SNUBA REIMPLEMENTATION

Eric Chan, Arthur Liang, Shivam Singhal, Andrew Wei
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, WA 98195, USA
`{yee96, liangw6, shivam42, nowei}@cs.washington.edu`
Repo: `https://github.com/eric573/Snuba_2.0`

ABSTRACT

In this paper, we talk about our experience with reimplementing Snuba (Varma & Ré, 2018), a system that uses weak supervision to label unlabeled data based on a smaller labeled dataset. We include discussion on how our implementation differed from the original paper, improvements that we made, and also future directions.

1 INTRODUCTION

In the age of big data, we have (as the name suggests) a lot of data. Systems have been collecting metrics and data has been easier to get than ever before, allowing companies and people to make data-driven decisions. Machine learning and deep learning systems utilize this data for training, learning from the features and labels to improve models that perform some task. While we have a lot of data collected, much of it is useless unless we have corresponding labels for the data.

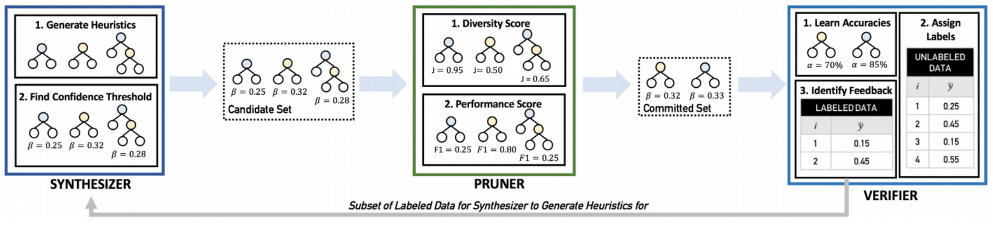
As such, it is important that we have labels for our data. The problem here is that labeling data is expensive and time-consuming, sometimes requiring subject-matter experts (SMEs) to label the data accurately for fields like medicine or the physical sciences. As such, the field of weak supervision has developed several schemes for labeling data.

One such scheme is Snuba (Varma & Ré, 2018), a weak supervision system for automatically labeling data via the use of automatically generated heuristics. In this paper, we will recount our experience with reimplementing Snuba.

2 ARCHITECTURE

The architecture of Snuba is split up into 3 phases:

1. **Synthesizer**: Generates heuristics
2. **Pruner**: Finds good heuristics
3. **Verifier**: Verify heuristics added and send unconfident points back to the Synthesizer for the next round


1. Generate Heuristics
2. Find Confidence Threshold
$\beta = 0.25$ $\beta = 0.32$ $\beta = 0.28$
SYNTHESIZER
Candidate Set
1. Diversity Score
J = 0.95 J= 0.50 J = 0.65
2. Performance Score
F1 = 0.25 F1 = 0.80 F1 = 0.25
PRUNER
$\beta = 0.32$ $\beta = 0.33$
Committed Set
1. Learn Accuracies
$\alpha = 70\%$ $\alpha = 85\%$
2. Assign Labels
UNLABELED DATA
3. Identify Feedback
LABELED DATA
VERIFIER
Subset of Labeled Data for Synthesizer to Generate Heuristics for

i	$\tilde{y}$
1	0.15
2	0.45

i	$\tilde{y}$
1	0.25
2	0.45
3	0.15
4	0.55

Figure 1: Snuba architecture.

2.1 Synthesizer

The Synthesizer takes in the labeled data and generates heuristics based on a combination of features. By heuristics, we mean simple models that are fast to train and easy to evaluate, like Logistic Regression (LR), Decision Trees (DT), or K-Nearest Neighbors (NN) along with a threshold, β, for determining certainty/confidence cutoffs. This is one of the most time-consuming parts of the labeling procedure, as it is a combinatorial search for good feature combinations, which in general takes longer than exponential time. As an example, even if we just have 384 features and we combine 2 features, then we would have to generate 73,536 different heuristic models and evaluate all of them to find the best heuristic. If we go to 3 features, this is 9,363,584 different heuristic models. Even if it takes 1ms to train each model, it would take 2.6 hours to find a single heuristic model. Thus, it is impractical to search for a heuristic combinatorially with too many features.

Algorithm 1: Synthesizer

```
Function GenerateHeuristics($f, X, y^*$)
Input: Heuristic Model $f \in \{DT, LR, NN\}$, (Subset of) Labeled Data $X \in \mathbb{R}^{N_L \times D}$, Labels
        $y^* \in \{-1, 1\}^{N_L}$
Output: Candidate set of heuristics $H$, feature combinations $X_{comb}$
$H, X_{comb} = [\,], [\,]$
for $D' = 1..D$ do
    // Generate feature combinations of size D'
    $idx_{comb}$ = GenComb($X, D'$)
    for $i = 1..len(idx_{comb})$ do
        $X' = X[:, idx_{comb}[i]]$
        $h = f(X', y^*)$
        $y_{prob}$ = predictProb($h, X'$)
        $\beta$ = FindBeta($y_{prob}, y^*$)
        $H = H \cup (h, \beta)$
        $X_{comb} = X_{comb} \cup idx_{comb}[i]$
return $H, X_{comb}$

Function FindBeta($y_{prob}, y^*$)
$betaList$ = [0.25, 0.3, 0.35, 0.4, 0.45, 0.5]
for $j = 1..len(betaList)$ do
    $\beta = betaList[j]$
    $F1[j]$ = calcF1($y^*, y_{prob}, \beta$)
return $betaList$[argmax($F1$)]

Function GenComb($X, D'$)
$D = X.shape[1]$ // # of features
return all combinations of size $D'$ from $D$
```

Note: highlighted text is what has changed from the original Snuba pseudocode. We will explain the significant changes in our Implementation section.

The β value is a threshold for when to determine whether we should leave something as labeled or unlabeled. $\hat{y}_i$ is the prediction for point i and β is used in the following manner:

$$\hat{y}_i = \begin{cases} 1 & \text{if } P[\hat{y}_i = 1] \geq 0.5 + \beta \\ 0 & \text{if } |P[\hat{y}_i = 1] - 0.5| < \beta \\ -1 & \text{if } P[\hat{y}_i = 1] \leq \beta - 0.5 \end{cases}$$

The Synthesizer basically generates the heuristics, assigns each heuristic the best β it can, and records the appropriate feature indices used for the heuristic and passes these to the Pruner.

To choose an appropriate β value, we need to weigh the trade-offs between coverage and accuracy. We do this through a proxy with the use of the F1 score, with Precision capturing some sense of accuracy and Recall capturing some sense of coverage. The F1 score is defined as follows:

- Precision (P): correctly labeled points over total points $\frac{\sum_{i=1}^{N_L} \mathbb{1}(\hat{y}_i = y_i^*)}{\sum_{i=1}^{N_L} \mathbb{1}(\hat{y}_i \neq 0)}$
- Recall (R): correctly labeled points over total number of points $\frac{\sum_{i=1}^{N_L} \mathbb{1}(\hat{y}_i = y_i^*)}{N_L}$
- F1 Score: harmonic mean of P and R, $2\frac{P \times R}{P+R}$

2.2 Pruner

The Pruner is in charge of ranking the heuristics based on two metrics, performance and coverage. We define performance as the F1 score and then the coverage as whether this heuristic labels points that haven't been labeled by any previous heuristic. We do this by applying the heuristic model and thresholding on both the labeled and unlabeled datasets. Then using the output for the labeled dataset, we calculate the F1 score. We then use the output on the unlabeled dataset to calculate whether this labels anything that hasn't been labeled before. We then store these score computations with the heuristic model and beta and sort them based on the score to determine the best heuristics in a given round. We then return the top $num\_heuristics$ of these.

Algorithm 2: Pruner

```
1  Function SelectBestHeuristic($(H, H_C, y^*, X_L, X_U, X_{comb}, n, w)$)
   Input: Candidate ($H$) and committed set of heuristics ($H_C$), Labels $y^* \in \{-1, 1\}^{N_L}$,
   Labeled dataset $X_L \in \mathbb{R}^{N_L \times D}$, Unlabeled dataset $X_U \in \mathbb{R}^{N_U \times D}$,
   combinations of indices $X_{comb}$, Labeled indicator $n \in \{0, 1\}^{N_U}$, score weighting $w \in [0, 0.5]$,
   number of heuristics to return, $num\_heuristics$
   Output: Set of best heuristics in candidate set, $h_{best} \subset H$
2  $h_{scores}$ = [ ]
3  for $i = 1..len(H)$ do
4      $h_i, \beta_i = H[i]$
5      $x_{comb,i} = X_{comb}[i]$
6      $\hat{y}_L^i$ = applyHeuristic($(h_i, X_L[:, x_{comb,i}])$)
7      $f_{score}$ = calcF1($(\hat{y}_L^i, y^*)$)
8      $\hat{y}_U^i$ = applyHeuristic($(h_i, X_U[:, x_{comb,i}])$)
9      $j_{score}$ = calcJaccard($(\hat{y}_U^i, n)$)
10     $h_{scores} = h_{score} \cup ((h_i, \beta_i), (1 - w) * j_{score} + w * f_{score}, h_i)$
11 Sort $h_{scores}$ by the score calculation
12 $h_{best}$ = select out $(h, \beta)$ from the $num\_heuristics$ best scores
13 Update $n$ based on confidence of heuristic for each of $(h, beta)$ in $h_{best}$
14 return $h_{best}$
```

2.3 Verifier

The Verifier tries to learn the confidence of the predictions based on the heuristics in the committed set. The Verifier's main job is to try and filter out the confident labels and sends the unconfident labels back into another loop through the Snuba procedure. The Snuba procedure ends in two ways: 1.) the generative model used to determine the confidence of the labels believes that it's not learning the confidence of the heuristics properly or 2.) we no longer have any unconfident points.

Algorithm 3: Verifier

```
1  Function FindFeedback($(H_C, y^*, X_L, X_U)$)
   Input: Committed set of heuristics $H_C$, True label of the training data $y^* \in \{-1, 1\}^{N_L}$,
          Labeled data $X_L$, Unlabeled data $X_U$
   Output: subset of labeled data $O'$, Probabilistic labels $\tilde{y}_U$
2  Train generative model on $X_L$ or $X_U$
3  $\tilde{y}_L$ = Predict probabilistic of $X_L$
4  $O'$ = Get uncertain points of labeled dataset where $|\tilde{y}_U^i - 0.5| \leq \nu$
5  return $O'$
```

2.3.1 Label Aggregator

Snuba uses a Naive Byes Model trained with Stochastic Gradient Descent for the Label Aggregator. The label aggregator is in charge of determining the confidence/accuracies of labels for the set of committed heuristics. Note that the heuristics were trained on the labeled dataset. Then the label aggregator trains the model over the output of the heuristics when applied on the unlabeled dataset. It takes all of the heuristic accuracies/outputs and calculates the expected probabilities of the labeled dataset to give us a good indication of the confidence these heuristics give based on an estimation of the correct assignments. Essentially, we map the heuristic outputs back from the generative model to see how well they actually capture patterns from the labeled dataset.

3 Implementation

Following the paper, we broke the implementation down into 3 parts, the Synthesizer, Pruner and Verifier. Our understanding of the Synthesizer is that it is a component that takes in black box heuristic models and systematically chooses some combination of the feature in the labeled set that the heuristic performs best in, with some degree of confidence. The Synthesizer then outputs each of these heuristics to be pruned by the Pruner.

The Pruner is a component that chooses the best heuristics to be added to the final committed heuristic set. The committed heuristics set is a set of heuristics which will then be used to generate labels for unlabeled data. In this case, the Pruner will then evaluate each of heuristics from the Synthesizer based on a weighted combination of F1 score and Coverage which is calculated using Jaccard Distance. The heuristics that achieves the best score is added to the committed heuristics. The committed heuristics is then passed into the Verifier.

The Verifier is a component that attempts to figure out how to aggregate the labels generated by the committed heuristics to generate a final label for each unlabeled point. The way this was achieved was to use an unsupervised model to find the best weighting. There are a high degree of freedom in terms of how to choose to aggregate the heuristics. For the purpose of the implementation, we chose to use the default label aggregator implemented by the paper.

In the process of implementing the pipeline, we made some key changes to the original proposed design in order to turn it into a production ready pipeline.

- β ranges for heuristic confidence

  During our initial implementation our β ranges between 0 to 0.5 (inclusive). However, when training, we observed that the Synthesizer would always pair the heuristics with $\beta = 0$, because there is no enforcement of confidence into each of the heuristics. The goal of beta is to enforce some form of confidence to the model itself, therefore, we decided to set a starting beta value of 0.25.

- Bootstrap initial stages of the pipeline with 3 heuristics

  The confidence threshold of set by the paper, $\nu = \frac{1}{2} - \frac{1}{(M+1)^\eta}$ is the confidence threshold. When we run the pipeline from scratch, the value of ν is very wide, leading to the pipeline to return no uncertain points at the first iteration. This makes sense as the value $M = 1$ allows for low confidence points to be classified as being confident. Therefore, we decided to change the implementation by bootstrapping the pipeline with 3 heuristics, leading to a tighter bound on ν that subsequently did not cause the pipeline to stop at iteration 1.

We also train the generative model on the labeled dataset instead of the unlabeled dataset as described in the Snuba. In practice, we have found both approaches to have comparable performance.

4 Improvements

We tried a couple of improvements to Snuba that the original paper mentioned. We played around with multiple heuristic models (simultaneously), different generative models for the label aggregator, and a different way to factor in heuristic diversity. We also discussed multi-class classification, using correctness and confidence for pruning the labaled dataset, and feature reduction.

4.1 Multiple Heursitic Models

Snuba relies on fixing the type of heuristic model (Logistic Regression, Decision Trees, or K-Nearest Neighbors); each instantiation of Snuba uses exaclty one of these model types. To be clear, this is not to say the same heuristic is utilized across feature sets and runs; each combination of primitive features in each iteration trains a new model, but of the same type.

We relaxed this requirement to allow all three model types to be considered. This entails training each of the three model types (LR, DT, NN) on each combination of primitives, for each iteration. We found that this does *not* take substantially longer to do, and that the training time is more affected by the number of features to consider in the primitive combinations (for reference, we used $D = 1$, as opposed to the original paper's $D = 4$ because of constrained compute). Now, the Synthesizer (Section 2.1) outputted 3 times as many heuristics. We did not have to change the Pruner, as it evaluates performance and diversity agnostic to the model type.

We found that individually run, Logistic Regression performed better than Decision Trees, on accuracy and coverage. Combining pairs of model types, we found that including K-Nearest Neighbors tanked the metrics. When combining Logistic Regression and Decision Trees, accuracy on both labeled and unlabeled datasets dropped by a percent, for a similar gain in coverage. Overall, we found that increasing heuristic types, at least at the level of LR vs DT helped increase coverage at the expense of accuracy (on the order of a percentage point). However, we hypothesize, based on our reading and understanding of Snuba, that the real gains come from increasing D to incorporate richer features in whatever heuristics do get trained.

4.2 Different Label Aggregator

The current generative model used in the Label Aggregator (Section 2.3.1) is a Naive Bayes Model. The original Snuba (Varma & Ré (2018)) code has some details as to how this works, but the basic idea as stated above is that the model takes in as features the heuristics' outputs on the labeled and unlabeled datasets. One thing to note before moving further is that the generative model is exactly that, a probability model. And so, that's how the probabilistic labels come to arise; they are simply the marginals of the probability distribution being modeled. The Snuba paper refers to this model as learning the accuracies, but specifically, what this model learns is how confident each heuristic is and how to weight them accordingly. In other words, the output of each heuristic is a feature (and each feature can take values -1, 0, 1) on a number of datapoints. The output/classes of the Naive Bayes model are "truth" values, which are basically just marginal probabilities; an output close to 0 or 1 means a "confident" prediction of -1 or 1, respectively, while an output close to 0.5 (within some threshold ν) entails an "unconfident" prediction, basically 0.

It is important to note that what seems to be happening in the original paper is "semi-supervised" parameter estimation. The basic idea, which is exactly what happens in Snuba, is to train a classifier on a labeled dataset (in our case the synthesizer generating heuristics trained on the labeled dataset), and then, in a loop, predicting class probabilities on the complete dataset (computing all marginals on both the heuristic outputs on the unlabeled dataset), and then using these probabilities to retrain the model (in this case training the label aggregator which uses the heuristics) based on the probabilities. So, this might seem a bit confusing, and definitely seemed that way to us. The basic gist is that the "Naive Bayes Classifier" in Snuba is a combination of the heuristics and the label aggregator. Namely, the label aggregator tries to maximize the likelihood of seeing the heuristic outputs on the unlabeled dataset, with the heuristics being trained on the labeled dataset.

In order to change this, we tried to change the model (from the existing label aggregator definition), to another model. We tried Restricted Boltzman Machines and Gaussian mixture models, which we thought would work due to their unsupervised nature. However, these models didn't work; some variations performed really poorly, others failed due to underflow errors. We then tried scikit-learn's Bernoulli Naive Bayes Model, which we realized after the fact wouldn't be a good fit because of the new model's reliance on the features being binary values (as opposed to -1, 0, 1). And, the BernoulliNB model is also a purely supervised model, at least as defined in the API, so it broke the paradigm of the label aggregator working without ground truth labels.

4.3 CHANGING HEURISTIC DIVERSITY

The Pruner determines its diversity score based on the Jaccard dissimilarity between a heuristic's predictions on the unlabeled dataset and a record of which examples in the unlabeled dataset have been given confident predictions (not necessarily correct). This is denoted as n, the "Labeled Indicator" in Section 2.2. In the original paper, setting an element of n to 1 means that *at least 1* heuristic confidently labeled that data point. We change n to record whether all heurisitcs collectively confidently label that datapoint.

This was done via changing n using the output of the Label Aggregator on the unlabeled dataset after the Verifier was run. Doing so factors in contributions from all heuristics. Our hope in doing this was to factor in disagreements between heuristics; say two heuristics both confidently label point i, but have conflicting labels. We wanted this to be reflected as a point on which the overall system was not confident on, so that another heuristic could be synthesized which takes care of this.

We found that this doesn't actually work. The number of entries in n grows consistently in the original version; in ours, it oscillates, but decreased in the "long run" (namely, over the 20 iterations of Snuba we ran in our experiments). Originally, n was equal to the unlabeled dataset coverage, but after our change, it was much smaller. So we didn't know whether we had to make some adjustments to the coverage calculation as well (we think not, but remain somewhat unsure). The change didn't improve coverage, and had a very small (we are thinking negligible) downside on accuracy.

4.4 FUTURE IMPROVEMENTS

Some areas of improvement we discussed but did not implement were multi-class classification, using correctness and confidence for pruning the labeled dataset and feature reduction. We found that Snuba already allowed multi-class classification, and showcases their results (albeit brief) in their appendix. We were mostly entertaining this possibility as a learning experience rather than an improvement, but thought we'd mention our thought process.

We also thought about the way the Verifier chooses which points of the labeled dataset to pass on to the next iteration. It does this with a confidence score which doesn't really take into account accuracy at all. This might be a deliberate choice on the author's part to prioritize coverage instead of accuracy, seeing as how coverage seems to be the harder metric to optimize. However, we were discussing changing this to include correctness. This would be a change in the last part of the Verifier (see line 4 of Algorithm 3).

Some other ideas we had, but did not really explore, were throwing away unintersting/unhelpful features to reduce search (e.g. finding β, pruning the candidate set, etc.), and, also trying some sort of boosting of unconfident/incorrect training labels over iterations.

5 EVALUATION

In this section, we present the results of our reimplementation of Snuba and show that our reimplementation has comparable performance to the original Snuba paper. Furthermore, we also provide analysis into Snuba and into our improvements in section 4.

5.1 EVALUATION SETUP

5.1.1 DATASET

For this project, we focus on the IMDb dataset. This text dataset contains movie descriptions, such as the plots of the movies and types of the movies. For the purposes of evaluation, we consider a simple classification problem of classifying the text of movie plots as action vs. romance movie types.

5.1.2 Implementation Details

Specifically, we first filter the dataset so it contains movies of either action or romance types. Then, we reserve 500 data points for testing. Finally, we take the remaining dataset and using a 4:1 split into a unlabeled and a labeled dataset (we have ground-truth labels for all data, but we do not use labels from unlabeled dataset during training to imitate having unlabeled data).

We further pre-process this dataset into bag-of-words features and filter out low-occurrence features. After this step, the remaining features, along with both the labeled and the unlabeled dataset, are the inputs to our reimplementation of Snuba.

Note that this is the same pre-processing step as done in the Snuba implementation (Varma & Ré, 2018).

5.1.3 Performance Metric

We compare performance in terms of both the heuristic's performance and the downstream performance. Note that since the Snuba pipeline has multiple heuristics, e.g. multiple logistic regression models, we refer to heuristic performance as the performance of the output of the label aggregator, which combines labels from different heuristics and outputs a single probabilistic label for each data point above a certain confidence threshold.

For heuristic performance, we mainly analyze two aspects, accuracy and coverage, where coverage is calculated as the fraction of points that the heuristics is confident enough to label and accuracy is the fraction of correctly labeled points among those that were labeled. Since the output of Snuba pipeline is labels for the unlabeled dataset, accuracy and coverage is tested on that unlabeled dataset (not the test dataset).

We also evaluate Snuba's performance in terms of the accuracy of a downstream LSTM models. While this indicates the overall performance of applying Snuba in practice, it also relies on the model, i.e. different model architectures, hyperparameters, or even the randomness of stochastic gradient descent might skew the results. For consistency, we use the same LSTM model and parameters/hyperparameters as in the original implementation of Snuba.

5.2 Results

5.2.1 Comparison with Original Snuba Implementation

As shown in figure 2, our re-implementation of Snuba shows comparable performance to the original Snuba. This verifies our understanding and reimplementation of the Snuba pipeline.

5.2.2 Multiple Heuristics Models

We also compare the performance of using different heuristic models, as shown in figure 3. We found that using a mixture of logistic regression and decision tree heuristics models had an accuracy of 0.767 and coverage of 0.601, which shows a trade-off of coverage over accuracy compared to just using the logistic regression model, which has an accuracy of 0.799 and coverage of 0.583.

6 Relatedness to the class

The basic/original premise of weak supervision is to use humans to create heuristics which can label large unlabeled datasets to produce more training data. The class has discussed various methods to do this, including the original (Snorkel), as well as similar paradigms such as semi-supervised learning. Snuba is one such take on weak supervision: why use humans at all when you have machines? Snuba, in summary, automates the heuristic generation part so that instead of a human, a small labeled dataset can prime the system. As such, it is a direct follow-up on the the original ideas of weak supervision, and is directly related to the class.

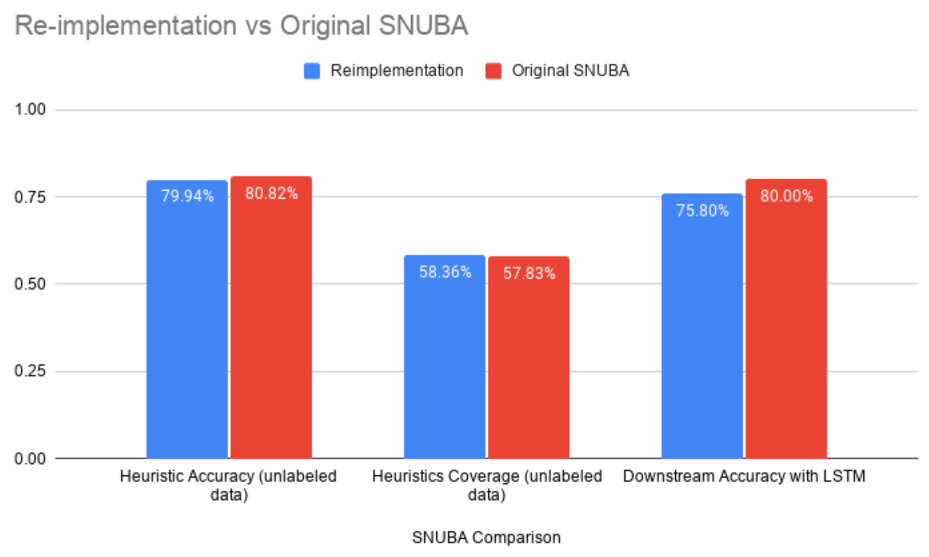


Figure 2: Re-implementation vs Original Snuba

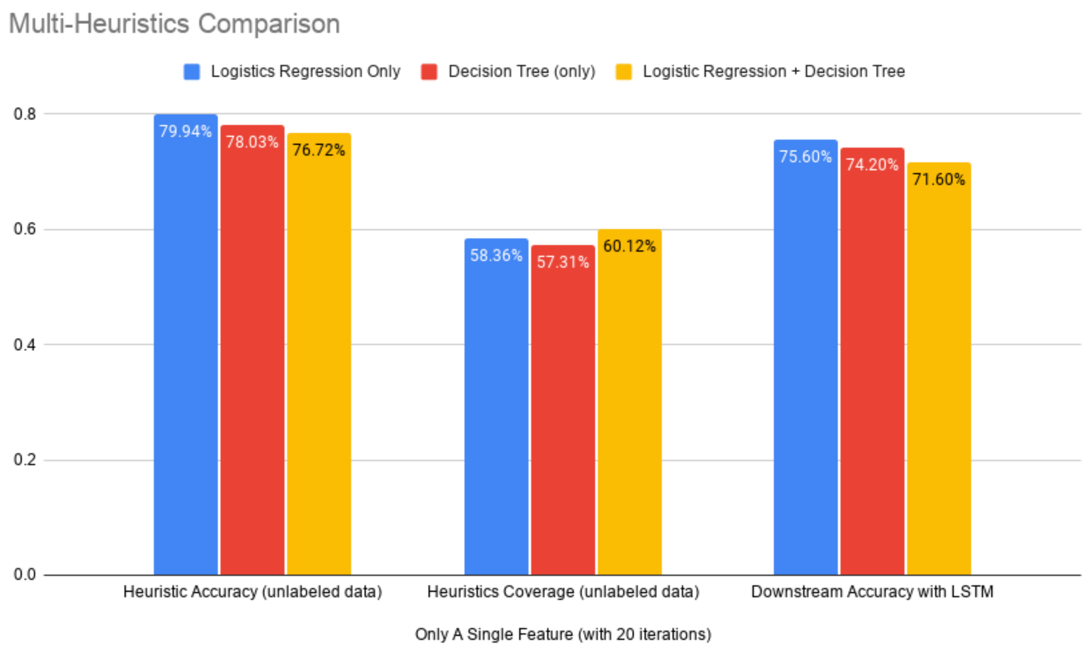


Figure 3: Multi-Heuristics Comparison

7 Conclusion

In this project, we reimplemented Snuba and noted the differences between what was described in the original paper and what was required for the actual implementation of Snuba. We also tried and suggested improvements. Additionally, we experimented with it and tested its labeling capabilities on the IMDb dataset.

Labeling data is a pain and it is expensive when scaled up or for areas where we require Subject Matter Expertise. Thus, automatic label generation is a worthwhile area to look into for modern machine learning pipelines.

References

Imdb dataset. URL `https://www.imdb.com/interfaces/`.

Paroma Varma and Christopher Ré. Snuba: Automating weak supervision to label training data. *Proc. VLDB Endow.*, 12(3):223–236, November 2018. ISSN 2150-8097. doi: 10.14778/3291264.3291268. URL `https://doi.org/10.14778/3291264.3291268`.