Adversarial NLI, Addressing Dataset Biases: CS 7643

Bhaskar Joshi, Guoyi Cao, Matt Raporte, Zhijiao Chen
Georgia Institute of Technology

Abstract

We used the Adversarial NLI dataset provided by Facebook, and curated a subset of this dataset by applying a bias reduction technique called AFLite which is a practical approximation of the original AFOpt algorithm. With the advent of competitive NLI models we also require a dataset that is able to satisfy its training needs. The newer models are usually more complex than the older ones hence have more trainable parameters. This means if the training dataset is biased or doesn't have diversity, the training process might suffer and this might aid in overestimating the model's ability and consequently not performing well on the out of sample dataset. The training dataset is as important as the model selection for producing a robust model that can generalize well to real world conditions. Using AFLite we were able to demonstrate the reduction in predictability/bias and improvement in the accuracy of the model on the filtered dataset with respect to models trained on the original dataset.

1. Introduction/Background/Motivation

As per the paper[3], current datasets used to train NLI (Natural Language Inference) models are biased towards "predictable" inference and hence the model is able to memorize the dataset and overfit. Due to this even though the model accuracy looks great, it performs poorly in the real world. Le Bras, Ronan, et al. [3] proposed the algorithm AFLite (Lightweight Adversarial Filtering), that helps reduce bias in a dataset. This algorithm iteratively removes instances that are accurately predicted by simple linear classifiers and only uses the difficult ones for training. This approach has shown success in NLI and vision experiments.

1.1. Data

We used ANLI datasets collected via 3-round HAMLET (Human-And-Model-in-the-Loop Enabled Training) by Nie, Yixin, et al [5][6]. During data collection annotators write hypotheses given context and desired target, then the data is split into train, dev and test sets. A language model is trained using the train set along with supplementary datasets and validated with the dev set. After wrong predictions are verified by human annotators, these wrong records are split between train, dev and test sets while all correctly predicted instances are added only to the train set.

The dataset is provided in JSONL format where each line is a record with JSON format. One record of the dataset contains a "uid" which is the record identifier, "context" holds the main premise, "hypothesis" is the statement which is to be tested against the context, and "label" which provides label whether the hypothesis is an entailment (e), neutral (n), or contradiction (c) with respect to the context. The field "reason" is provided by the annotators stating why the hypothesis is labeled as such. This field is not used in our project. An example of the data (in JSON format) looks like below:

"uid": "1b2ade0a-93bc-4e9a-82ee-b354c8044cf8",

"context":"The diocese of Vannida (in Latin: Dioecesis Vannidensis) is a suppressed and titular See of the Roman Catholic Church. It was centered on the ancient Roman Town of Vannida, in what is today Algeria, is an ancient episcopal seat of the Roman province of Mauritania Cesariense.",

"hypothesis": "The diocese of Vannida is located in Europe",

"label": "c",

"reason": "No it is in Algeria"

Relying on human intuition alone to verify the diversity of the dataset will eventually introduce bias in the dataset[2][1]. Using a programmatic approach will help us quantify what amount of bias is present in the dataset and what is the amount of data we want to remove from the training set. Providing a de-biased dataset for NLI training will help improve model performance and we can be more confident that our model will generalize to real world data.

2. Approach

The research papers we referred to heavily filtered the dataset. The amount of data that survived was around 1/3rd of the original size. Along with comparing multiple filtering approaches we also wanted to experiment if it's really necessary to filter a dataset as aggressively as the papers we referenced. Our hypothesis was that by removing so much data

we might negatively affect the NLI training process. Therefore, we applied two filtering approaches programmatically and prepared multiple datasets:

1. AFLite filtering (instances with high predictability)
2. Removing duplicates (instances with high cosine similarity scores)

We used the resulting datasets to train NLI models and compared the results. We were able to improve the test set accuracy compared against the baseline of the models provided in [6] by using the dataset that was finally selected.

2.1. AFLite Algorithm

AFLite is an algorithm proposed by Le Bras, Ronan, et al."Adversarial Filters of Dataset Biases."[3]. The objective of the algorithm is to prevent task overfitting to dataset biases. For NLI, the bias is hypothesis-only reliance in Natural Language Inference by Gururangan et al., 2018 [1]. This research on this bias indicates that the model potentially exploits solely on hypothesis instead the relationship between hypothesis and context to predict the label. AFLite is designed to be a bottom-up method to systematically reduce dataset biases.

AFLite takes in feature representation from a NLI model for each data entry in the dataset. For each iteration, the dataset will be partitioned into a training subset and a validation subset. A classifier will be trained using the training subset based on the feature representation and the corresponding label. Then the classifier is used to predict on the validation subset with the prediction correctness being recorded. After several iterations, the data entries with the highest predictability scores will be removed to conclude one round of filtering. The filtering will continue until it meets the stopping rule.

Due to the limited timeframe and resources, we cannot afford to train a model from scratch for feature representation of the dataset. A pre-trained BERT-Large model implemented for NLI from the paper Adapting by Pruning: A Case Study on BERT[4] is utilized to extract the feature representation for R1, R2 and R3 training datasets. The feature vector is in the size of 1,024 and is the last layer before the fully connected layer and softmax in the model architecture.

The classifier being used is a one-layer linear classifier connecting the feature vectors with the labels. A new classifier is constructed for each iteration under each round of filtering. Hyperparameter is the same as advised by the paper with batch size 92, 3 epochs and learning rate 1e-5. Fig 1 is the pseudo code for AFLite.

2.2. Deduplication Algorithm

Apart from bias reduction the problem statement provided by Facebook also mentioned that a deduplication solution might also be useful. This is a valid expectation since any data collection process might introduce duplicate

Algorithm 1 AFLITE

Input: dataset $D = (X, Y)$, pre-computed representation $\Phi(X)$, model family $\mathcal{M}$, target dataset size n, number of random partitions m, training set size $t < n$, slice size $k \leq n$, early-stopping threshold τ
Output: reduced dataset S
$S = D$
while $|S| > n$ **do**
  // Filtering phase
  forall $i \in S$ **do**
    Initialize multiset of out-of-sample predictions $E(i) = \emptyset$
  for *iteration* $j : 1..m$ **do**
    Randomly partition S into $(T_j, S \setminus T_j)$ s.t. $|S \setminus T_j| = t$
    Train a classifier $\mathcal{L} \in \mathcal{M}$ on $\{(\Phi(x), y) \mid (x, y) \in S \setminus T_j\}$ ($\mathcal{L}$ is typically a linear classifier)
    forall $i = (x, y) \in T_j$ **do**
      Add the prediction $\mathcal{L}(\Phi(x))$ to $E(i)$
  forall $i = (x, y) \in S$ **do**
    Compute the predictability score $\tilde{p}(i) = |\{\hat{y} \in E(i) \text{ s.t. } \hat{y} = y\}| / |E(i)|$
  Select up to k instances S' in S with the highest predictability scores subject to $\tilde{p}(i) \geq \tau$
  $S = S \setminus S'$
  if $|S'| < k$ **then**
    break
return S

Figure 1. AFLite Algorithm from Le Bras, Ronan, et al. Jul. 2020

records. Therefore, we wanted to verify if removing the duplicate records has any substantial effect on the final model performance compared against AFLite results. For deduping we used BERT-base sentence embeddings and Cosine similarity of the embeddings in conjunction to get the similarity score of two sentences. If the similarity was equal to or above 98% (this was chosen after manually reviewing the results and can be treated as another hyperparameter) we kept only one of the sentences. Fig 2 is the pseudo code for deduping.

3. Experiments and Results

3.1. Baseline Training

We initially wanted to use a state-of-art RoBERTA architecture to validate our approach, however because of computational constraints we had to use BERT-base (a smaller parameter BERT transformer model) which trained two orders of magnitude faster. We also decided to reduce the train weights parameter (which controls how heavily we upsample the training data) during early experiments to further reduce training time while still achieving high accuracy scores. The original ratio weights were 10, 20, 10 for R1, R2 and R3, we used 2, 4, 2 for initial training. The model is validated with dev data during training every 4,000 batches. After the model is trained, it is validated with the corresponding test data. The total training time for these models is 7 hours each using Google Colaboratory.

Algorithm 2: Deduplication

Input : List, L of hypotheses of a context
Output: Reduced dataset
Remove stop words from the list
Generate vector embedding for each sentence
Calculate cosine similarity matrix of the vectors
Calculate lower triangular matrix
for *i in the list* **do**
  compare cosine similarity factor of vector(i) with vector(i+1)
  if *cosine similarity factor >= 0.98* **then**
    Flag the vector(i) for deletion;
  else
    pass;
  end
  if *cosine similarity factor >= 0.98 and one of two vectors already marked for deletion* **then**
    Remove the deletion flag;
  else
    pass;
  end
  increment i by 1
end
Return the indexes marked for deletion
Remove the records flagged for deletion from the list
Return the reduced list, L

Figure 2. Deduplication Algorithm

3.2. Deduplication and Filtering

We then created 4 datasets with the two algorithms in section 2.2. With the deduplication algorithm, we compared the similarity score of two sentences and if the similarity was equal to or above 98% we kept only one of the sentences. This approach was applied only to the hypothesis and comparison was performed between hypotheses per each context so that we don't accidentally eliminate valid candidates across different contexts. With the similarity factor selected it removed roughly 4%-5% of the original data. The resulting accuracy with this dataset was slightly better than the baseline but didn't perform as well as the AFLite filtered dataset. The amount of improvement was encouraging and we wanted to prepare multiple datasets with deduping based on different levels of cosine similarity index, but limitation in training resources and time meant we would not be able to make use of these datasets. Given more time we would like to experiment with datasets deduped using multiple similarity factors and compare the result with AFLite filtered dataset.

For the AFLite algorithm, we design the experiment to adopt the stopping rule of keeping 1/3, 1/2and 2/3 of the original dataset. Other values for hyperparameters are inherited or calculated proportionally based on the values utilized in the paper. The partition is set to 10% for training and 90% for validation. It runs for 64 iterations for each round of filtering and the threshold is 75% accuracy for filtering. If the count of data entries that exceed the threshold is greater than 2.5% of the count of the dataset, we only eliminate the top 2.5% from the dataset, to avoid filtering out too many entries in one round. The algorithm will run until it meets the stopping rule. Figure 3 shows that the average predictability score decreases since we filtered out predictable records throughout the process.

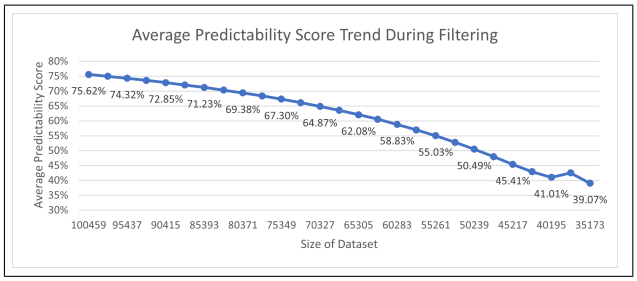

Figure 3. Average Predictability Score Trend During AFLite Filtering Process

Table 1 shows some examples of records that was filtered (dedup, 1/2 filter), and survived.

Context	Warwick Todd is a fictional character created by author and comedian Tom Gleisner. Todd is the author of four fictional cricket diaries:"The Warwick Todd Diaries" (1997), "" (1998), "Warwick Todd Goes the Tonk" (2001) and "Warwick Todd - Up in the Block Hole" (2009). The first 3 books in the series were Australian bestsellers.
AFLite removed	Warwick Todd did not play cricket
Survived	Tom Gleisner is Warwick Todd
Dedup removed	Warwick Todd is Tom Gleisner.
Survived	Tom Gleisner has met Warwick Todd face to face
AFLite removed	Warwick Todd is not a real person.
Survived	The Warwick Todd Diaries were released before the year 2000.
AFLite removed	Warwick Todd Goes the Tonk is Tom Gleisner's most popular novel.
AFLite removed	Tom Gleisner is working on a fifth Warwick Todd diary,
Survived	Tom Gleisner wrote the book, "Warwick Todd Goes the Tonk" in 2001.
Dedup removed	The Warwick Todd Diaries were released before the year 2009.

Table 1. Examples of records filtered and survived.

3.3. Filtered Dataset Training

Four BERT-base models are trained with the deduped and filtered datasets. We adjusted ratio weights based on the baseline training parameter to keep the size of training

sets comparable. i.e. For 1/3 filtered data, ratio weight is 3x baseline ratio weight; for 1/2 filtered data, ratio weight is 2x baseline ratio weight, etc. Validation steps are the same as 3.1.

3.4. Training the Final Model

We saw improvements in all of the 4 experiment datasets (details are in section 3.5), where 1/2 filtered data has the highest and most stable dev accuracy as shown in figure 4. Finally, we selected the 1/2 filtered data, increased ratio weight to default (10, 20, 10) and trained RoBERTa-Large so that we could compare our final model performance against the high-scores in the Adversarial NLI repository. With the dataset we prepared we were able to increase the accuracy across all data subsets when compared to the baseline provided by Facebook. The total training time for the final model is 21 hours using Google Colaboratory pro with better GPU and processor allocation. Details are present in the result section.

3.5. Results

Using the AFLite algorithm we prepared a more robust subset of the dataset by filtering out the predictable records and preserving a significant amount of diversity thereby making the dataset less biased towards predictable inference. Once this dataset was used to train the NLI model we were able to see improvement in accuracy compared to the baseline provided.

Figure 4. Dev accuracy from initial training (ratio weight 2, 4, 2 in baseline model)

In table 2 we can see the baseline accuracy value provided by Facebook[3] and the result we were able to receive trained on RoBERTa-large.

This result helps to bolster our initial hypothesis that we should not be removing two third data from the provided dataset. When we define the stopping criteria, we might be aggressive and remove some data points which were aiding in diversity of the dataset. Under the hood, removing these data points affected the training process therefore, the model used by Facebook might not have got as good results as ours. Of course, each dataset is different and the best approach is to scope the ML training in reduced form with different sets of filtered data and choose the dataset that best performs in this selection process.

Model (roberta-large)	**A1(dev)**	**A2(dev)**	**A3(dev)**
Huggingface	73.8	50.8	46.1
Group-16	74.5	53.8	48.2
Model (roberta-large)	**A1(test)**	**A2(test)**	**A3(test)**
Huggingface	73.6	49.3	45.5
Group-16	74.0	52.3	47.8

Table 2. Comparison of accuracy between baseline and group-16

4. Experience

The most pressing issue our project faced was related to hardware for training the model. The models we wanted to train were huge. BERT-base checkpoints were close to 1 GB and RoBERTa-Large was 4 GB in size and we faced some issues while storing the model checkpoints on google drive. The limitation of Google colab and the overhead of loading the checkpoints once the connection was severed, ate into the model selection and training time that we initially provisioned.

Therefore, we made a decision early on to train with smaller ratio weights and limit the training models to BERT-base and RoBERTa-Large so that we can compare different filtering methods before committing to a bigger model run. It also helped us work in parallel and everyone got a chance to work with the transformer model.

With this project we were able to closely monitor how the dataset used for training affects the final outcome of the training. With the bias and redundancy introduced in the data collection process it is imperative that we address these problems before feeding the data to the ML training model. We were also able to understand the inner workings of different NLI models and the amount of effort it needs to train the models. Given more time and resources we would like to further this experiment by extending the language models, and datasets prepared with different schemes.

5. Work Division

Student Name	Contributed Aspects	Details
Bhaskar Joshi	Deduplication, NLI model testing; NLI training; Report writing	Experimented and created the solution for deduplication of the dataset. Trained the NLI model for dedup dataset, 2/3rd dataset, and final RoBERTa model.
Guoyi Cao	AFLite Feature Representation; AFLite classifier; AFLite Testing; NLI training;	Run feature representation calculation through pre-trained BERT-Large. Implemented linear classifier and ran AFLite for filtering. Trained 1/3rd dataset model.
Matt Raporte	NLI Model preparation; NLI training	Tested training time of different NLI models. Picked hyperparameters for baseline and trained.
Zhijiao Chen	AFLite predictability score; AFLite testing; NLI training; Report writing	Implemented AFLite. Trained NLI model for 1/2 filtered dataset. Updated validation script for test data validation.

Appendices

A. Project code repository

https://github.com/matt-raporte/anli

B. Dataset repository

https://github.com/facebookresearch/anli

C. AFLite Training scheme

Filtered Data	n (Result Size)	t (Train Ratio)	m (Iteration)	k (Stop Rule)	T (Predictability Filter)
1/3	5649 (R1) 15153(R2) 33486(R3)	0.1	64	424 (R1) 1137(R2) 2511(R3)	0.75
1/2	8473(R1) 22730(R2) 50230(R3)	0.1	64	424 (R1) 1137(R2) 2511(R3)	0.75
2/3	11297(R1) 30307(R2) 66973(R3)	0.1	64	424 (R1) 1137(R2) 2511(R3)	0.75

Table 3. AFLite Training scheme

D. Models

NLI Training for dataset selection		**Final NLI Training**	
Model	BERT-Base	Model	RoBERTa-Large
adam_epsilon	1e-08	adam_epsilon	1e-08
epochs	2	epochs	2
eval_frequency	2000	eval_frequency	4000
gradient_accumulation_steps	4	gradient_accumulation_steps	4
learning_rate	1e-05	learning_rate	1e-05
max_grad_norm	1.0	max_grad_norm	1.0
max_length	156	max_length	156
per_gpu_eval_batch_size	16	per_gpu_eval_batch_size	16
per_gpu_train_batch_size	4	per_gpu_train_batch_size	8
single_gpu	True	single_gpu	True
weight_decay	0.0	weight_decay	0.0

Table 4. Models Hyper-parameter

E. Hardware

Environment	**Device**	**Memory**	**Computation Capability**
Google Colab standard	Tesla T4	12 GB	7.5
Google Colab pro	Tesla P100-PCIE-16GB	16 GB	6

Table 5. Hardware

F. BERT-Base accuracy of datasets

Experiment Name	Train Weight	Epoch	Valid Data	Dev Accuracy	Test Data	Test Accuracy
Baseline	2,4,2	2	anli_r1_dev	0.538	anli_r1_test	-
			anli_r2_dev	0.445	anli_r2_test	-
			anli_r3_dev	0.443	anli_r3_test	-
Filtered Half	4,8,4	2	anli_r1_dev	0.583	anli_r1_test	0.558
			anli_r2_dev	0.504	anli_r2_test	0.477
			anli_r3_dev	0.455	anli_r3_test	0.482
Filtered One-third	6,12,6	2	anli_r1_dev	0.602	anli_r1_test	0.578
			anli_r2_dev	0.494	anli_r2_test	0.486
			anli_r3_dev	0.437	anli_r3_test	0.479
Filtered Two-third	3,6,3	2	anli_r1_dev	0.563	anli_r1_test	-
			anli_r2_dev	0.482	anli_r2_test	-
			anli_r3_dev	0.455	anli_r3_test	-
Deduped	2,4,2	2	anli_r1_dev	0.53	anli_r1_test	-
			anli_r2_dev	0.458	anli_r2_test	-
			anli_r3_dev	0.449	anli_r3_test	-

Table 6. BERT-Base accuracy of datasets

References

[1] Gururangan et al. Annotation artifacts in natural language inference data", 2018. arXiv:1803.02324v2 [cs.CL]. 1, 2

[2] Geirhos et al. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness", 2018. arXiv:1811.12231 [cs.CL]. 1

[3] Ronan Le Bras et al. Adversarial filters of dataset biases.", 2020. arXiv:2002.04108v3 [cs.CL]. 1, 2

[4] Yang Gao et al. Adapting by pruning: A case study on bert", 2021. arXiv:2105.03343v1 [cs.LG]. 2

[5] Yixin Nie et al. Adversarial nli: A new benchmark for natural language understanding.", 2020. arXiv:1910.14599v2 [cs.CL]. 1

[6] Facebook. Anli github repository", 2021. The ANLI repository, includes code for training state-of-art models including RoBERTa, ALBert, BART, ELECTRA, XLNet. 1, 2