A Meta-Analysis of Existing Results

In this section, we present a meta-analysis of existing results to highlight the inconsistencies among them. Fig. 1 of (Gal et al., 2017) (replicated as Fig. 11 left) shows that BALD (Bayesian active learning by disagreement) using MC-dropout achieves much higher performance than the random baseline on one image dataset. By contrast, Fig. 2 of (Sinha et al., 2019) (replicated as Fig .11 right) shows that MC-Dropout methods struggle to outperform even the random baseline on four datasets.

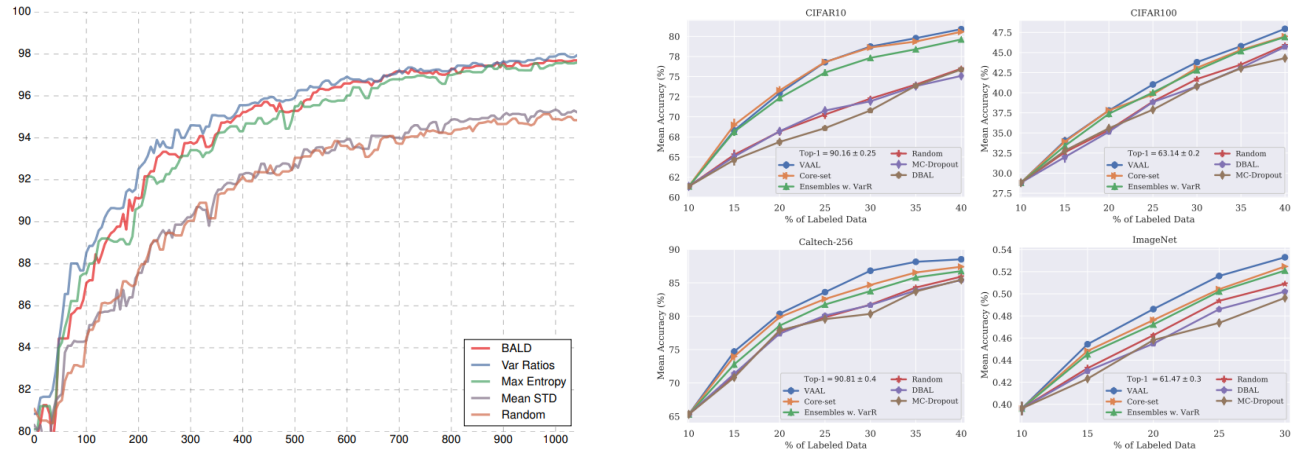

Figure 11: Left: Fig. 1 of (Gal et al., 2017). Right: Fig. 2 of (Sinha et al., 2019).

Fig. 2 of (Gal et al., 2017) (replicated as Fig. 12 top) shows that active learning strategies with MC-Dropout-based uncertainty estimation consistently outperforms their deterministic counterparts on a CV task. However, Fig. 4 of (Shen et al., 2018) (replicated as Fig. 12 bottom) finds no discernible difference between MC-Dropout-based BALD and other deterministic heuristics on an NLP task.

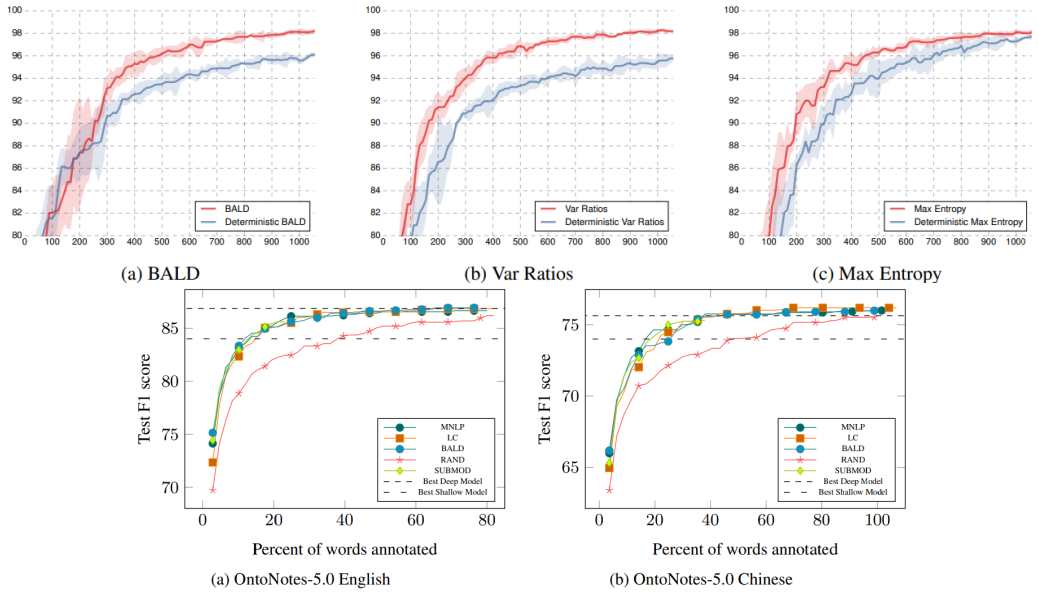

Figure 12: Top: Fig. 2 of (Gal et al., 2017). Bottom: Fig. 4 of (Shen et al., 2018).

For meta-active-learning methods, Fig. 3 of (Fang et al., 2017) (replicated as Fig. 13 top) shows that the RL-based PAL (policy active learning) consistently outperforms uncertainty and random baselines in a cross-lingual transfer setting. However, Fig .2 of (Liu et al., 2018) (replicated as Fig. 13 middle) shows that PAL are hardly better than uncertainty or random baselines in both cross-domain and cross-lingual transfer settings, while their proposed ALIL (active learning by imitation learning) is better than all studied baselines. Nevertheless, Fig. 4

of (Vu et al., 2019) (replicated as Fig. 13 bottom) again shows that both PAL and ALIL are not better than the uncertainty-based heuristic on a named entity recognition task, while their proposed active learning by dreaming method ranks the best.

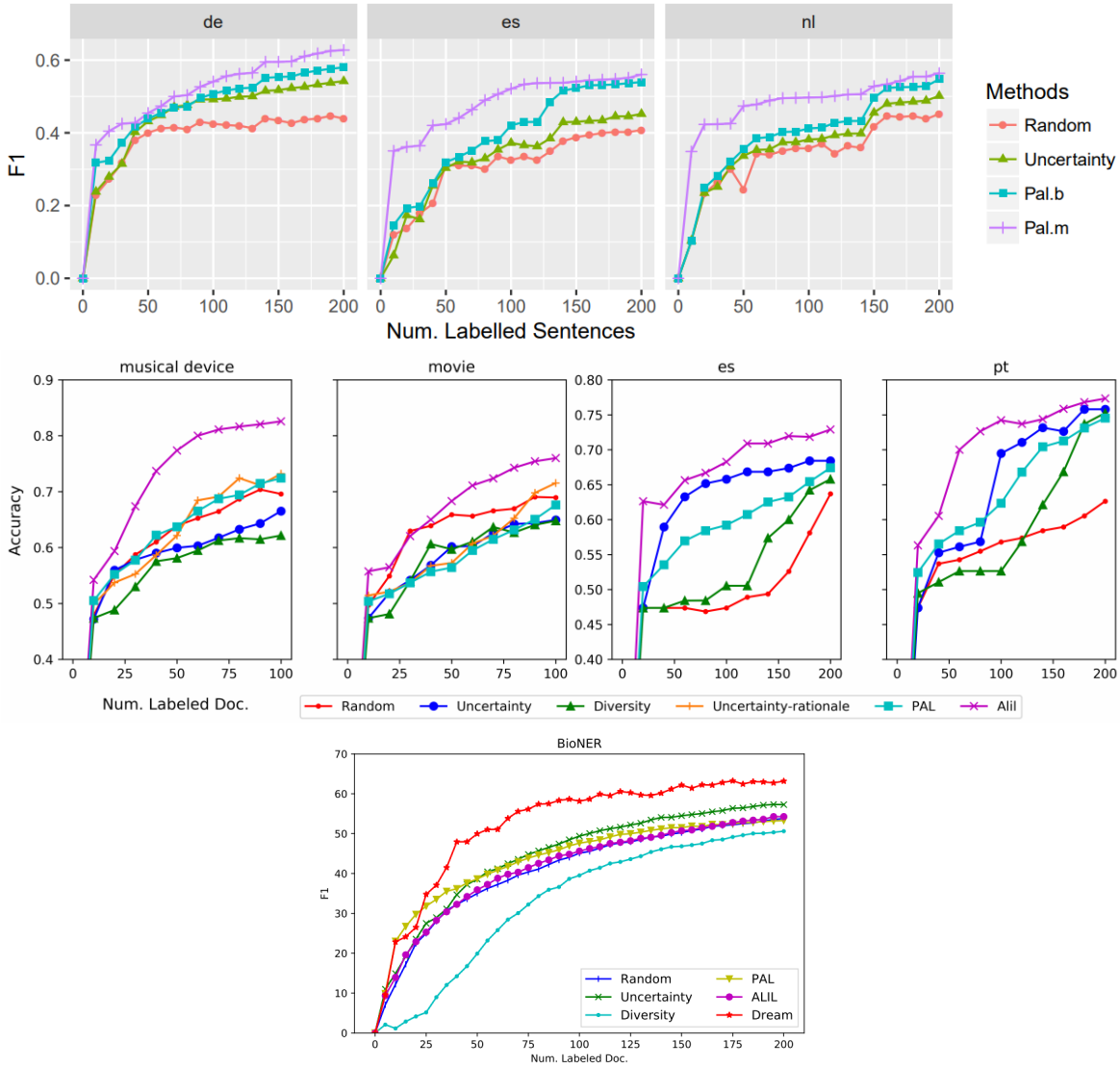

Figure 13: Top: Fig. 3 of (Fang et al., 2017). Middle: Fig .2 of (Liu et al., 2018). Bottom: Fig. 4 of (Vu et al., 2019).

These results demonstrate concerning inconsistency and lack of generalization to even very similar datasets or tasks. In addition, the above-presented performance curves are typically the sole analysis conducted by the authors. Thus, we believe that it is enlightening to analyze more aspects of a proposed strategy and its performance. In addition, a comparison with the optimal strategy could reveal actionable ways to improve the proposed strategy, which is the main motivation of our work.

B Considerations for Stochastic Acquisition Functions

It is straightforward to extend the discussion in Sec. 3 to stochastic acquisition functions, such as those that require stochastic uncertainty estimation, like BALD, or those that are represented as stochastic policies.

First, we introduce ζ to capture such randomness, so that $\Delta\mathcal{D}^L_{k+1} = \mathcal{A}(\theta_k, \mathcal{D}^L_k, \zeta)$ remains deterministic. Then we can establish a one-to-one relationship between $\mathcal{A}(\cdot, \cdot, \zeta)$ and order σ_ζ on $\mathcal{D}^U$. By definition of ξ-optimal order σ^*_ξ, we still have $q_\xi(\sigma^*_\xi) \geq q_\xi(\sigma_\zeta)$, and $q_\xi(\sigma^*_\xi) \geq \mathbb{E}_\zeta\left[q_\xi(\sigma_\zeta)\right]$. In other words, the ξ-optimal order σ^*_ξ outperforms

both the expected quality of the stochastic acquisition function and any single instantiation of stochasticity.

The above discussion suggests that not accounting or for stochasticity in acquisition function or unable to do so (i.e. using the expected quality) is inherently sub-optimal. Since many heuristics require uncertainty estimation through explicitly stochastic processes such as MC-Dropout (Gal and Ghahramani, 2016), the stochastic components may become the bottleneck to further improvement as AL algorithms approach the optimal limit.

C Model Architecture

Input: $28 \times 28 \times 1$
Conv: 32 5×5 filters
(Optional: MC-Dropout)
Max-Pool: 2×2
ReLU
Conv: 64 5×5 filters
(Optional: MC-Dropout)
Max-Pool: 2×2
ReLU
Linear: 1024×128
(Optional: MC-Dropout)
ReLU
Linear: 128×10

(a) CNN Object Class.

Input: L tokens
GloVe embedding: 300 dim
BiLSTM: 300 hidden dim
Average of L hiddens
(Optional: MC-Dropout)
Linear: 600×7

(b) LSTM Intent Class.

Input: L tokens
GloVe embedding: 300 dim
Conv: 50 size-3 kernels
ReLU
Conv: 50 size-3 kernels
ReLU
Conv: 50 size-3 kernels
ReLU
Average-Pool
Linear: 50×50
ReLU
(Optional: MC-Dropout)
Linear: 50×7

(c) CNN Intent Class.

Input: L tokens
GloVe embedding: 300 dim
Averege of L embeddings
Linear: 300×100
ReLU
Linear: 100×100
ReLU
(Optional: MC-Dropout)
Linear: 100×7

(d) AOE Intent Class.

Table 3: Model Architectures for Object and Intent Classification

C.1 Object Classification

We follow the architecture used by Kirsch et al. (2019). The MC-Dropout layers are present in BALD and BatchBALD for uncertainty estimation. The architecture is summarized in Tab. 3(a).

C.2 Intent Classification

We use an LSTM architecture as our main architecture of study for the intent classification task. We initialize the word embeddings with GloVe (Pennington et al., 2014) and jointly train it along with other model parameters. The architecture is shown in Tab. 3(b). Again, MC-Dropout is only used for uncertainty estimation in BALD.

To study model transfer performance, we also used (1D-)CNN and Average of Embedding (AOE) architectures, shown in Tab. 3(c) and (d), as well as the pre-trained RoBERTa architecture (Liu et al., 2019) with a linear classification layer on top.

C.3 Named Entity Recognition

For named entity recognition, we used the same architecture as Shen et al. (2018), except that we removed character-level embedding layer since an ablation study did not find it beneficial to the performance.

Specifically, the model architecture is encoder-decoder. The encoder builds the representation for each word in the same way as the BiLSTM architecture for intent classification. At each decoding step, the decoder maintains the running hidden and cell state, receives the concatenation of the current word encoder representation and previous tag one-hot representation as the input, and predicts the current tag with a linear layer from output hidden state. At the first step, a special `[GO]` tag was used.

D Extended Quality Summary

Tab. 4 presents the numerical quality scores for optimal, heuristic, and random orders. The object and intent classification tasks are evaluated and averaged across 5 different model seeds, and the NER task is evaluated on

a single model seed. In all cases, we can observe a large gap between the optimal order and the rest, indicating abundant room of improvement for the heuristics.

	Object Class.	Intent Class.		NER
Optimal	0.761	0.887	Optimal	0.838
Max-Entorpy	0.682	0.854	Min-Confidence	0.811
BALD	0.676	0.858	Norm. Min-Conf.	0.805
BatchBALD	0.650	N/A	Longest	0.793
Random	0.698	0.816	Random	0.801

Table 4: Complete results of optimal, heuristic, and random order qualities.

E Optimal Performance Curves

Fig. 14 shows the performance curves on the validation and test sets for intent classification (left two panels) and named entity recognition (right two panels). Similar to those for object classification (Fig. 3), the shape of the curves are quite similar on validation and test sets, indicating good generalization.

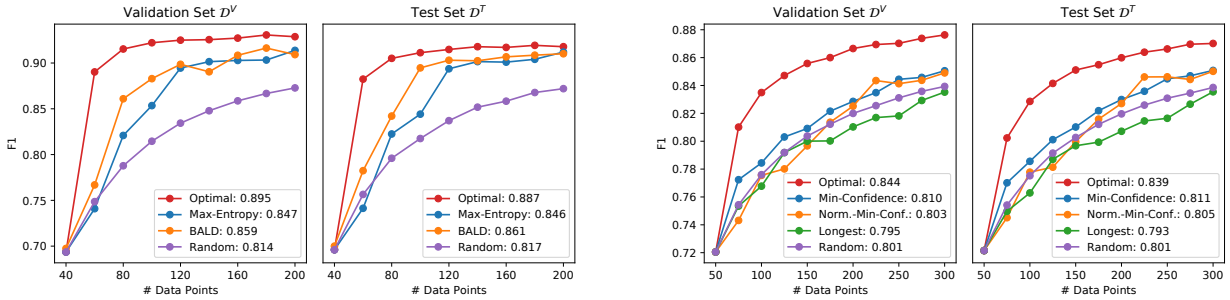

Figure 14: Performance curves for intent classification (left two panels) and NER (right two panels).

F Additional Visualizations for Object Classification

Fig. 15 presents the input (via PCA and t-SNE dimensionality reduction) and output distribution for BALD and BatchBALD order. The labels for data points sampled by the BALD and BatchBALD heuristics are much less balanced than the optimal and random orders. In addition, BatchBALD seems to focus on only a few concentrated regions in the input space.

G Output Distribution-Matching Regularization (ODMR)

In this section we describe the mixed results of regularizing the output distribution on object and intent classification. Since NER is a structured prediction task where the output tags in a sentence are correlated (e.g. I-tags always follow B- or I-tags), it is not clear what is the best way to enforce the distribution-matching property in the tag space and we leave the investigation to future work.

Alg. 3 presents the OMDR algorithm. Compared to the IDMR algorithm (Alg. 2), there are two additional challenges. First, the labels for the pool set are not observed, so when the algorithm tries to acquire a data point with a certain label, it can only use the predicted label, which, with few data, can be very wrong in some cases. As soon as a data point is selected for annotation, its label is revealed immediately and used to calculate d_{cur} during the next data point selection. In other words, data annotations are not batched.

Second, due to the unavailability of pool labels, the only labels for which we can use to compute the reference distribution are from $\mathcal{D}_0^L$ and $\mathcal{D}^M$. If they are small, as are typically, the reference label distribution can be far from the true distribution $\mathbb{P}_Y$, especially for rare classes. Thus, we employ the add-1 smoothing (i.e. adding 1 to each label count before calculating the empirical count distribution, also known as Laplace smoothing) to ensure that rare classes are not outright missed.

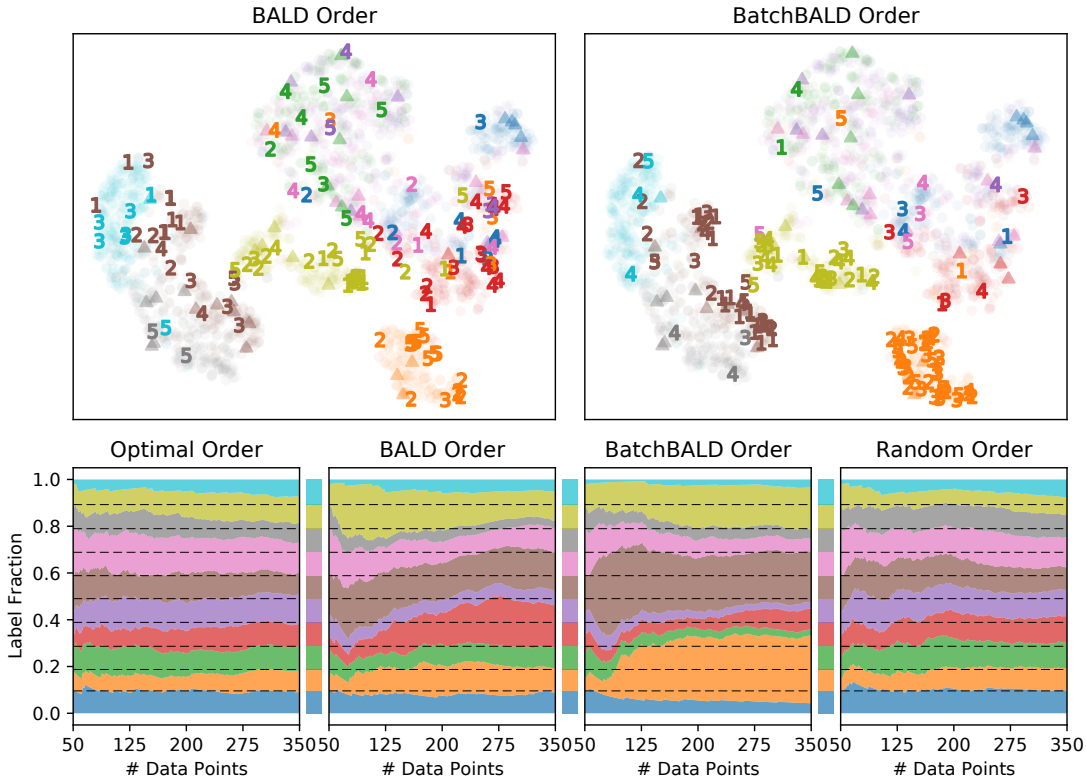

Figure 15: Top: the first five batches numerically labeled in t-SNE embedding space, along with warmstart (triangle) and test (circle) samples. Bottom: label distribution w.r.t. labeled set size, with test-set distribution shown between plots and as dashed lines.

Algorithm 3: Output Distribution-Matching Regularization (IDMR)

Input: $\mathcal{A}\left(\mathcal{D}^U, m_\theta, \mathcal{D}^L\right)$ that returns the next data point in $\mathcal{D}^U$ to label

$d_{\text{ref}} = \texttt{label-distribution}\left(\mathcal{D}^L_{0,Y} \cup \mathcal{D}^M_Y, \texttt{add-one=True}\right)$;

$d_{\text{cur}} = \texttt{label-distribution}\left(\mathcal{D}^L_Y, \texttt{add-one=False}\right)$;

$l^* = \arg\min_l \left(d_{\text{cur}} - d_{\text{ref}}\right)_l$;

$\mathcal{D}^U_{l^*} = \{x \in \mathcal{D}^U : m_\theta(x) = l^*\}$;

return $\mathcal{A}\left(\mathcal{D}^U_{b^*}, m_\theta, \mathcal{D}^L\right)$

In addition to Alg. 3, we also experiment with three "cheating" versions of ODMR. The most extreme version (test+groundtruth) uses test set (rather than the accessible set $\mathcal{D}^L_0 \cup \mathcal{D}^M$) to estimate label distribution and the groundtruth label (rather than the predicted label) are used to construct the subset $\mathcal{D}^U_{l^*}$. The other two versions (accessible+groundtruth and test+predicted) drops either piece of information. The actual "non-cheating" ODMR (accessible+predicted) is the only version that is practically feasible in reality.

Fig. 16 presents the four versions of ODMR-augmented heuristics on the intent classification result. We can see that all four versions are able to outperform the baseline heuristic, and three out of four (including the "non-cheating" ODMR) are able to outperform the random baseline. In addition, the label distribution are much more uniform compared to the unregularized version in Fig. 7 (bottom middle).

Fig. 17 presents the four versions of ODMR-augmented max-entropy (top) and BALD (bottom) heuristics on the intent classification result. Unlike object classification, none of the "cheating" or "non-cheating" versions are able to outperform the respective vanilla heuristic. Interestingly, using predicted rather than groundtruth labels achieves better performance in both cases.

Overall, the results are inconclusive about whether ODMR can help. Since it uses the predicted label to guide data acquisition, this kind of bootstrapping is prone to "confident mistakes" when a data point is predicted

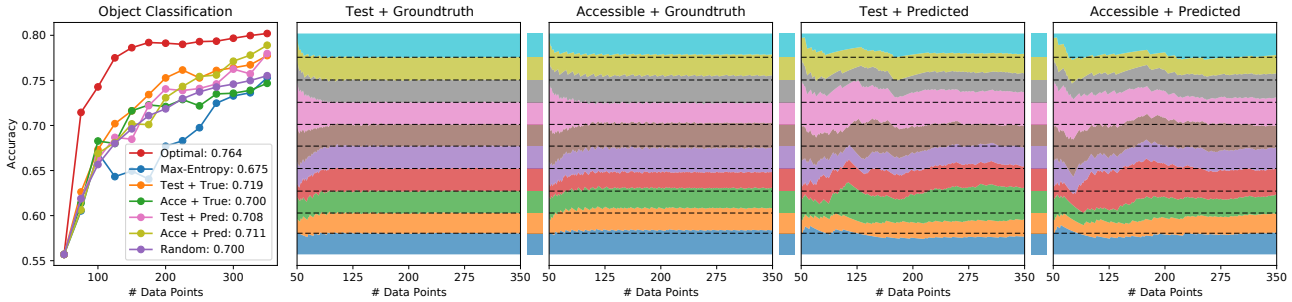

Figure 16: ODMR with max-entropy heuristic on object classification along with observed label distribution.

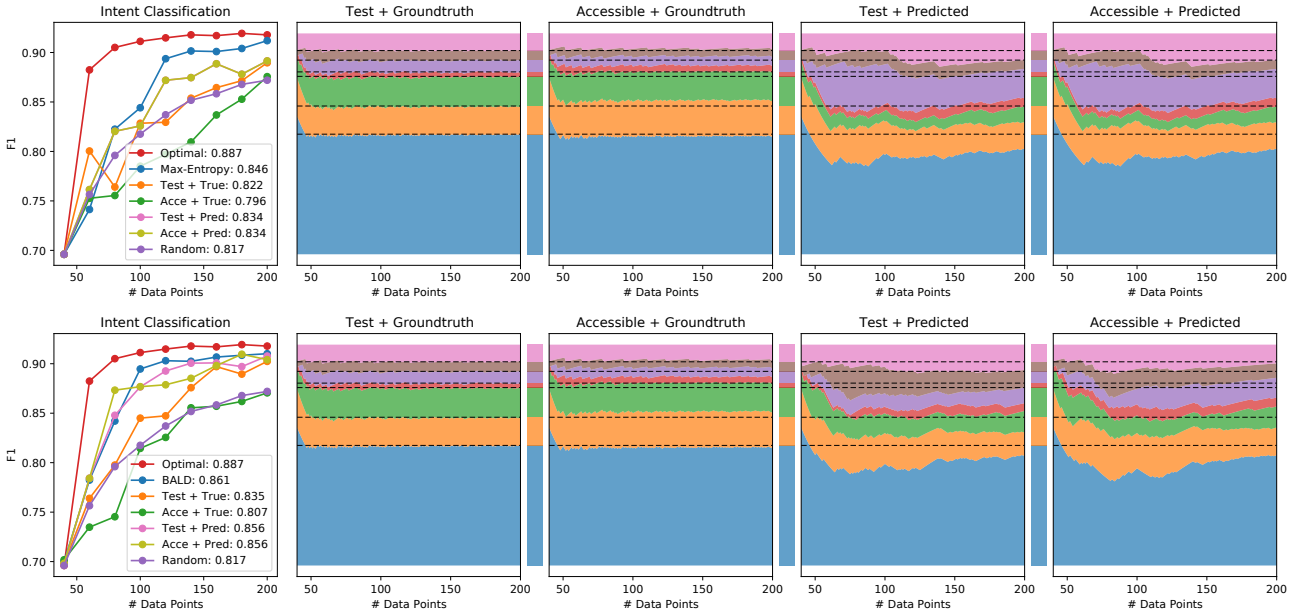

Figure 17: ODMR with max-entropy (top) and BALD (bottom) heuristics on intent classification along with observed label distribution.

wrong but with high certainty and thus never acquired, while the selection mechanism of IDMR provides a much more independent, yet still heuristic-guided, way to cover the data distribution. Visually, ODMR performance curves for both tasks are much less smooth than IDMR curves (Fig. 10), indicating unstability of using output labels to regularize a prediction task. Additional studies are required to determine when and how ODMR can help, and we leave them to future work.