise-aware loss function. Alternative labelling models have been subsequently proposed based on triplet methods [21] and game optimisation [5]. Given the popularity of Snorkel's open-source library and its use in previous studies on assisted data programming, we will use it to experimentally evaluate LFs produced by Witan and other assisted data programming methods in the context of classification.

As Witan and other assisted data programming methods aid in the construction of LFs, we highlight desiderata for LFs that produce accurate classifiers. The primary trade-off in LF design is between coverage and accuracy [10, 53]; LFs that are general enough to apply to many instances risk misclassifying a significant proportion of those instances, while highly specific and accurate LFs may cover too few instances for the subsequently trained classifier to generalise to broader patterns. An ideal set of LFs optimises for both coverage and accuracy, taking advantage of the fact that labelling models are best suited to resolving a moderate degree of overlap and disagreement between LFs [49]. As labelling models often assume the class distribution can be specified a priori or estimated from LF outputs [21, 49], it is also important for LF coverage to be representative of the true class distribution and not unduly biased towards certain classes over others. In order to avoid catastrophic results for extremely biased LFs, our experiments will apply Snorkel with a uniform class prior. Finally, while Snorkel has mechanisms for explicitly modelling dependencies between LFs (e.g. when one LF "reinforces" another by identically labelling a strict subset of its covered instances [54]), it assumes independence by default [49]. Methods have been developed for automatically learning dependencies [54], and other labelling models are designed to be robust to dependencies [5]. However, because empirical results suggest classifier accuracy can be degraded by modelling all but the strongest dependencies [11], we will limit our experiments to applying Snorkel under the assumption that all LFs are independent.

2.2 Assisted Data Programming

A variety of methods have previously been proposed to help users construct effective LFs with minimal effort. We now review these *assisted data programming (ADP)* methods, focussing on the interaction modes they support and how they compare to Witan.

Some ADP methods simplify the user's task of defining decision criteria for LFs. For text classification, one approach is to have the user simply specify keywords associated with each class [25, 28, 32]. Such keyword lists can be filtered of "noisy" keywords through co-training [28], used to seed LDA topic models applied as classifiers [32], or even applied to non-text classification tasks by leveraging text descriptions for some training instances [25]. Another approach for assisting users is to identify subsets of instances [13] or regions of the input space [52] where existing LFs perform poorly so that the user may target those deficiencies with additional LFs. However, these methods still rely on users designing criteria for LFs themselves, which relies on them manually reviewing many instances to identify common patterns of interest.

Another family of ADP methods generates LFs from a small number of manually labelled instances. Some of these methods still rely on the user providing the criteria that influenced their labelling decisions to form the basis of LFs, whether by making a search query for instances to cover with an LF [37], marking important keywords in text [19], or providing natural language explanations for their decisions [26]. Other methods generate LFs based solely on labelled instances. One such method [36] generates LFs that propagate manual labelling to similar instances found using a text search engine, though such instance-based LFs lack interpretable decision criteria. Other methods generate LFs by training arbitrary classifiers on a small labelled training set [41, 53]. The Snuba system [53] can produce reasonably interpretable LFs when a simple classification model (such as a decision stump) is used to generate LFs. Because of this, we will include Snuba as a representative of instance-based LF generation methods in our experimental evaluation.

Rather than having the user label instances, some assisted data programming methods propose LFs and ask the user to review their decision criteria and then approve or reject each LF. Similar approaches for active learning predate data programming, where users review iteratively proposed classification rules to expand a labelled training set [17, 48]. Methods for data programming have been proposed that can learn LFs either for a single "positive" class [23, 24] or for both classes in a binary classification task [10, 29]. As it is important for the user to be able to interpret and evaluate the decision criteria of a proposed LF, these methods have been used to propose LFs with decision criteria based on the presence of text keywords [10, 23, 29], itemset patterns [24], regular expressions and parse tree structures [23], and neighbourhoods of images [10]. Of these methods, only Interactive Weak Supervision (IWS) [10] is amenable to both multi-class classification (though only details for binary classification are described by the authors) and non-text-based LFs. Therefore, we will include IWS in our experimental evaluation as a representative of methods requiring user review of generated LFs. A limitation of this family of methods is their reliance on continuous feedback from the user after each LF is proposed, as the feedback informs the selection of the next LF.

A key limitation of all the ADP approaches discussed above is that they require the user to know the set of classes for their classification task a priori. We believe that the ability to begin data programming without a clear idea of the final class structure is invaluable for many tasks, such as in document topic classification, where the user is interested in first discovering the range of topics that are present. This need inspired our development of Witan,

Table 1: Assisted Data Programming Approaches

Does not require:	Labelled instances	Designed LFs	Prior set of classes	Continuous feedback
LF design aids [13, 25, 28, 32, 52]	✓	×	×	×
Instance-based LF design aids [19, 26, 37]	×	×	×	✓
Instance-based LF generation [36, 41, 53]	×	✓	×	✓
Feedback-based LF generation [10, 23, 24, 29]	✓	✓	×	×
LFs from clusters [3, 33, 34]	✓	User interaction not supported		
Clustering by intent [7, 20]	Min 1 class	✓	✓	✓
Witan	✓	✓	✓	✓

which requires no initial supervision to generate a set of possible LFs for the user to review and assign class labels to. While unsupervised clustering has previously been used to create LFs [3, 33, 34], such methods only present a single possible clustering to the user, whereas Witan will propose diverse LFs that represent a variety of orthogonal partitionings of the data. The inherent subjectivity of clustering and the importance of allowing users to choose between equally valid clusterings has been recognised in the *interactive clustering* literature [6], though the majority of interactive clustering methods do not produce clusters with interpretable inclusion criteria that are stable between user interactions [43] as desired for LFs. *Clustering by intent (CBI)* [7, 20] is an interactive clustering approach that produces interpretable rules similar to the LFs generated by Witan, but it relies on the user providing labelled instances for at least one class at the outset.

Table 1 compares Witan to other ADP approaches; Witan's key distinguishing properties are summarised below:

- The user does not need to design LF decision criteria; the user only evaluates and labels Witan-generated LFs.
- No manual instance labelling is required; all labels are assigned by LFs with interpretable decision criteria.
- LFs can be generated in an unsupervised fashion without a priori knowledge of the set of classes.
- The user can review the whole set of LFs after generation; Witan does not need feedback after generating each LF.

3 THE PROPOSED ALGORITHM

We now present our proposed Witan algorithm for unsupervised LF generation. We first describe the utility measure for selecting LFs and use it to formulate the core Witan algorithm. We then present extensions for generating LFs with conjunctive and disjunctive conditions and for incorporating user feedback into LF selection.

We assume Witan is provided with a training set $X \in \{0,1\}^{n,m}$ of n unlabelled instances comprised of m binary features. Binary

features are required by Witan's utility function, but they also allow us to construct an LF λ using any Boolean combination of a feature subset $d^\lambda \in \mathcal{P}(\{1,\ldots,m\})$ to produce a coverage vector $c^\lambda \in \{0,1\}^n$. We say that LF λ assigns a user-given class label y^λ to instances covered by c^λ and abstain on non-covered instances:

$$\lambda(x_i) = \begin{cases} y^\lambda & \text{if } c_i^\lambda = 1 \\ \emptyset & \text{otherwise} \end{cases} \tag{1}$$

For example, a simple LF λ_j that covers instances for which feature j is positive would have $d^{\lambda_j} = \{j\}$ and $c^{\lambda_j} = x_{*j}$. Note that, as an unsupervised algorithm, Witan selects LFs with optimal coverage vectors c^λ and leaves class label y^λ for the user to decide later. Following previous advice on selecting LFs [10, 49], users should be instructed to approve an LF for a class if they believe it will be more accurate than an LF that randomly assigns any class label.

Datasets containing non-binary features can still be used with Witan by applying appropriate feature transformations, such as one-hot encoding categorical features and binning or thresholding numeric features. In the framework of Boecking et al.'s LF families [10], binary features could be produced from any LF family where each LF assigns only a single class value. These include LF families that test for the presence of a keyword in text, test for the presence of motifs in a times series, or detect the presence of objects in an image. Whatever binary features are used, it is important that a user should be able to easily understand which instances a feature would be positive for so that they can judge whether or not that feature would be a useful decision criterion for an LF. Throughout this section, we will describe examples of applying Witan to a set of binary bag-of-words features that each indicate the presence of a particular word in a film or television review, with the intent of generating LFs for positive/negative sentiment classification.

3.1 LF Utility Function

In order for Witan to select LFs to propose to the user, we require a utility function by which we can compare a set of candidate LFs. Ideally, this function should select LFs that are good classifiers for one of the classes in the target classification task. A similar problem is faced by the "covering rule" learning algorithms used for classification, which must select feature conditions that optimise a rule's classification performance [22]. To solve this problem, the classic CN2 rule learner [12] takes an information-theoretic approach by preferring a rule with coverage vector c that minimises the conditional entropy $H(Y|c)$ of the class variable Y. Decision tree learners also commonly make branching decisions b that maximise the reduction in entropy from the previous model state to the new state, referred to as the *information gain* [22, 46]: $IG(Y,b) = H(Y) - H(Y|b)$.

In the context of unsupervised LF generation, the immediate barrier to using entropy in a utility function is that we do not have access to class labels Y for our training set, so we cannot determine which LF conditions will help predict Y. However, assuming the binary features in X are relevant to the classification task, it is fair to assume that at least some features will correlate with Y. Therefore, Witan will aim to construct an LF condition c^λ to predict a class value $y \in Y$ with $P(y|c^\lambda) \gg P(y)$ by reducing the entropy of y-correlated features x_{*j}. To see how this works, consider the relationship between $P(y|c^\lambda)$ and $P(y|x_{*j})$:

$$\begin{aligned}
P(y|c^\lambda) &= P(x_{*j}|c^\lambda)P(y|c^\lambda, x_{*j}) + P(\neg x_{*j}|c^\lambda)P(y|c^\lambda, \neg x_{*j}) \\
&\geq P(x_{*j}|c^\lambda)P(y|c^\lambda, x_{*j}) \text{ (law of total prob.)} \\
P(y|c^\lambda, x_{*j}) &\to P(y|x_{*j}) \text{ as } P(c^\lambda|x_{*j}) \to 1 \text{ (prob. chain rule)} \\
P(c^\lambda|x_{*j}) &= P(x_{*j}|c^\lambda)P(c^\lambda)/P(x_{*j}) \text{ (Bayes' rule)} \\
\therefore\ &\min(P(y|c^\lambda)) \\
&\to P(y|x_{*j}) \text{ as } P(x_{*j}|c^\lambda) \to 1 \ \&\ P(c^\lambda)/P(x_{*j}) \to 1 \\
&\to P(y|\neg x_{*j}) \text{ as } P(\neg x_{*j}|c^\lambda) \to 1 \ \&\ P(c^\lambda)/P(\neg x_{*j}) \to 1
\end{aligned} \tag{2}$$

Therefore, the minimum ability of c^λ to predict y can approach that of x_{*j} by maximising its coverage $P(c^\lambda)$ and ability to predict x_{*j}: $P(x_{*j}|c^\lambda)$ (or $P(\neg x_{*j}|c^\lambda)$ if x_{*j}'s absence predicts y). For example, with our bag-of-words features, an LF condition based on a word like "wonderful" that is common (relatively high $P(c^\lambda)$) and helps predict the presence of positive words ($P(\text{"loved"}|\text{"wonderful"}) \gg P(\text{"loved"})$ where $P(y^+|\text{"loved"}) \gg P(y^+)$) and the absence of negative words ($P(\neg\text{"hated"}|\text{"wonderful"}) \gg P(\neg\text{"hated"})$ where $P(y^+|\neg\text{"hated"}) \gg P(y^+)$) also helps predict the positive class: $P(y^+|\neg\text{"wonderful"}) \gg P(y^+)$. Because Witan does not know which features predict classes through either their presence or absence, it should aim to minimise the entropy of all features (maximising $\max(P(x_{*j}|c^\lambda), P(\neg x_{*j}|c^\lambda))\forall x_{*j}$). Similar entropy-based metrics have been used in hierarchical divisive clustering (HDC) to select features for splitting clusters: splits based on maximising the mean of the gain ratio [45] or normalised information gain [56] over all feature variables compared favourably to other criteria [56]. While these algorithms seek a single optimal clustering that assigns each instance to a single cluster, Witan will select LFs that can partition instances in a variety of ways, allowing the user to choose which LFs best suit their classification task. This LF diversity is achieved by ensuring the full set of LFs reduces entropy across many different features, e.g. "horror" predicts genre-aligned words that are not predicted by sentiment-aligned keywords.

In light of the discussion above, Witan's utility function is based on the information gain achieved by LF λ on a binary feature x_{*j}:

$$IG(x_{*j}, c^\lambda) = H(x_{*j}) - H(x_{*j}|c^\lambda) \tag{3}$$

where the entropy of Boolean variable E is given by:

$$H(E) = -P(E)\log_2 P(E) - (1-P(E))\log_2(1-P(E)) \tag{4}$$

and the probabilities can be calculated from the training set:

$$P(x_{*j}) = \frac{1}{n}\sum_{i=1}^{n} x_{i,j} \qquad P(x_{*j}|c^\lambda) = \frac{\sum_{i=1}^n x_{i,j}c_i^\lambda}{\sum_{i=1}^n c_i^\lambda} \tag{5}$$

Because the coverage vectors are binary, there is no need to normalise the information gain or use a gain ratio to reduce bias towards multi-valued conditions [56].

We can now define the utility function U that Witan will aim to maximise when selecting LFs:

$$U_{X,\bar{H},\gamma,w}(\lambda) = \sum_{\substack{j=1,\\ j\notin d^\lambda}}^{m} w_j \sum_{\substack{i=1,\\ c_i^\lambda=1}}^{n} \max(0, \bar{h}_{i,j} - H(x_{*j}|c^\lambda))^\gamma \tag{6}$$

In essence, U sums the information gain achieved by LF λ across all features j over all covered instances i, with several parameters and characteristics of note:

- Values from an entropy matrix $\bar{H}$ are used in place of $H(x_{*j})$, allowing Witan to update $\bar{H}$ to account for the information gain of previously selected LFs.
- As an additional LF cannot detract from information already gained, negative gain is ignored through use of max.
- To reduce bias towards less interpretable LFs based on many features, information gain on any feature in d^λ is ignored.
- Because information gain is only summed over instances covered by c^λ, U favours higher coverage as desired.
- We found empirically that LFs with high information gain over a smaller number of instances were more useful than those with weaker gain over many instances. Such LFs can be preferred by setting $\gamma > 1$, which increases the importance of high information gain. We found $\gamma = 2$ to be a reasonable setting in our ablation study (Section 4.5).
- Weights vector $w \in \{[0, +\infty)\}^m$ controls the relative importance of gain on each feature. We will discuss the use of w in Section 3.4, but for now assume all weights are 1.

3.2 Core of the Algorithm

Having defined utility function U, we now use it to formulate the core Witan algorithm laid out in Algorithm 1. Witan begins by initialising matrix $\bar{H}$ with the entropy of each feature for each instance in the training data X (line 1) and a set of candidate LFs $\hat{\Lambda}$ that each represent a simple condition on a single binary feature (line 2). The user may choose to constrain the search space of candidate LFs to only include those covering a minimum proportion $c^{\min} \in [0, 1]$ of training instances, and may optionally specify a set of seed LFs $\tilde{\Lambda}$ to initialise the set of selected LFs Λ (lines 3–9).

Each iteration of Witan's main loop (lines 10–17) selects the candidate LF that maximises utility function U (line 11, Equation (6)) and adds it to Λ (line 13). After an LF λ is selected, the entropy matrix $\bar{H}$ used as the baseline entropy by U is updated (line 15). Entries for all instances i covered by the LF ($c_i^\lambda = 1$) are assigned the LF's conditional entropies for each feature j ($H(x_{*j}|c^\lambda)$) if those entropies are less than the current $\bar{H}_{i,j}$ entries (line 23). In this way, U's information gain will always be computed against the minimum entropy achieved by any selected LF matching a given instance. Because of this, each iteration selects a new LF that can provide information gain either on instances not covered by other LFs (increasing coverage) or on features that are not well predicted by other LFs (increasing LF diversity). We would ideally compute information gain against the conditional entropy of the full set of selected LFs ($H(x_{*j}|c^\lambda \; \forall \; \lambda \in \Lambda)$), but the computational cost of fully re-computing $\bar{H}$ in each iteration would be prohibitive, and taking the minimum entropy implies a conservative assumption that the set of selected LFs has maximal information redundancy.

Witan's main loop continues until Λ covers a user-specified proportion $C^{\min} \in [0, 1]$ of training instances (line 10), at which point Λ is returned so that the user may choose which LFs are relevant to their classification task and assign class labels to them.

Witan's algorithm also contains several procedures that we will provide alternative definitions for when we describe extensions to the core algorithm in subsequent sections. These control the feature weights used by the utility function U (`getWeights`, which currently weights all features equally), how the set of candidate LFs $\hat{\Lambda}$ is updated after an LF is selected (`updateCandidates`, which currently just removes the selected LF from the candidate set), and how a selected candidate LF may be extended before being added to Λ (`extendLF`, which does not currently extend LFs).

Algorithm 1: Core Witan Algorithm

Input : Training data X, seed LFs $\tilde{\Lambda}$, gain exponent γ, min. LF coverage $c^{\min}$, stopping coverage $C^{\min}$
Output : Set of proposed LFs Λ

1 $\bar{h}_{i,j} \leftarrow H(x_{*j}) \; \forall \; i \in \{1, \ldots, n\}, \; j \in \{1, \ldots, m\}$;
2 $\hat{\Lambda} \leftarrow \{\hat{\lambda}[d^{\hat{\lambda}} = \{j\}; \; c^{\hat{\lambda}} = x_{*j}]$
$\forall \; j \in \{1, \ldots, m\} : \frac{1}{n} \sum_{i=1}^{n} c_i^{\hat{\lambda}} \geq c^{\min}\}$;
3 $\Lambda \leftarrow \{\}$;
4 **foreach** $\tilde{\lambda} \in \tilde{\Lambda}$ **do**
5 $\quad \Lambda \leftarrow \Lambda \cup \{\tilde{\lambda}\}$;
6 $\quad w \leftarrow$ `getWeights(`Λ`)`;
7 $\quad$ `updateEntropies(`$\bar{H}, \tilde{\lambda}$`)`;
8 $\quad$ `updateCandidates(`$\hat{\Lambda}, \tilde{\lambda}$`)`;
9 **end**
10 **while** $\frac{1}{n} \sum_{i=1}^{n} \max(\{c_i^\lambda \; \forall \; \lambda \in \Lambda\}) < C^{\min}$ **do**
11 $\quad \lambda \leftarrow \arg\max_{\hat{\lambda} \in \hat{\Lambda}} U_{X, \bar{H}, \gamma, w}(\hat{\lambda})$;
12 $\quad \lambda \leftarrow$ `extendLF(`λ`)`;
13 $\quad \Lambda \leftarrow \Lambda \cup \{\lambda\}$;
14 $\quad w \leftarrow$ `getWeights(`Λ`)`;
15 $\quad$ `updateEntropies(`$\bar{H}, \lambda$`)`;
16 $\quad$ `updateCandidates(`$\hat{\Lambda}, \lambda$`)`;
17 **end**
18 **return** Λ;
19 **Procedure** `getWeights(`Λ`)`**:**
20 $\quad w_j \leftarrow 1 \; \forall \; j \in \{1, \ldots, m\}$;
21 $\quad$ **return** w
22 **Procedure** `updateEntropies(`$\bar{H}, \lambda$`)`**:**
23 $\quad \bar{h}_{i,j} \leftarrow \min(\bar{h}_{i,j}, H(x_{*j}|c^\lambda)) \; \forall \; i, j : c_i^\lambda = 1$;
24 **Procedure** `updateCandidates(`$\hat{\Lambda}, \lambda$`)`**:**
25 $\quad \hat{\Lambda} \leftarrow \hat{\Lambda} \setminus \{\lambda\}$;
26 **Procedure** `extendLF(`λ`)`**:**
27 $\quad$ **return** λ;

3.3 Conjunctive and Disjunctive LFs

In the core Witan algorithm, the best LF is selected from a set of candidates in each iteration, which is similar to IWS's approach of searching through a family of LFs [10]. To limit the computational cost in each iteration, both the core Witan algorithm and IWS constrain the candidate set to LFs with conditions based on a single feature or heuristic. In this section, we extend the core Witan algorithm to generate LFs based on logical conjunctions (ANDs)

Algorithm 2: `updateCandidates` for conjunctive LFs

Input : Training data X, gain exponent γ, min. LF coverage $c^{\min}$, entropy matrix $\bar{H}$, weights vector w, set of candidate LFs $\hat{\Lambda}$, newly selected LF λ

Output : Updated set of candidate LFs $\hat{\Lambda}$

1 $\hat{\Lambda} \leftarrow \hat{\Lambda} \cup \{\hat{\lambda}[d^{\hat{\lambda}} = (d^{\lambda} \cup \{j\});\ c^{\hat{\lambda}} = (c^{\lambda} \wedge x_{*j});\ p^{\hat{\lambda}} = \lambda] \quad \forall\ j \in \{1, \ldots, m\} \quad : \frac{1}{n}\sum_{i=1}^{n} c_i^{\hat{\lambda}} \geq c^{\min},\ j \notin d^{\lambda}\}$;

2 $\hat{\Lambda} \leftarrow \hat{\Lambda} \setminus \{\hat{\lambda}\ \forall\ \hat{\lambda} \in \hat{\Lambda} : p^{\hat{\lambda}} = p^{\lambda},\ d^{\hat{\lambda}} \subseteq d^{\lambda}\}$;

3 $\hat{\Lambda} \leftarrow \underset{\hat{\Lambda}' \subseteq \hat{\Lambda},\ |\hat{\Lambda}'| \leq m}{\arg\max} \sum_{\hat{\lambda} \in \hat{\Lambda}'} U_{X,\bar{H},\gamma,w}(\hat{\lambda})$;

Algorithm 3: `extendLF` for disjunctive LFs

Input : Training data X, gain exponent γ, entropy matrix $\bar{H}$, set of candidate LFs $\hat{\Lambda}$, newly selected LF λ

Output : Extended selected LF λ

1 **loop**

2 $\quad w'_j \leftarrow \max(0, IG(x_{*j}, c^{\lambda}))\ \forall\ j \in \{1, \ldots, m\}$;

3 $\quad \hat{\lambda}' \leftarrow \underset{\hat{\lambda} \in \hat{\Lambda},\ p^{\hat{\lambda}} = p^{\lambda}}{\arg\max} U_{X,\bar{H},\gamma,w'}(\hat{\lambda})$;

4 $\quad \lambda' \leftarrow \lambda'[d^{\lambda'} = (d^{\lambda} \cup d^{\hat{\lambda}'}); c^{\lambda'} = (c^{\lambda} \vee c^{\hat{\lambda}'}); p^{\lambda'} = p^{\lambda}]$;

5 $\quad$ **if** $U_{X,\bar{H},\gamma,w'}(\lambda') > U_{X,\bar{H},\gamma,w'}(\lambda)$ **then**

6 $\quad\quad \lambda \leftarrow \lambda'$;

7 $\quad$ **else**

8 $\quad\quad$ **return** λ;

9 $\quad$ **end**

10 **end**

and disjunctions (ORs) of feature conditions, incurring minimal additional computational cost by dynamically updating the candidate set and selected LFs. We expect this to improve the effectiveness of selected LFs, as conjunctive LFs can target narrower subsets of instances to achieve higher information gain, while disjunctive LFs can target wider sets of instances to achieve higher coverage.

In Algorithm 2, we redefine `updateCandidates` to extend the LF candidate set $\hat{\Lambda}$ to include logical conjunctions of a selected LF λ's condition with other feature conditions. Specifically, for each feature j that is not already part of the selected LF's condition ($j \notin d^{\lambda}$) we construct a new conjunctive candidate LF $\hat{\lambda}$ such that $d^{\hat{\lambda}} = d^{\lambda} \cup \{j\}$ and $c^{\hat{\lambda}} = c^{\lambda} \wedge x_{*j}$, excluding LFs that do not achieve minimum coverage $c^{\min}$ (line 1). As the conjunction means that each new $\hat{\lambda}$ will only cover a subset of the instances already covered by λ, we can consider $\hat{\lambda}$ to be a *child* of λ. This forms a hierarchical structure of LFs, notated by defining the parent of $\hat{\lambda}$ as $p^{\hat{\lambda}} = \lambda$. This conjunctive structure enables the expression of LFs that identify subcategories of other LFs: an LF conditional on the word "movie" could be the parent of LFs conditional on words that identify how that movie is described, such as "horrible" or "liked". The new procedure definition also alters how selected LFs are removed from the candidate set to account for both conjunctive and disjunctive LFs: we now remove any candidates that are siblings of the original LF λ and are conditioned by a subset or equal set of features (line 2). Finally, to prevent increasing computational cost through the growth of the candidate set, we maintain the original size of m by only retaining candidates with the highest utility (line 3).

In Algorithm 3, we redefine `extendLF` to attempt extending a selected LF λ into a disjunction between the original LF's condition and other candidate LF conditions. Each iteration of `extendLF`'s main loop finds the sibling candidate LF $\hat{\lambda}'$ that maximises the utility function with weights w' that are relative to the information gain achieved by λ on each feature (lines 2–3). A sibling that provides information gain on a similar set of features to λ tends to have a logical similarity with λ, which leads to a disjunctive condition that makes sense to the user. For example, an LF for positive word "wonderful" could be extended with a disjunction to a similar positive word, such as "excellent". LFs λ and $\hat{\lambda}'$ are used to construct a new disjunctive LF λ' such that $d^{\lambda'} = d^{\lambda} \cup d^{\hat{\lambda}'}$ and $c^{\lambda'} = c^{\lambda} \vee c^{\hat{\lambda}'}$ (line 4). If λ' provides greater w'-weighted utility than the original LF λ, then we can say that it is a superior LF fulfilling the same role for feature diversity, and we can replace λ with λ' (lines 5–6). Iteration continues to add further disjunctions to λ until there is no further increase in utility, at which point λ is returned so that it can be added to Λ (line 8).

3.4 Incorporating Feedback

We have highlighted the benefits of being able to use Witan in an unsupervised fashion, namely that the user does not need to have a fixed idea of their class structure before Witan can begin LF generation; input is only required from the user to assign class labels to Witan-generated LFs. However, other LF generation methods like IWS and Snuba make use of some form of user supervision to guide the generation of LFs that are suited to the user's classification task. We now describe an approach for guiding Witan's selection of LFs with user feedback on seed and previously generated LFs.

We allow the user to provide feedback by marking any LF in the set of selected LFs Λ as `approved`. We represent feedback on an LF λ with $f^{\lambda} \in \{\texttt{approved}, \texttt{null}\}$, where $f^{\lambda} = \texttt{null}$ when the user has not provided feedback for λ. Given feedback for a subset of selected LFs, we would like Witan to select subsequent LFs that support a class structure consistent with approved LFs: $\{\lambda\ \forall\ \lambda \in \Lambda : f^{\lambda} = \texttt{approved}\}$. One way of interpreting this feedback is that `approved` LFs have successfully achieved information gain on features that are correlated with the target class variable. For example, if the user approves an LF for positive word "wonderful" that provides information gain on other positive and negative words, then we should favour LFs that also provide information gain on such words as they will likely also be effective classifiers of positive/negative sentiment. Therefore, we can utilise feedback by using it to update the weight w_j of information gain for each feature j in utility function U, increasing the weight of features for which `approved` LFs provided high information gain. The intent is to prioritise LFs for classes complementary to `approved` LFs, as such LFs will optimise Witan's utility function by further reducing entropy on similar feature sets while covering different sets of instances.

In Algorithm 4, we redefine `getWeights` to incorporate feedback f^{λ} for selected LFs $\lambda \in \Lambda$. If no feedback is available, equal weights are returned (line 3), otherwise weights are computed as follows.

Algorithm 4: `getWeights` for incorporating feedback

Input : Training data X, selected LFs Λ
Output: Weights vector w

1 $\Lambda^f \leftarrow \{\lambda \in \Lambda : f^\lambda = \text{approved}\}$;
2 **if** $\Lambda^f = \emptyset$ **then**
3 $\quad w_j \leftarrow 1 \; \forall \; j \in \{1, \ldots, m\}$;
4 **else**
5 $\quad w_j \leftarrow 0 \; \forall \; j \in \{1, \ldots, m\}$;
6 $\quad$ **foreach** $\lambda \in \Lambda^f$ **do**
7 $\quad\quad w'_j \leftarrow \max(0, IG(x_{*j}, c^\lambda)) \; \forall \; j$;
8 $\quad\quad w_j \leftarrow w_j + \frac{w'_j}{\sum_{l=1}^{m} w'_l} \; \forall \; j$;
9 $\quad$ **end**
10 **end**
11 **return** w

For each `approved` LF λ, we construct vector w' of the information gains achieved by λ on each feature (line 7). The final weights vector w is constructed by summing normalised w' for all `approved` LFs (line 8), such that the sum of weights contributed by each LF equals 1. Note that in Algorithm 1, the weights are updated in each iteration (line 14), allowing the user to interactively guide Witan by immediately providing feedback on each LF selected by Witan.

3.5 Analysis of Witan

We now analyse Witan, highlighting interaction modes it affords, assessing its computational complexity, qualitatively analysing Witan-generated LFs, and discussing potential failure modes.

3.5.1 Interaction Modes. Because Witan can be run without any labelled instances, seed LFs, or even predefined class structure, it achieves the primary goal of performing LF generation without any initial supervision. Once a set of LFs has been generated, the user can choose which LFs to assign class labels. We argue that this is a straightforward task for the user, as the conjunctive and disjunctive feature-based conditions generated by Witan are simply expressed in conjunctive normal form (an "AND of ORs"). Similar decision rules have been found to be a highly understandable format for expressing the behaviour of machine learning models to users [51]. Additionally, a user study with IWS's single-condition LFs found that users were able to label LFs accurately and that users could label LFs faster than they could label individual instances [10].

Users may also provide initial seed LFs for Witan to grow with additional LFs that increase coverage and feature diversity. Furthermore, once a user has labelled a set of Witan-generated LFs, they may use them as seed LFs in a subsequent execution of Witan. With the feedback extension described in Section 3.4, Witan can use the knowledge of whether or not a generated LF was approved and assigned a class label to guide LF generation toward those that are more likely to be useful for the user's target classification task.

In the extreme case, a user may operate Witan in a step-by-step manner: generating a single LF, labelling that LF, and then continuing to run Witan with feedback. Such a step-by-step approach is not suitable when the user must review a full set of LFs to determine the class structure, but rather when the user already knows their desired class structure (as required for feedback-based LF generators like IWS) or would be able to recognise that LF conditions that identify classes relevant to their classification task. For example, a user starting to classify warranty claims according to fault type may not know what types of fault exist but would recognise that LF conditions on keywords like "battery" or "charging" clearly identify a battery-related fault type. We experiment with step-by-step labelling to demonstrate the feedback extension in Section 4.5.

3.5.2 Computational Complexity. The time complexity of each Witan iteration is dominated by the computation of utility function U for each candidate LF $\hat{\lambda} \in \hat{\Lambda}$. While the $O(n)$ computation of $H(x_{*j}|c^{\hat{\lambda}})$ for each j and $\hat{\lambda}$ can be cached between iterations, performing the sum over m features and n instances in U for each of up to m candidate LFs still results in a complexity of $O(m^2 n)$. Therefore, the runtime of Witan is largely determined by the number of features, as demonstrated in our runtime study (Section 4.3).

The time complexity of Witan is not affected by the proposed extensions to the core algorithm. `updateCandidates` in Algorithm 2 computes U for up to m new candidates for a complexity of $O(m^2 n)$, and also ensures the number of candidates does not exceed m. `extendLF` in Algorithm 3 can use cached candidate conditional entropies and re-weight them by w', leaving the $O(mn)$ computation of $U_{X,\bar{H},Y,w'}(\lambda')$ in each of a maximum of m loop iterations as the dominating factor. The complexity of computing the Boolean combination of features to produce a coverage vector c^λ for an LF added by these extensions is at most $O(mn)$, so it does not impact the overall complexity. Finally, `getWeights` in Algorithm 4 performs the $O(n)$ computation of IG for m features and each $\lambda \in \Lambda^f$, giving complexity $O(|\Lambda^f|mn)$, where $|\Lambda^f|$ is expected to be much smaller than m and entropies used to compute IG could be cached.

Witan's memory requirements are dominated by entropy matrix $\bar{H}$ and candidate coverage vectors $c^{\hat{\lambda}} \; \forall \; \hat{\lambda} \in \hat{\Lambda}$ that both require $O(mn)$ space, as well as cached candidate conditional entropies $H(x_{*j}|c^{\hat{\lambda}}) \; \forall \; j \in \{1, \ldots, m\}, \hat{\lambda} \in \hat{\Lambda}$ that require $O(m^2)$ space.

3.5.3 Qualitative Analysis. Figure 2 presents LFs generated by 10 iterations of Witan on bag-of-words features from three text classification datasets, colouring LFs that were accurate enough for a simulated user to assign a class label, as described in Section 4. Unlabelled LFs are shown in grey. "$\vee$"-separated keywords form disjunctive (OR) LF conditions, and nested LFs prefixed by "$\wedge$" represent conjunctions (ANDs) between parent and child LFs.

Overall, the LFs tend to use coherent sets of keywords with understandable meanings while also achieving desirable levels of coverage and accuracy. Note that while positive/negative sentiment analysis is the classification target for the IMDb reviews, unsupervised Witan produces a diverse range of LFs that could be used to classify reviews based on other concepts (e.g. by format: "movie" vs "episode"; or genre: "documentary" vs "animation"). In the dataset for discriminating painter and architect biographies Witan even begins to identify LFs associated with sub-classes of the intended target classes, such as "data/enterprise" architects. Finally, we note that Witan is able to capture a range of classes in the multi-class 20Newsgroups dataset, though only one LF is identified for the rarer "science" class, and no LFs are identified for the rare "politics" class after only 10 iterations.

IMDb Review Sentiment

- **negative:** waste∨worst (coverage: 13%, accuracy: 90%)
- documentary
- movie
  - **negative:** ∧ waste∨bad∨stupid∨crap (coverage: 22%, accuracy: 79%)
- **positive:** wonderful∨excellent∨superb (coverage: 14%, accuracy: 80%)
- films
- animation
- episodes∨episode
- **positive:** loved (coverage: 5%, accuracy: 75%)
- rent

Bias Bios: Painter or Architect

- **painter:** paintings∨painting (coverage: 25%, accuracy: 99%)
- **architect:** architecture∨development∨architect∨software∨architectural∨solutions∨management (coverage: 34%, accuracy: 95%)
- **painter:** art∨gallery∨artist (coverage: 30%, accuracy: 88%)
- **architect:** data∨enterprise (coverage: 5%, accuracy: 98%)
- **architect:** experience∨projects (coverage: 19%, accuracy: 86%)
- **painter:** paints∨life∨paint∨color (coverage: 15%, accuracy: 89%)
- **architect:** microsoft∨technologies∨business∨services∨applications∨web∨application (coverage: 13%, accuracy: 91%)
- **painter:** york (coverage: 8%, accuracy: 71%)
- **painter:** colors (coverage: 2%, accuracy: 97%)
- **painter:** images∨work∨arts∨born∨school∨landscape (coverage: 43%, accuracy: 70%)

20Newsgroups Topics

- nntp
- **computer:** thanks (coverage: 15%, accuracy: 57%)
  - **computer:** ∧ advance (coverage: 3%, accuracy: 68%)
- article
  - **religion:** ∧ rutgers (coverage: 3%, accuracy: 87%)
- **sports:** bike (coverage: 2%, accuracy: 100%)
- **computer:** hi∨windows∨pc∨graphics∨dos∨mac (coverage: 18%, accuracy: 80%)
- **sports:** hockey∨season∨baseball (coverage: 6%, accuracy: 94%)
- **science:** clipper (coverage: 3%, accuracy: 97%)
- **religion:** god∨christians∨christian∨jesus∨bible∨christianity∨religion∨waco∨jews∨christ∨religious∨church∨israel (coverage: 18%, accuracy: 56%)

Figure 2: Examples of Witan-generated LF sets

3.5.4 Potential Failure Modes. We now discuss scenarios where Witan's approach to generating LFs may produce a sub-optimal classifier. Recall that the intuition behind Witan's utility function presented in Section 3.1 is based on the assumption that some features will be correlated with target classes. Therefore, Witan is unlikely to produce accurate LFs if no features correlate with the target classes. Furthermore, as Witan favours LFs with high coverage, it may struggle to identify LFs for rare classes, such as in classification tasks with high imbalance or numerous target classes (as demonstrated with the 20Newsgroups LFs in Section 3.5.3 and in the multi-class experiments of Section 4.4). This issue may be addressed in some cases by increasing γ to prioritise fine-grained LFs with higher information gain over those with higher coverage or by providing feedback to focus Witan on LFs that are complementary to selected LFs. We also note that accurately representing rare classes is a general problem for weak supervision from coarse LFs, and discuss the potential for future work to augment Witan with instance-level labelling in Section 5.

4 EXPERIMENTAL STUDY

The following experiments compare Witan to state-of-the-art LF generation methods (IWS [10] and Snuba [53]), LFs produced by clustering methods (HDC [56] and CBI [20]), and common low-effort labelling approaches (semi-supervised learning [59] and active learning [31]). We demonstrate that, even without initial supervision, Witan is able to achieve competitive F1 scores on both binary and multi-class classification tasks. We also compare method runtimes and perform an ablation study to justify Witan's design decisions. All experiments were performed on 64-bit Ubuntu 20.04 running on a 12x2.60GHz Intel Core i7 CPU with 48GB of memory.

4.1 Experiment Framework

As the downstream goal of labelling is to produce an effective classifier, we evaluate the performance of a classifier trained using each labelling method. In each experiment, a given labelling method is applied to an unlabelled training dataset to produce labels for all or a subset of instances. For methods that produce LFs, probabilistic training labels are generated from the LFs using the Snorkel labelling model [49] under assumptions of independent LFs and balanced classes[1]. Labelled training instances are then embedded into 300 feature dimensions via truncated Singular Value Decomposition (SVD) and used to train a multilayer perceptron with two hidden layers of 20 units, RELU activations, softmax output, and logarithmic loss. This is the same discriminative model used to evaluate IWS [10], except the sigmoid output has been replaced with softmax to support multi-class classification. We evaluate classification performance by the mean F1 score achieved across target classes on the ground-truth labels of a held-out test set. While the AUC metric used in the evaluation of IWS [10] evaluates performance across a range of classification thresholds, we believe it is important to focus on the performance of the default threshold in the context of low-effort labelling, given that a user will likely not have access to labelled instances with which to tune the threshold. F1 also has a precedent in past evaluations of data programming [3, 50, 53].

To compare labelling methods under similar conditions of user effort, we specify a budget of "user interactions" (Interaction Count *IC*) that a method may leverage in each experiment. The form of each interaction varies depending on the nature of the method, but the user effort required for each interaction is approximately equivalent. For methods that accept input in the form of instance labels (Snuba, semi-supervised learning, and active learning), each provided label is considered a single interaction. For LF-based methods (Witan, IWS, HDC, and CBI), each LF the user is asked to approve or reject is counted as an interaction. For Witan and CBI, each interaction also includes the selection of the class label that an approved LF should assign. If the interaction budget is less than the number of candidate LFs, surplus interactions will effectively be unused. These user interactions are simulated in experiments based on ground truth labels for training instances: instance-based methods may query specific instance labels while LFs are approved if they would be at least 20% more accurate on the training dataset than an ideal random classifier for the target class (acc $\geq \frac{1}{k}+0.2$). As Witan, IWS, and CBI can also accept seed LFs, we perform seeded experiments with two seed LFs for each class based on randomly selected features with accuracy: $0.2 \leq (\text{acc} - \frac{1}{k}) \leq 0.35$. The described framework of counting interactions (specifically, equating instance labelling and LF approval), simulating users, and providing seed LFs is based on that used to evaluate IWS [10].

With the exception of IWS requiring the full set of classes for initialisation, the user simulated in this experimental framework is not expected to have prior knowledge of the class structure, but rather enough understanding of their classification task to identify

[1] Although this assumption will not hold for all datasets, it is a realistic operating condition when the user does not know the true class balance.

the correct class for a given instance or LF condition. While extensive understanding would be required to identify instance classes for instance-based methods or to identify classes for HDC's LF conditions (where the conditioned features increase with the number of LFs), we expect identifying classes for the LF conditions of Witan and CBI to require minimal prior knowledge, as discussed in Section 3.5.1. We justify this claim as well as including the selection of an LF's class label in each user interaction for Witan and CBI by arguing that if an LF condition is relevant enough to a proposed class for the user to approve an IWS-generated LF, then they should just as easily be able to identify the correct class for such an LF condition. Also note that, under this experimental framework, any of the evaluated methods may fail to label instances for one or more classes after a budgeted number of interactions, in which case the evaluated classifier will not be trained on those classes.

We present mean F1 scores over five random seedings of methods and seed LFs, except for unseeded Witan and HDC which do not depend on a random seed. Standard deviations are omitted for brevity but never exceed 0.3. The median standard deviation for each method across experiments in each of Table 3, Figure 4, and Figure 5 never exceeds 0.090, except for semi-supervised learning which never exceeds 0.135. Because runtimes are reported for non-parallelised experiments that are more time-consuming to execute, they represent a single random seeding.

4.1.1 Evaluated Methods. We evaluate Witan both with the extensions for conjunctive and disjunctive LFs described in Section 3.3 and without them (referred to as Witan-Core). Instead of stopping according to $C^{\min}$, Witan generates a number of LFs equal to the allowed *IC*, and each LF is either rejected or labelled by the simulated user. While both Witan variants include the feedback extension (Section 3.4), only seed LFs will be marked as `approved`. Without seed LFs, both variants will test Witan's ability to generate an entire set of LFs without supervision, while interactive user feedback is evaluated in our ablation study. Except as noted in the ablation study, Witan is run with $\gamma = 2$ and $c^{\min} = 0.02$.

We compare Witan to two state-of-the-art LF generation methods: IWS [10] and Snuba [53]. We base our implementations on each paper's accompanying source-code. For IWS, we experiment with the AS and LSE-AC ($\tilde{m} = 100$) LF acquisition strategies. As IWS's authors only describe binary classification, we exclude IWS from multi-class experiments. For Snuba, we generalise some binary-class assumptions in the original implementation to support multi-class classification, use decision stumps as heuristic models to generate reasonably interpretable LFs, and randomly sample a number of instances from the training set equal to *IC* for the input dataset. We use the formulation of ν in the paper over that in the codebase, as $m + 1$ in the divisor ensures ν is never negative. However, we retain some implementation details of the codebase: the range of $[0.25, 0.45]$ for β, which the codebase states achieves better results, and keeping three LFs from the first Snuba iteration. We also use the fixed number of 20 iterations implemented in the codebase rather than the stopping condition described in the paper because hyper-parameter values were not specified and initial experimentation with this method was less promising than fixed iterations.

We also evaluate using clustering to generate LFs that can be subsequently labelled by users. We evaluate hierarchical divisive clustering (HDC), implementing the foundational MGR algorithm [45] with improvements to select conditions based on MNIG and to always bisect the largest sub-cluster [56]. Unlike most clustering methods, HDC's clusters are described by feature conditions, though its iterative bisections result in complex conditions with up to as many features as there are clusters. We also evaluate clustering by intent (CBI) [20], which produces clusters with interpretable conditions, but relies on initial seeding and interactive user feedback. Our implementation trains the "residual" classifier on instances covered without disagreement by seed LFs and uses the same model as the downstream classifier. We use hyper-parameter values from the original paper, except we always take up to 1000 instances for the residual, capped at 50% of the training set for smaller datasets.

Finally, we compare against semi-supervised learning and active learning, which are commonly used approaches for learning with few labels. For semi-supervised learning, we apply label spreading [59] with a KNN kernel to a random sample of training instances equal to the allowed interaction count. We initialise active learning with random training instances until instances from at least two classes have been selected, and then iteratively apply uncertainty sampling [31] to query for additional training labels until the interaction budget is exhausted. In each uncertainty sampling iteration, the downstream classification model described above is trained with the current set of labelled training instances, and the training instance classified with least certainty is queried for its label.

4.1.2 Datasets. Table 2 lists the binary-class and multi-class text classification datasets used in our experiments. Text is transformed to word count feature representations using the pipeline from the IWS codebase [10]. We use binary bag-of-words features with at least 2% coverage of training instances as candidates for LF conditions in IWS, Snuba, and Witan. These feature counts are shown in parentheses next to the raw feature count in Table 2. The word `reuters` and other boilerplate text identifying the non-fake news source was removed from the FNS dataset. Many of these datasets have been used in other studies on weak supervision and data

Table 2: Binary and multi-class datasets used in experiments

	Description	Train n	k	m (cov. $\geq$ 2%)
IMD	IMDb review sentiment [35]	25,000	2	18,525 (791)
IMG	IMDb review drama/comedy	15,261	2	13.761 (770)
BPA	Painter/architect bio [15]	6,118	2	3,130 (246)
BPT	Professor/teacher bio [15]	12,294	2	4,858 (236)
BJP	Journalist/photographer bio [15]	16,129	2	5,597 (217)
BPP	Professor/physician bio [15]	27,238	2	6,752 (222)
AZN	Amazon review sentiment [27]	160,000	2	22,130 (160)
YLP	Yelp review sentiment [58]	19,000	2	7,868 (474)
PLT	Plots: action/romance [53]	973	2	211 (61)
FNS	Fake news identification [1]	22,449	2	21,179 (1,626)
BDB	DBPedia: politics/company [58]	40,022	2	9,161 (178)
BAG	AGNews: business/tech [58]	29,979	2	7,454 (137)
ATW	Airline tweet sentiment [14]	5,771	2	846 (60)
DMG	Identifying disaster tweets [40]	2,915	2	1,143 (91)
SPM	SMS spam identification [2]	2,786	2	439 (42)
TWN	20Newsgroups topics [30]	8,935	5	11,948 (799)
DBP	DBPedia categories [58]	56,000	14	12,682 (142)
AGN	AGNews topics [58]	60,000	4	12,494 (101)
NYT	NYT topics [39]	15,999	9	27,029 (2,372)

programming [10, 53, 55] and some feature in the recently proposed WRENCH benchmark datasets for weak supervision [57]. For consistency with other studies, we use predefined train/test splits where available, and use a random 50/50 split otherwise. All datasets except for ATW, SPM, TWN, and NYT have approximately balanced class distributions. DMG and SPM were retrieved from the UCI Machine Learning Repository [18]. IMG is the subset of IMD reviews for titles belonging to either, but not both, of the most common genres (Drama or Comedy); successfully performing different classification tasks on the same input data highlights Witan's ability to produce diverse LFs without initial supervision.

4.2 Binary-Class Experiments

Table 3 presents the performance of both unseeded and seeded labelling methods on binary classification tasks, including the baseline performance achieved by a classifier trained on the fully labelled training set. We present the F1 score achieved after 25 and 100 user interactions (*IC*), representing low and moderate user effort, respectively. The best performing methods for each dataset after each *IC* are highlighted in bold, revealing the following insights:

- For $IC = 25$, Witan is the best performing method. This low interaction setting is important not only because it requires less user effort, but also because a smaller set of LFs will be more understandable and maintainable.
- For $IC = 100$, Witan-Core is the best performing method. For all but three datasets, Witan-Core achieves F1 within 0.1 of the fully supervised baseline, highlighting the ability of Witan to train an accurate classifier with moderate labelling effort. We discuss performance differences between Witan and Witan-Core in our ablation study (Section 4.5).
- Active learning is the next best performer, particularly with more interactions. This suggests that the classification tasks where active learning performs best are better learned with specific example instances than with coarse LFs that cover many instances. However, we emphasise that LFs have the additional benefit over labelled instances of acting as an understandable and maintainable knowledge representation.
- As expected, seeded methods tend to perform better than their unseeded counterparts. In particular, IWS without seeding sometimes fails to generate any approved rules (F1 = 0) for $IC = 25$. However, this difference is often small, particularly after more interactions.

We compared methods across datasets using Friedman tests and accompanying Nemenyi post-hoc tests [16] for $IC = 25$ and $IC = 100$, and in both cases rejected the null-hypothesis that "all methods produce equivalent F1 scores" at the 95% confidence level. The critical difference diagrams in Figure 3 show groups of methods with statistically insignificant differences. Note that all Witan variants and active learning are in the top group for both values of *IC* and that seeded variants of Witan achieve the best overall performance.

Figure 4 further demonstrates how performance changes with *IC*. Note that the performance of most methods tends to converge after many interactions, but Witan achieves near-optimal performance with far fewer interactions than other methods. We also note that performance may drop at higher *IC* for LF-based methods if additional labelled LFs are of poorer quality or result in an incorrect class balance in instance labels; poor performance on even one class due to incorrect balance can greatly affect F1 scores.

Table 3: Binary classification F1 scores for unseeded and seeded labelling methods at 25 and 100 interactions (IC).

Method	IC	IMD	IMG	BPA	BPT	BJP	BPP	AZN	YLP	PLT	FNS	BDB	BAG	ATW	DMG	SPM
Full supervision		0.836	0.780	0.950	0.898	0.933	0.942	0.905	0.872	0.779	0.976	0.995	0.901	0.822	0.964	0.941
Witan	25	0.727	**0.710**	**0.923**	**0.848**	**0.892**	**0.860**	**0.772**	**0.737**	**0.675**	**0.900**	**0.979**	**0.777**	0.510	0.723	**0.816**
	100	0.753	0.768	0.931	0.817	0.887	0.875	0.779	0.795	0.624	0.905	0.949	0.708	0.550	0.690	0.781
Witan-Core	25	**0.738**	0.703	0.910	0.836	0.859	0.840	0.760	0.737	0.667	0.891	0.978	0.728	**0.632**	0.783	0.745
	100	**0.785**	**0.771**	**0.938**	0.833	**0.900**	0.848	**0.834**	**0.801**	0.719	**0.908**	**0.976**	0.764	0.594	0.846	0.843
IWS-AS	25	0.363	0.511	0.619	0.398	0.504	0.660	0.559	0.427	0.635	0.448	0.774	0.479	0.536	0.676	0.536
	100	0.575	0.746	0.815	0.477	0.845	0.835	0.796	0.565	**0.721**	0.391	0.965	0.688	0.607	0.808	0.805
IWS-LSE-AC	25	0.000	0.354	0.793	0.524	0.640	0.839	0.574	0.398	0.467	0.375	0.938	0.525	0.526	0.634	0.675
	100	0.656	0.594	0.850	0.509	0.826	0.850	0.790	0.651	**0.721**	0.476	0.964	0.668	0.604	0.809	0.805
Snuba	25	0.458	0.436	0.718	0.726	0.693	0.820	0.507	0.497	0.601	0.584	0.781	0.489	0.519	0.781	0.624
	100	0.501	0.521	0.844	0.763	0.759	0.830	0.651	0.668	0.631	0.796	0.862	0.647	0.503	0.783	0.598
HDC	25	0.000	0.388	0.663	0.746	0.688	0.802	0.576	0.495	0.596	0.788	0.753	0.749	0.403	**0.865**	0.777
	100	0.545	0.428	0.686	0.746	0.695	0.862	0.572	0.500	0.714	0.836	0.759	**0.806**	0.442	0.894	0.452
Semi-supervised	25	0.543	0.464	0.664	0.609	0.743	0.450	0.521	0.468	0.536	0.796	0.861	0.436	0.450	0.502	0.575
	100	0.607	0.599	0.551	0.567	0.491	0.590	0.700	0.467	0.558	0.642	0.913	0.592	0.546	0.789	0.571
Active learning	25	0.551	0.562	0.846	0.739	0.739	0.860	0.622	0.596	0.583	0.799	0.894	0.519	0.568	0.761	0.728
	100	0.666	0.642	0.917	**0.837**	0.845	**0.911**	0.787	0.745	0.684	0.895	0.973	0.780	**0.715**	**0.914**	**0.876**
Seeded Witan	25	**0.774**	**0.761**	**0.931**	0.834	**0.890**	**0.880**	0.742	**0.795**	**0.696**	**0.909**	**0.980**	**0.768**	0.544	0.759	0.696
	100	0.779	0.748	0.931	0.806	0.880	**0.881**	0.808	**0.802**	0.616	0.903	0.960	0.726	0.570	0.573	0.696
Seeded Witan-Core	25	0.756	0.756	0.926	**0.840**	0.863	0.838	**0.785**	0.768	0.680	0.893	0.974	0.756	**0.654**	**0.792**	**0.767**
	100	**0.790**	**0.763**	**0.941**	**0.822**	**0.900**	0.861	**0.835**	0.796	0.718	**0.915**	**0.976**	**0.770**	0.595	**0.846**	**0.843**
Seeded IWS-AS	25	0.703	0.657	0.752	0.576	0.644	0.763	0.646	0.594	0.678	0.649	0.899	0.600	0.645	0.716	0.637
	100	0.771	0.721	0.912	0.606	0.849	0.836	0.810	0.736	**0.721**	0.522	0.968	0.687	0.607	0.805	0.806
Seeded IWS-LSE-AC	25	0.333	0.393	0.877	0.652	0.705	0.842	0.632	0.443	0.659	0.521	0.942	0.610	0.621	0.761	0.725
	100	0.613	0.457	0.904	0.602	0.834	0.849	0.784	0.721	**0.721**	0.750	0.967	0.686	**0.607**	0.821	0.806
Seeded CBI	25	0.554	0.596	0.708	0.672	0.619	0.715	0.510	0.594	0.525	0.740	0.722	0.593	0.428	0.497	0.620
	100	0.617	0.603	0.796	0.744	0.781	0.778	0.507	0.653	0.525	0.809	0.926	0.620	0.428	0.497	0.620

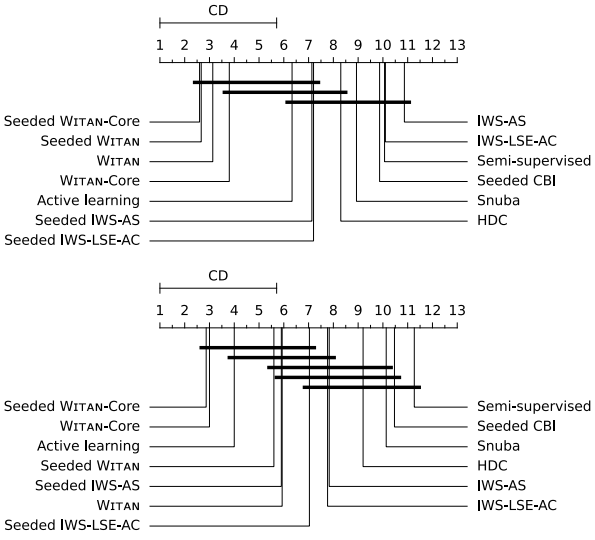

Figure 3: Critical difference diagrams for binary classification F1 of labelling methods at *IC* of 25 (above) and 100 (below).

4.3 Runtime Study

Table 4 compares the runtimes of labelling methods after 100 interactions, with methods faster and slower than Witan highlighted in blue and red, respectively. As the computational complexity of Witan does not depend on the number of classes, we only present runtimes for binary-class datasets. Without extending the set of candidate LFs with conjunctions and disjunctions, Witan-Core is slightly faster than Witan. CBI, Snuba, and semi-supervised learning are also faster in some cases, as their runtimes are not heavily dependent on the number of user interactions. Witan is generally faster than IWS and active learning, with the exception of the FNS dataset. Witan is much slower than other methods on FNS due to its large number of features m and because Witan's runtime scales with m^2. With the exception of this extreme case, Witan's maximum average runtime per user interaction is less than 4 seconds (for IMD), making it practical for use in an interactive setting.

4.4 Multi-Class Experiments

Figure 5 compares labelling methods on four multi-class classification tasks. All methods are unseeded, except for CBI which requires

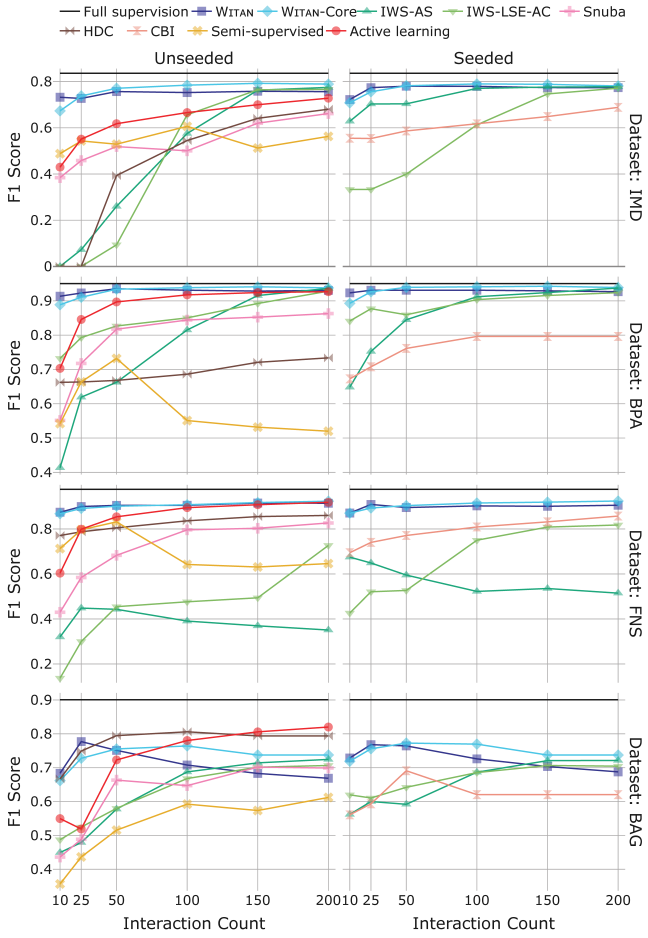

Figure 4: Binary classification results.

seed LFs for two random classes to train its "residual" classifier. In contrast to the binary classification results, Witan-Core begins to outperform Witan at lower *IC* and HDC, active learning, and semi-supervised learning are stronger performers, especially at higher *IC*. These results suggest that the smaller classes of multi-class problems are better captured by lower coverage LFs (e.g. those generated

Table 4: Runtime in seconds for labelling methods. Runtimes highlighted when slower than Witan by more than 10 or 60 seconds, or faster than Witan by more than 10 or 60 seconds.

Method	IC	IMD	IMG	BPA	BPT	BJP	BPP	AZN	YLP	PLT	FNS	BDB	BAG	ATW	DMG	SPM
Witan	100	310.4	173.8	17.0	23.7	23.0	50.1	105.8	98.0	0.2	1576.0	43.8	19.4	0.9	1.3	0.2
Witan-Core	100	333.2	187.6	13.3	19.2	19.0	26.4	57.3	93.6	0.3	1340.0	21.9	12.7	0.4	0.7	0.2
IWS-AS	100	626.0	559.9	612.7	621.2	617.0	620.8	668.9	632.8	577.6	653.6	641.3	636.2	583.0	618.2	472.5
IWS-LSE-AC	100	655.7	584.3	635.3	640.8	636.1	652.3	690.6	653.0	596.0	673.5	665.6	655.9	602.1	642.5	497.3
Snuba	100	158.7	144.4	48.6	48.6	45.0	46.7	58.6	95.0	14.7	326.2	41.8	31.8	14.9	20.3	3.9
HDC	100	423.4	250.7	11.0	19.1	22.0	19.9	83.8	133.5	0.3	1019.2	34.4	20.6	0.3	0.9	0.2
CBI	100	70.4	55.3	7.1	12.7	15.3	20.7	118.6	34.2	0.7	115.2	26.0	17.9	2.3	2.4	1.4
Semi-supervised	100	30.5	11.2	1.0	4.4	6.4	20.7	550.7	12.9	0.0	33.8	34.3	19.4	0.6	0.1	0.1
Active learning	100	927.8	682.0	203.5	272.4	303.1	355.1	987.5	432.5	40.9	1132.8	459.7	391.6	74.8	102.8	55.8

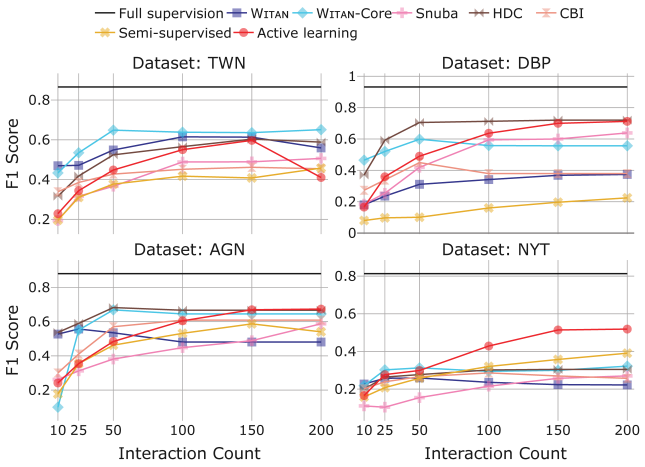

Figure 5: Multi-class classification results.

by HDC and Witan-Core, without disjunctions) or by labelling specific example instances (e.g. active and semi-supervised learning). An extreme example of this is the highly-imbalanced NYT dataset, where the smallest class contains only $\sim$ 1% of instances. However, we re-emphasise that semi-supervised and active learning do not produce interpretable LFs, and HDC LFs are difficult to interpret due to their many features. Overall, Witan-Core is still generally the best performer up to a moderate interaction count ($IC \leq 100$).

4.5 Ablation Study

Table 5 presents the results of our ablation study comparing variants of Witan across binary-class and multi-class datasets, highlighting methods that achieve an F1 score greater or less than Witan in blue and red, respectively. These results again highlight that while Witan outperforms Witan-Core at low IC for binary classification, Witan-Core is superior at higher IC and for multi-class datasets. From the *No ANDs* and *No ORs* variants that respectively disable conjunctive and disjunctive LFs, we can see that these differences are largely the result of disjunctive LFs. This confirms our observation that Witan-Core performs better in settings where narrower, more targeted LFs without disjunctions are more appropriate. The generally inferior results with $\gamma = 1$ justify our default setting of $\gamma = 2$ to prefer high information gain over wide coverage for LFs. Finally, we test Witan with interactive user feedback: we set $f^{\lambda} = \texttt{approved}$ for each generated LF λ if it is accurate enough to be assigned a class label by the simulated user. If an LF is not approved, conjunctive child LFs are not added to the candidate set. There is a strong performance benefit with interactive feedback for some datasets, though the performance drop for other datasets highlights the importance of exploring LFs that are not necessarily similar to already approved LFs in order to achieve diversity.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed Witan, a novel algorithm for the unsupervised generation of labelling functions (LFs) for data programming. The main advantage of Witan is its support for a variety of user interaction modes, most notably being able to effectively generate LFs without any interactive feedback, labelled instances, seed LFs, or even a predefined set of classes. Our experimental results demonstrate that Witan can be used to train a classifier that is competitive with alternative labelling methods on both binary and multi-class classification tasks, even without initial supervision.

This work also creates opportunities for future research. While we have experimented with bag-of-words LFs for text classification tasks, Witan could also be applied to other kinds of data with LFs based on characteristics of images or numeric features. Furthermore, given the hierarchical nature of Witan's conjunctive LFs, it may be beneficial to account for the inherent dependencies among LFs in the labelling model. Finally, based on past successes combining weak supervision with active learning [9, 42] and on our own positive results with instance-level labelling for smaller classes, we expect it could be beneficial for multi-class and imbalanced tasks to combine Witan with active-learning, allowing a small set of labelled instances to act as specific "exceptions" to coarser LFs.

ACKNOWLEDGMENTS

This research was funded by a Callaghan Innovation R&D Fellowship Grant (FPAP1902), for a Fisher & Paykel Appliances Ltd project. The first author is also funded by a Doctoral Fees Scholarship from Auckland University of Technology.

Table 5: F1 scores of Witan variants. Scores highlighted when less than baseline Witan by more than 0.01 or 0.05, or greater than baseline Witan by more than 0.01 or 0.05.

Method	IC	IMD	IMG	BPA	BPT	BJP	BPP	AZN	YLP	PLT	FNS	BDB	BAG	ATW	DMG	SPM	TWN	DBP	AGN	NYT
Witan	25	0.727	0.710	0.923	0.848	0.892	0.860	0.772	0.737	0.675	0.900	0.979	0.777	0.510	0.723	0.816	0.472	0.235	0.557	0.258
	100	0.753	0.768	0.931	0.817	0.887	0.875	0.779	0.795	0.624	0.905	0.949	0.708	0.550	0.690	0.781	0.615	0.341	0.481	0.236
Core	25	0.738	0.703	0.910	0.836	0.859	0.840	0.760	0.737	0.667	0.891	0.978	0.728	0.632	0.783	0.745	0.535	0.520	0.551	0.302
	100	0.785	0.771	0.938	0.833	0.900	0.848	0.834	0.801	0.719	0.908	0.976	0.764	0.594	0.846	0.843	0.638	0.558	0.645	0.294
No ANDs	25	0.772	0.729	0.936	0.858	0.897	0.877	0.572	0.737	0.675	0.906	0.973	0.775	0.501	0.734	0.816	0.537	0.223	0.507	0.256
	100	0.783	0.770	0.932	0.781	0.893	0.887	0.572	0.808	0.675	0.925	0.973	0.758	0.501	0.732	0.796	0.614	0.223	0.507	0.252
No ORs	25	0.741	0.703	0.910	0.836	0.859	0.830	0.760	0.737	0.667	0.891	0.978	0.728	0.645	0.783	0.745	0.542	0.529	0.551	0.300
	100	0.784	0.767	0.940	0.835	0.900	0.848	0.827	0.781	0.601	0.908	0.977	0.764	0.552	0.820	0.769	0.640	0.562	0.647	0.272
$\gamma = 1$	25	0.727	0.719	0.917	0.842	0.869	0.888	0.705	0.791	0.719	0.891	0.976	0.719	0.661	0.549	0.624	0.512	0.237	0.488	0.250
	100	0.744	0.746	0.928	0.833	0.865	0.901	0.777	0.782	0.684	0.907	0.969	0.722	0.554	0.449	0.624	0.572	0.277	0.477	0.255
Interactive	25	0.770	0.636	0.917	0.820	0.880	0.863	0.744	0.782	0.697	0.837	0.976	0.794	0.502	0.801	0.767	0.533	0.259	0.642	0.253
	100	0.754	0.683	0.922	0.788	0.863	0.882	0.842	0.783	0.657	0.818	0.974	0.753	0.494	0.702	0.738	0.514	0.330	0.633	0.244

REFERENCES

[1] Hadeer Ahmed, Issa Traore, and Sherif Saad. 2018. Detecting opinion spams and fake news using text classification. *Security and Privacy* 1, 1 (2018), e9.

[2] Tiago A Almeida, José María G Hidalgo, and Akebo Yamakami. 2011. Contributions to the study of SMS spam filtering: new collection and results. In *Proceedings of the 11th ACM symposium on Document engineering*. 259–262.

[3] Pedro Alonso Doval. 2021. *Strategies for the programmatic generation of labelled corpus for text classification.* Master's thesis. University of Vigo.

[4] Reem Alrashdi and Simon O'Keefe. 2020. Automatic labeling of tweets for crisis response using distant supervision. In *Companion Proceedings of the Web Conference 2020*. Association for Computing Machinery, New York, NY, USA, 418–425.

[5] Chidubem Arachie and Bert Huang. 2021. Constrained labeling for weakly supervised learning. In *Uncertainty in Artificial Intelligence*. PMLR, 236–246.

[6] Juhee Bae, Tove Helldin, Maria Riveiro, Sławomir Nowaczyk, Mohamed-Rafik Bouguelia, and Göran Falkman. 2020. Interactive clustering: a comprehensive review. *ACM Computing Surveys (CSUR)* 53, 1 (2020), 1–39.

[7] Benjamin Bergner and Georg Krempl. 2016. Active Subtopic Detection in Multi-topic Data. In *Proceedings of the Workshop Active Learning: Applications, Foundations and Emerging Trends, AL@iKNOW 2016*. 35–44.

[8] William Betham. 1834. *The Origin and History of the Constitution of England: And of the Early Parliaments of Ireland.* William Curry, Jun. And Co., 44.

[9] Samantha Biegel, Rafah El-Khatib, Luiz Otavio Vilas Boas Oliveira, Max Baak, and Nanne Aben. 2021. Active WeaSuL: Improving Weak Supervision with Active Learning. arXiv:2104.14847 [cs.LG]

[10] Benedikt Boecking, Willie Neiswanger, Eric Xing, and Artur Dubrawski. 2021. Interactive Weak Supervision: Learning Useful Heuristics for Data Labeling. In *International Conference on Learning Representations*.

[11] Salva Rühling Cachay, Benedikt Boecking, and Artur Dubrawski. 2021. Dependency Structure Misspecification in Multi-Source Weak Supervision Models. arXiv:2106.10302 [cs.LG]

[12] Peter Clark and Tim Niblett. 1989. The CN2 induction algorithm. *Machine Learning* 3, 4 (1989), 261–283.

[13] Benjamin Cohen-Wang, Stephen Mussmann, Alex Ratner, and Chris Ré. 2019. Interactive programmatic labeling for weak supervision. In *Proceedings of the KDD Data Collection, Curation, and Labeling for Mining and Learning Workshop*.

[14] Crowdflower. 2019. *Twitter US Airline Sentiment.* Retrieved July 4, 2022 from https://www.kaggle.com/crowdflower/twitter-airline-sentiment

[15] Maria De-Arteaga, Alexey Romanov, Hanna Wallach, Jennifer Chayes, Christian Borgs, Alexandra Chouldechova, Sahin Geyik, Krishnaram Kenthapadi, and Adam Tauman Kalai. 2019. Bias in bios: A case study of semantic representation bias in a high-stakes setting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*. 120–128.

[16] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research* 7 (2006), 1–30.

[17] Jun Du and Charles X Ling. 2010. Asking generalized queries to domain experts to improve learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 6 (2010), 812–825.

[18] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. Retrieved July 4, 2022 from http://archive.ics.uci.edu/ml

[19] Sara Evensen, Chang Ge, Dongjin Choi, and Çağatay Demiralp. 2020. Data Programming by Demonstration: A Framework for Interactively Learning Labeling Functions. arXiv:2009.01444 [cs.LG]

[20] George Forman, Hila Nachlieli, and Renato Keshet. 2015. Clustering by intent: a semi-supervised method to discover relevant clusters incrementally. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 20–36.

[21] Daniel Fu, Mayee Chen, Frederic Sala, Sarah Hooper, Kayvon Fatahalian, and Christopher Ré. 2020. Fast and three-rious: Speeding up weak supervision with triplet methods. In *International Conference on Machine Learning*. PMLR, 3280–3291.

[22] Johannes Fürnkranz and Peter A Flach. 2005. Roc 'n'rule learning—towards a better understanding of covering algorithms. *Machine Learning* 58, 1 (2005), 39–77.

[23] 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*. 2217–2225.

[24] Arnaud Giacometti and Arnaud Soulet. 2017. Interactive pattern sampling for characterizing unlabeled data. In *International Symposium on Intelligent Data Analysis*. Springer, 99–111.

[25] Maxim Grechkin, Hoifung Poon, and Bill Howe. 2018. EZLearn: Exploiting Organic Supervision in Automated Data Annotation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. 4085–4091.

[26] Braden Hancock, Paroma Varma, Stephanie Wang, Martin Bringmann, Percy Liang, and Christopher Ré. 2018. Training Classifiers with Natural Language Explanations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1884–1895.

[27] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th International Conference on World Wide Web*. 507–517.

[28] Giannis Karamanolakis, Daniel Hsu, and Luis Gravano. 2019. Leveraging Just a Few Keywords for Fine-Grained Aspect Detection Through Weakly Supervised Co-Training. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 4611–4621.

[29] David Kartchner, Wendi Ren, David Nakajima An, Chao Zhang, and Cassie S Mitchell. 2020. ReGAL: Rule-Generative Active Learning for Model-in-the-Loop Weak Supervision. In *NeurIPS 2020 Workshop on Human And Model in the Loop Evaluation and Training Strategies*.

[30] Ken Lang. 1995. Newsweeder: Learning to filter netnews. In *Machine Learning Proceedings 1995*. Elsevier, 331–339.

[31] David D Lewis and William A Gale. 1994. A sequential algorithm for training text classifiers. In *Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*. Springer, 3–12.

[32] Bin Lu, Myle Ott, Claire Cardie, and Benjamin K Tsou. 2011. Multi-aspect sentiment analysis with topic models. In *2011 IEEE 11th International Conference on Data Mining Workshops*. IEEE, 81–88.

[33] Zhipeng Luo and Milos Hauskrecht. 2018. Hierarchical active learning with group proportion feedback. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. 2532–2538.

[34] Zhipeng Luo and Milos Hauskrecht. 2019. Hierarchical Active Learning with Proportion Feedback on Regions. In *Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2018. Lecture Notes in Computer Science*, Vol. 11052. 464–480.

[35] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. 142–150.

[36] Neil Mallinar, Abhishek Shah, Tin Kam Ho, Rajendra Ugrani, and Ayush Gupta. 2020. Iterative Data Programming for Expanding Text Classification Corpora. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 13332–13337.

[37] Neil Mallinar, Abhishek Shah, Rajendra Ugrani, Ayush Gupta, Manikandan Gurusankar, Tin Kam Ho, Q Vera Liao, Yunfeng Zhang, Rachel KE Bellamy, Robert Yates, et al. 2019. Bootstrapping conversational agents with weak supervision. *Proceedings of the AAAI Conference on Artificial Intelligence* 33, 01 (2019), 9528–9533.

[38] Gary Marcus. 2018. Deep Learning: A Critical Appraisal. arXiv:1801.00631 [cs.AI]

[39] Yu Meng, Jiaming Shen, Chao Zhang, and Jiawei Han. 2018. Weakly-supervised neural text classification. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 983–992.

[40] Hussein Mouzannar, Yara Rizk, and Mariette Awad. 2018. Damage Identification in Social Media Posts using Multimodal Deep Learning. In *The 15th International Conference on Information Systems for Crisis Response and Management (ISCRAM)*. 529–543.

[41] Mona Nashaat, Aindrila Ghosh, James Miller, and Shaikh Quader. 2020. Asterisk: Generating Large Training Datasets with Automatic Active Supervision. *ACM Transactions on Data Science* 1, 2 (2020), 1–25.

[42] Mona Nashaat, Aindrila Ghosh, James Miller, Shaikh Quader, Chad Marston, and Jean-Francois Puget. 2018. Hybridization of active learning and data programming for labeling large industrial datasets. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 46–55.

[43] Thais Rodrigues Neubauer, Sarajane Marques Peres, Marcelo Fantinato, Xixi Lu, and Hajo Alexander Reijers. 2021. Interactive clustering: a scoping review. *Artificial Intelligence Review* 54, 4 (2021), 2765–2826.

[44] Andrew Ng. 2021. *MLOps: From Model-centric to Data-centric AI.* DeepLearning.AI. Retrieved November 24, 2021 from https://www.deeplearning.ai/wp-content/uploads/2021/06/MLOps-From-Model-centric-to-Data-centric-AI.pdf

[45] Hongwu Qin, Xiuqin Ma, Tutut Herawan, and Jasni Mohamad Zain. 2014. MGR: An information theory based hierarchical divisive clustering algorithm for categorical data. *Knowledge-Based Systems* 67 (2014), 401–411.

[46] John Ross Quinlan. 1986. Induction of decision trees. *Machine Learning* 1, 1 (1986), 81–106.

[47] Martin Rajchl, Matthew C. H. Lee, Franklin Schrans, Alice Davidson, Jonathan Passerat-Palmbach, Giacomo Tarroni, Amir Alansary, Ozan Oktay, Bernhard Kainz, and Daniel Rueckert. 2016. Learning under Distributed Weak Supervision. arXiv:1606.01100 [cs.CV]

[48] Parisa Rashidi and Diane J Cook. 2011. Ask me better questions: active learning queries based on rule induction. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 904–912.

[49] Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid Training Data Creation with Weak Supervision. *Proceedings of the VLDB Endowment* 11, 3 (2017), 269–282.

[50] Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data programming: Creating large training sets, quickly. *Advances in Neural Information Processing Systems* 29 (2016), 3567–3575.

[51] Simone Stumpf, Vidya Rajaram, Lida Li, Margaret Burnett, Thomas Dietterich, Erin Sullivan, Russell Drummond, and Jonathan Herlocker. 2007. Toward harnessing user feedback for machine learning. In *Proceedings of the 12th International Conference on Intelligent User Interfaces*. 82–91.

[52] Paroma Varma, Dan Iter, Christopher De Sa, and Christopher Ré. 2017. Flipper: A systematic approach to debugging training sets. In *Proceedings of the 2nd Workshop on Human-in-the-Loop Data Analytics*. 1–5.

[53] Paroma Varma and Christopher Ré. 2018. Snuba: Automating Weak Supervision to Label Training Data. *Proceedings of the VLDB Endowment* 12, 3 (2018), 223–236.

[54] Paroma Varma, Frederic Sala, Ann He, Alexander Ratner, and Christopher Ré. 2019. Learning dependency structures for weak supervision models. In *International Conference on Machine Learning*. PMLR, 6418–6427.

[55] Zihan Wang, Dheeraj Mekala, and Jingbo Shang. 2020. X-Class: Text Classification with Extremely Weak Supervision. arXiv:2010.12794 [cs.CL]

[56] Wei Wei, Jiye Liang, Xinyao Guo, Peng Song, and Yijun Sun. 2019. Hierarchical division clustering framework for categorical data. *Neurocomputing* 341 (2019), 118–134.

[57] Jieyu Zhang, Yue Yu, Yinghao Li, Yujing Wang, Yaming Yang, Mao Yang, and Alexander Ratner. 2021. WRENCH: A Comprehensive Benchmark for Weak Supervision. arXiv:2109.11377 [cs.LG]

[58] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *Advances in Neural Information Processing Systems* 28 (2015), 649–657.

[59] Dengyong Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. 2004. Learning with local and global consistency. In *Advances in Neural Information Processing Systems*. 321–328.