A DATASETS AND MODEL ARCHITECTURES

A.1 BEER REVIEW

Data We use the BeerAdvocate review dataset (McAuley et al., 2012) and consider three *binary* aspect-level sentiment classification tasks: LOOK, AROMA and PALATE. This dataset was originally downloaded from https://snap.stanford.edu/data/web-BeerAdvocate.html.

Lei et al. (2016) points out that there exist strong correlations between the ratings of different aspects. In fact, the average correlation between two different aspects is 0.635. These correlations constitute as a source of biases when we apply predictors to examples with conflicting aspect ratings (e.g. beers that looks great but smells terrible).

We randomly sample 2500 positive examples and 2500 negative examples for each task. We apply `ls` to identify non-generalizable splits across these 5000 examples. The average word count per review is 128.5.

Representation backbone Following previous work (Bao et al., 2021), we use a simple text CNN for this dataset. Specifically, each input review is encoded by pre-trained FastText embeddings Mikolov et al. (2018). We employ 1D convolutions (with filter sizes $3, 4, 5$) to extract the features Kim (2014). We use 50 filters for each filter size. We apply max pooling to obtain the final representation ($\in \mathbb{R}^{150}$) for the input.

Predictor The Predictor applies a multi-layer perceopton on top of the previous input representation to predict the binary label. We consider a simple MLP with one hidden layer (150 hidden units). We apply ReLU activations and dropout (with rate 0.1) to the hidden units.

Splitter The Splitter concatenates the CNN representation with the binary input label. Similar to the Predictor, we use a MLP with one hidden layer (150 ReLU units, dropout 0.1) to predict the splitting decision $\mathbb{P}(z_i \mid x_i, y_i)$. We note that the representation backbones of the Splitter and the Predictor are *not shared* during training.

A.2 TOX21

Data The dataset contains 12,707 chemical compounds. Each example is annotated with two types of properties: Nuclear Receptor Signaling Panel (AR, AhR, AR-LBD, ER, ER-LBD, aromatase, PPAR-gamma) and Stress Response Panel (ARE, ATAD5, HSE, MMP, p53). The dataset is publicly available at http://bioinf.jku.at/research/DeepTox/tox21.html.

Representation backbone Following Mayr et al. (2016), we encode each input molecule by its dense features (such as molecular weight, solubility or surface area) and sparse features (chemical substructures). There are 801 dense features and 272,776 sparse features. We concatenate these features and standardize them by removing the mean and scaling to unit variance.

Predictor The Predictor is a multi-layer perceptron with three hidden layers (each with 1024 units). We apply ReLU activations and dropout (with rate 0.3) to the hidden units.

Splitter The Splitter concatenates the molecule features with the binary input label. Similar to the Predictor, we use a multi-layer perceptron with three hidden layers (each with 1024 units). We apply ReLU activations and dropout (with rate 0.3) to the hidden units.

A.3 WATERBIRD

Data This dataset is constructed from the CUB bird dataset (Welinder et al., 2010) and the Places dataset (Zhou et al., 2017). Sagawa et al. (2019) use the provided pixel-level segmentation information to crop each bird out from the its original background in CUB. The resulting birds are then placed onto different backgrounds obtained from Places. They consider two

types of backgrounds: water (ocean or natural lake) and land (bamboo forest or broadleaf forest). There are 4795/1199/5794 examples in the training/validation/testing set. This dataset is publicly available at https://nlp.stanford.edu/data/dro/waterbird_complete95_forest2water2.tar.gz

By construction, 95% of all waterbirds in the training set have water backgrounds. Similarly, 95% of all landbirds in the training set have land backgrounds. As a result, predictors trained on this training data will overfit to the spurious background information when making their predictions. In the validation and testing sets, Sagawa et al. (2019) place landbirds and waterbirds equally to land and water backgrounds.

For identifying non-generalizable splits, we apply `ls` on the training set and the validation set. For automatic de-biasing, we report the average accuracy and worst-group accuracy on the official test set. To compute the worst-group accuracy, we use the background attribute to partition the test set into four groups: `waterbirds` with water backgrounds, `waterbirds` with land backgrounds, `landbirds` with water backgrounds, `landbirds` with land backgrounds.

Representation backbone Following previous work (Sagawa et al., 2019; Liu et al., 2021a), we fine-tune torchvision's `resnet-50`, pretrained on ImageNet (Deng et al., 2009), to represent each input image. This results into a 2048 dimensional feature vector for each image.

Predictor The Predictor takes the `resnet` representation and applies a linear layer (2048 by 2) followed by Softmax to predict the label ($\{$`waterbirds`, `landbirds`$\}$) of each image. Note that we reset the Predictor's parameter to the pre-trained `resnet-50` at the beginning of each outer-loop iteration.

Splitter The Splitter first concatenates the `resnet` representation with the binary image label. It then applies a linear layer with Softmax to predict the splitting decision $\mathbb{P}(z_i \mid x_i, y_i)$. The `resnet` encoders for the Splitter and the Predictor are not shared during training.

A.4 CelebA

Data CelebA (Liu et al., 2015) is a large-scale face attributes dataset, where each image is annotated with 40 binary attributes. Following previous work (Sagawa et al., 2019; Liu et al., 2021a), we consider our task as predicting the blond hair attribute ($\in \{$`blond_hair`, `no_blond_hair`$\}$). The CelebA dataset is available for non-commercial research purposes only. It is publicly available at https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html.

While there are lots of annotated examples in the training set (162,770), the task is challenging due to the spurious correlation between the target blond hair attribute and the gender attribute ($\in \{$`male`, `female`$\}$). Specifically, only 0.85% of the training data are blond-haired males. As a result, predictors learn to utilize `male` as a predictive feature for `no_blond_hair` when we directly minimizing their empirical risk.

For identifying non-generalizable splits, we apply `ls` on the official training set and validation set. For automatic de-biasing, we report the average and worst-group performance on the official test set. To compute the worst-group accuracy, we use the gender attribute to partition the test set into four groups: `blond_hair` with male, `blond_hair` with female, `no_blond_hair` with male, `no_blond_hair` with female.

Representation backbone Following previous work (Sagawa et al., 2019; Liu et al., 2021a), we fine-tune torchvision's `resnet-50`, pretrained on ImageNet (Deng et al., 2009), to represent each input image. This results into a 2048 dimensional feature vector for each image.

Predictor The Predictor takes the `resnet` representation and applies a linear layer (2048 by 2) followed by Softmax to predict the label ($\{$`blond_hair`, `no_blond_hair`$\}$) of each image. Note that we reset the Predictor's parameter to the pre-trained `resnet-50` at the beginning of each outer-loop iteration.

Splitter The Splitter concatenates the `resnet` representation with the binary image label. It then applies a linear layer with Softmax to predict the splitting decision $\mathbb{P}(z_i \mid x_i, y_i)$. The `resnet` encoders for the Splitter and the Predictor are not shared during training.

A.5 MNLI

Data The MultiNLI corpus contains 433k sentence pairs (Williams et al., 2018). Given a sentence pair, the task is to predict the entailment relationship (`entailment`, `contradiction`, `neutral`) between the two sentences. The original corpus splits allocate most examples to the training set, with another 5% for validation and the last 5% for testing. In order to accurately measure the performance on rare groups, Sagawa et al. (2019) combine the training and validation set and randomly shuffle them into a 50/20/30 training/validation/testing split. The dataset and splits are publicly available at `https://github.com/kohpangwei/group_DRO`.

Previous work (Gururangan et al., 2018; McCoy et al., 2019) have shown that this crowd-sourced dataset has significant annotation artifacts: negation words (nobody, no, never and nothing) often appears in `contradiction` examples; sentence pairs with high lexical overlap are likely to be `entailment`. As a result, predictors may over-fit to these spurious shortcuts during training.

For identifying non-generalizable splits, we apply `ls` on the training set and validation set. For automatic de-biasing, we report the average and worst-group performance on the testing set. To compute the worst-group accuracy, we partition the test set based on whether the input example contains negation words or not: `entailment` with negation words, `entailment` without negation words, `contradiction` with negation words, `contradiction` without negation words, `neutral` with negation words, `neutral` without negation words.

Representation backbone Following previous work (Sagawa et al., 2019; Liu et al., 2021a), we fine-tune Hugging Face's `bert-base-uncased` model, starting with pre-trained weights (Devlin et al., 2018).

Predictor The Predictor takes the representation of the `[CLS]` token (at the final layer of `bert-base-uncased`) and applies a linear layer with Softmax activations to predict the final label (`entailment`, `contradictions`, `neutral`). Note that we reset the Predictor's parameter to the pre-trained `bert-base-uncased` at the beginning of each outer-loop iteration.

Splitter The Splitter concatenates the representation of the `[CLS]` token with the *one-hot* label embedding ($\in \{0, 1\}^3$). It then applies a linear layer with Softmax activations to predict the splitting decision $\mathbb{P}(z_i \mid x_i, y_i)$. The `bert-base-uncased` encoders for the Splitter and Predictor are not shared during training.

B Implementation Details

B.1 Identifying non-generalizable splits using `ls`

Optimization For Beer Review, Tox21, Waterbirds and CelebA, we update the Splitter and Predictor with the Adam optimizer (Kingma & Ba, 2014). In Beer Review and Tox21, the learning rate is set to 10^{-3} with no weight decay (as we already have dropout in the MLP to prevent over-fitting). We use a batch size of 200. In Waterbirds and CelebA, since we start with pre-trained weights, we adopt a smaller learning rate 10^{-4} (Sagawa et al., 2019) and set weight decay to 10^{-3}. We use a batch size of 100. For MNLI, we use the default setting for fine-tuning BERT: a fixed linearly-decaying learning rate starting at 0.0002, AdamW optimizer (Loshchilov & Hutter, 2017), dropout, and no weight decay. We use a batch size of 100.

Stopping criteria For the Predictor's training, we held out a random 1/3 subset of $\mathcal{D}^{\text{train}}$ for validation. We train the Predictor on the rest of $\mathcal{D}^{\text{train}}$ and apply early-stopping when the validation accuracy stops improving in the past 5 epochs. For the Splitter's training, we compare the average loss $\mathcal{L}^{\text{total}}$ of the current epoch and the average loss across the past 5 epochs. We stop training if the improvement is less than 10^{-3}.

B.2 AUTOMATIC DE-BIASING

Method details We use the Splitter learned by `ls` to create groups that are informative of the biases. Specifically, for each example (x_i, y_i), we first sample its splitting decision from the Splitter $\hat{z}_i \sim \mathbb{P}(z_i \mid x_i, y_i)$. As we have seen in Figure 4, these splitting decisions reveal human-identified biases. Similar to the typical group DRO setup (Sagawa et al., 2019), we use these information together with the target labels to partition the training and validation data into different groups. For example in Waterbirds, we have four groups: $\{y = \texttt{waterbirds}, z = 0\}, \{y = \texttt{waterbirds}, z = 1\}, \{y = \texttt{landbirds}, z = 0\}, \{y = \texttt{landbirds}, z = 1\}$. We minimize the worst-group loss during training and measure the worst-group accuracy on the validation data for model selection. Specifically, we stop training if the validation metric hasn't improved in the past 10 epochs.

Optimization Modern neural networks are usually highly over-parameterized. As a result, they can easily memorize the training data and over-fit the majority groups even when we minimize the worst-group loss during training. Following Sagawa et al. (2019), we apply strong regularization to combat memorization and over-fitting. We grid-search over the weight decay parameter $(10^0, 10^{-1}, 10^{-2}, 10^{-3}, 0)$.

B.3 COMPUTING RESOURCES

We conducted all the experiments on our internal clusters with NVIDIA A100, NVIDIA RTX A6000, and NVIDIA Tesla V100.

C ADDITIONAL ANALYSES

Ablation study What would happen if we don't have the regularity constraints? Figure 7 presents our ablation analyses on Beer Look and Tox21. We observe that `ls` produces challenging splits w/ and w/o the regularity constraints. However, the resulting splits are very different:

- Regularizer Ω_2 is necessary to learn meaningful splits. When we remove Ω_2, `ls` learns to create training and testing splits with vastly different label distributions (Figure 7.c). For example in the training split of Beer Look, 61% of the examples are positive. In contrast, only 9% of the testing split is positive.
- Regularizer Ω_1 stabilizes the train/test ratio across datasets. When we remove Ω_1, the resulting size ratio between $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{test}}$ varies a lot across different datasets ($89\% : 11\%$ in Beer Look and $69\% : 31\%$ in Tox21). This is not surprising as biases may be distributed differently for different datasets.

Time complexity Table 2 presents the running time of `ls`. For MNLI, `ls` takes 27 hours to finish on a single GPU. Compared to empirical risk minimization, `ls` needs to perform second-order reasoning, and this brings in the extra time cost.

Improving the scalability of `ls` is crucial for large-scale applications. As we have emphasized in Section 4.2, `ls` requires training a new Predictor for each outer-loop iteration. While it guarantees faithfulness of the Predictor (to the current splits), this procedure can be costly for large datasets. Exploring training-free predictors will be a promising future direction to reduce time cost.

Multiple bias sources Real-world applications often have multiple bias sources. In CelebA, we observe that the same train-test split that correlates with the gender attribute (Figure 4) also correlates with other unknown attributes (Figure 8). In other words, `ls` combines minority groups from different bias sources ({`Blond_hair` with `Male`}, {`Blond_hair` with `Hat`}, {`Blond_hair` with `eyeglasses`} and {`Blond_hair` with `Blurry`}) to form its challenging testing split $\mathcal{D}^{\text{test}}$. Disentangling different bias sources will be an interesting future direction.

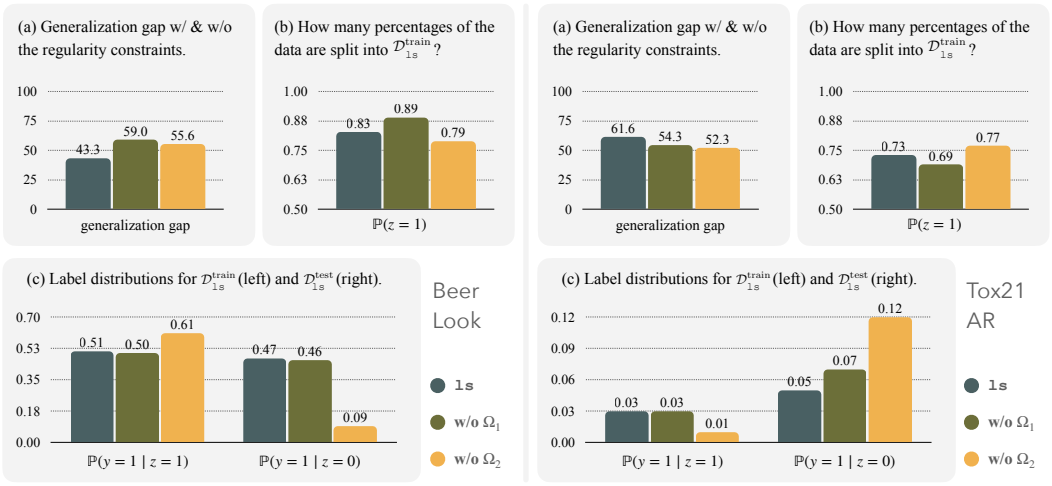

Figure 7: Ablation study of the two regularizer: Ω_1 (controlling the size ratio between $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{test}}$) and Ω_2 (balancing the label distributions across $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{test}}$). Three takeaways: (a) `ls` produces challenging splits regardless of the regularity constraints; (b) Without Ω_1, the size ratio between $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{test}}$ varies a lot across datasets ($89\% : 11\%$ in Beer Look and $69\% : 31\%$ in Tox21); (c) Without Ω_2, the learned splits exhibit huge label imbalance across $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{test}}$.

Table 2: Time cost of `ls` on a single V100 GPU. The results here are obtained from a shared internal server. Many other factors (such as cpu utilization, I/O and network) may impact the running time.

	Tox 21	Beer Look	Beer Aroma	Waterbirds	CelebA	MNLI
Data size (#annotations)	10,240	15,000	15,000	11,788	202,599	412,349
Model size (#parameters)	3.7M	1.9M	1.9M	23.5M	23.5M	109.4M
Time cost for `ls` (Algorithm 1)	0.6 hour	1.5 hours	2 hours	6.1 hours	5.6 hours	27.0 hours

D APPLICATION: LABEL NOISE DETECTION

In the presence of label noise, `ls` can reach high generalization gap by allocating all clean examples to the training split and all mislabeled examples to the testing split. Here we verify the effectiveness of `ls` as a label noise detector.

Data We consider the standard digit classification dataset MNIST as our test bed (LeCun & Cortes, 2010). The dataset is freely available at `http://yann.lecun.com/exdb/mnist/`.

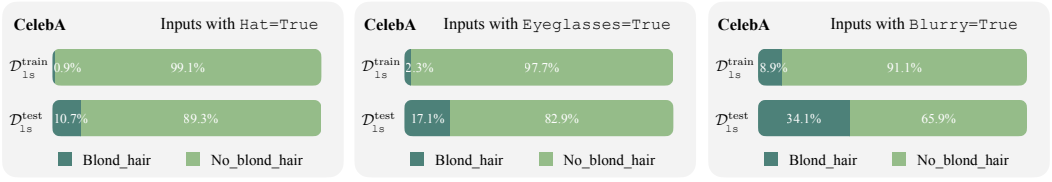

Figure 8: The dataset CelebA contains annotations for multiple face attributes. We observe that the same train-test split in Figure 4 correlates with other unknown attributes such as `Hat` (left), `Eyeglasses` (mid) and `Blurry` (right).

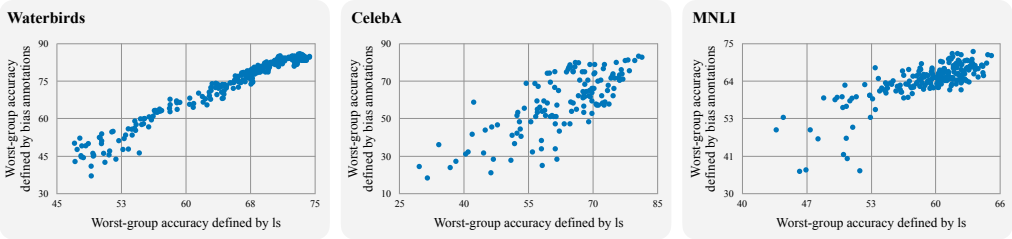

Figure 9: The spurious splits identified by `ls` provide a surrogate metric for model selection when biases are not known a priori.

We use the official training data (10 classes and 60,000 examples in total) and inject random label noise. Specifically, for a given noise ratio η, we sample a noisy label $\tilde{y}$ of each image based on its original clean label y:

$$\mathbb{P}(\tilde{y}) = \begin{cases} 1-\eta & \text{for } \tilde{y} = y, \\ \eta/9 & \text{for } \tilde{y} \neq y. \end{cases}$$

We apply `ls` to identify spurious splits from this noisy data collection for various η. In practice, we cannot assume access to the noise ratio. Therefore we keep $\delta = 0.75$ (Eq 1) as in our other experiments.

Representation backbone We follow the architecture from PyTorch's MNIST example[3]. Each input image is passed to a CNN with 2 convolution layers followed by max pooling (2×2). The first convolution layer has 32 filters, and the second convolution layer has 64 filters. Filter sizes are set to 3×3 in both layers.

Predictor The Predictor is a multi-layer perceptron with two hidden layers (each with 100 units). We apply ReLU activations and dropout (with rate 0.25) to the hidden units.

Splitter The Splitter concatenates the CNN features with the one-hot input label. Similar to the Predictor, we use a multi-layer perceptron with two hidden layers (each with 100 units). We apply ReLU activations and dropout (with rate 0.25) to the hidden units.

Optimization The optimization strategy is the same as the one for Tox 21 (Section B.1).

Evaluation metrics Given the spurious splits produced by `ls`, we can evaluate its precision and recall of identifying the polluted annotations:

- Precision = #polluted annotations in the testing split / #annotations in the testing split;
- Recall = #polluted annotations in the testing split / #polluted annotations.

We note that `ls` controls the train-test split ratio through the regularity constraint (Ω_1 in Eq 1). For ease of optimization, this constraint is implemented as a *soft* regularizer. As a result, the train-test split ratio needs to compete with other objectives (such as maximizing the generalization gap), and the resulting split ratio can be different for different noise ratios. To better understand the precision and recall, given the train-test split ratio produced by `ls`, we define an oracle which allocates as many polluted annotations as possible into the testing split:

- if #polluted annotations $\leq$ #annotations in the testing split:
  - Oracle precision = #polluted annotations / #annotations in the testing split;
  - Oracle recall = 100%;
- else:
  - Oracle precision = 100%;
  - Oracle recall = #annotations in the testing split / #polluted annotations.

[3] https://github.com/pytorch/examples/blob/master/mnist/main.py

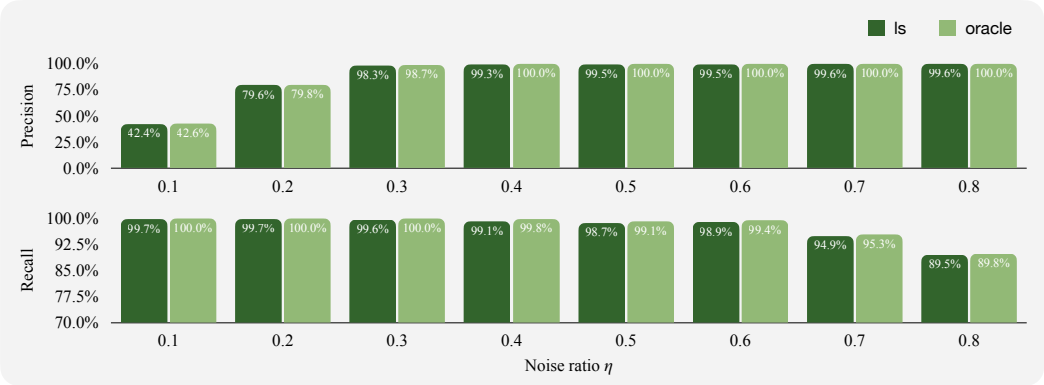

Figure 10: `ls` identifies label noise. X-axis: portion of the data that is mis-labeled.

Results We consider different noise ratios and present our results in Figure 10. We observe that `ls` consistently approaches to the oracle performance. When the noise ratio η is small $(0.1, 0.2)$, `ls` allocates most of the polluted annotations (¿99%) into the testing split. However, to meet the train-test ratio constraint, it also has to include some of the clean annotations into the testing split (non-perfect precision). When the noise ratio η is large $(0.7, 0.8)$, the testing split consists of mostly polluted annotations ($> 99\%$). The recall is not perfect because allocating all polluted annotations to the testing split will violate the regularity constraint.

E Additional Discussions

Why do we need a parametric Splitter? One may wonder if we can directly use the prediction correctness to create the final split (instead of learning the Splitter). The answer is no, and there are two reasons: 1) It may not satisfy the regularity constraints; 2) It ignores under-represented examples in the original training split that we used to train the Predictor. In this work, we parameterize the splitting decisions through a learnable mapping, the Splitter. This encourages similar inputs to receive similar splitting decisions, and we can also easily incorporate different constraints into the learning objective (Eq 3). Moreover, by iteratively refining the Predictor based on the updated Splitter, we obtain more challenging splits (Figure 6).

Algorithm 2 ls: learning to split (detailed version)

Input: dataset $\mathcal{D}^{\text{total}}$
Output: data splits $\mathcal{D}^{\text{train}}$, $\mathcal{D}^{\text{test}}$

Initialize *Splitter*, $\mathbb{P}_{Splitter}(z_i \mid x_i, y_i)$, from scratch or from pre-trained representations (such as ResNet, BERT, etc.)

Set `outer_iter` = 0.
repeat
  # Step 1: create train/test splits from $\mathcal{D}^{total}$
  if `outer_iter` is 0 **then**
    Split $\mathcal{D}^{\text{total}}$ into $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{test}}$ uniformly at random.
  else
    Set $\mathcal{D}^{\text{train}}, \mathcal{D}^{\text{test}} = \emptyset, \emptyset$.
    for each input label pair (x_i, y_i) in $\mathcal{D}^{\text{total}}$ **do**
      Sample $z_i \sim \mathbb{P}_{Splitter}(z_i \mid x_i, y_i; w)$.
      Add (x_i, y_i) into $\mathcal{D}^{\text{train}}$ if $z_i = 1$. Otherwise add it into $\mathcal{D}^{\text{test}}$.
    end for
  end if

  # Step 2: train and evaluate Predictor based on the current splits.
  Initialize *Predictor*, $\mathbb{P}_{Predictor}(y_i \mid x_i)$, from scratch or from pre-trained representations (such as ResNet, BERT, etc.)
  Randomly sample 1/3 of $\mathcal{D}^{\text{train}}$ as Predictor's validation data $\tilde{\mathcal{D}}^{\text{val}}$. We denote the remaining 2/3 of $\mathcal{D}^{\text{train}}$ as $\tilde{\mathcal{D}}^{\text{train}}$.
  repeat
    Train *Predictor* to minimize the empirical risk on $\tilde{\mathcal{D}}^{\text{train}}$ for one epoch.
    Evaluate *Predictor* on $\tilde{\mathcal{D}}^{\text{val}}$.
  until validation performance on $\tilde{\mathcal{D}}^{\text{val}}$ hasn't improved in the past 5 iterations
  Evaluate *Predictor* on $\mathcal{D}^{\text{test}}$.
  Compute the `gap` between Predictor's performance on $\tilde{\mathcal{D}}^{\text{val}}$ and its performance on $\mathcal{D}^{\text{test}}$.

  # Step 3: train Splitter to identify more challenging splits.
  repeat
    # train Splitter for one epoch
    Set `step` = 0.
    repeat
      Randomly sample a batch of examples from $\mathcal{D}^{\text{total}}$ and compute the regularity constraints Ω_1, Ω_2 over this batch (Eq 1).
      Randomly sample another batch of examples from $\mathcal{D}^{\text{test}}$ and compute $\mathcal{L}^{\text{gap}}$ (Eq 2).
      Compute the overall loss $\mathcal{L}^{\text{total}} = \mathcal{L}^{\text{gap}} + \Omega_1 + \Omega_2$ (Eq 3).
      Backprop $\mathcal{L}^{\text{total}}$ and update Splitter's parameters.
      `step` += 1
    until `step` = 100
    Set $\bar{\mathcal{L}}^{\text{total}}$ to be the average loss over the current epoch.

  until $\bar{\mathcal{L}}^{\text{total}}$ fails to improve by at least 10^{-3} in the past 5 iterations

  `outer_iter` += 1

until `gap` stops increasing in the past 5 iterations