Analyzing and Mitigating Dataset Artifacts in NLI

Project: Final Project - CS388
Dataset: SNLI (Stanford Natural Language Inference)
Model: ELECTRA-small
Goal: Detect and mitigate dataset artifacts using hypothesis-only baselines and ensemble debiasing

Project Structure

- **Part 1: Analysis** - Detect artifacts and analyze model errors
- **Part 2: Fix** - Implement and evaluate debiasing method

Setup and Installation

```
# Connecting using personal token

import os
from google.colab import userdata

os.environ['gituser'] = userdata.get('gituser')
os.environ['gitpw'] = userdata.get('gitpw')
os.environ['REPO'] = 'fp-dataset-artifacts'

!git clone https://$gituser:$gitpw@github.com/$gituser/$REPO.git
```

```
fatal: destination path 'fp-dataset-artifacts' already exists and is not an empty directory.
```

```
%cd fp-dataset-artifacts/
```

```
/content/fp-dataset-artifacts
```

```
# Install required packages
%pip install -q -r requirements.txt
```

Part 1: Analysis

Part 1.1: Baseline Model Training

Train a standard NLI model on SNLI dataset using both premise and hypothesis.

```
!python train/run.py --do_train --do_eval --task nli --dataset snli --model google/electra-small-discriminator --output_dir ./ou
```

```
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1764985165.216146   28271 cuda_dnn.cc:8579] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN w
E0000 00:00:1764985165.222989   28271 cuda_blas.cc:1407] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLA
W0000 00:00:1764985165.240953   28271 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid link
W0000 00:00:1764985165.240981   28271 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid link
W0000 00:00:1764985165.240984   28271 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid link
W0000 00:00:1764985165.240987   28271 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid link
2025-12-06 01:39:25.246196: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU i
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler f
Using the `WANDB_DISABLED` environment variable is deprecated and will be removed in v5. Use the --report_to flag to control the integratio
Some weights of ElectraForSequenceClassification were not initialized from the model checkpoint at google/electra-small-discriminator and a
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Preprocessing data... (this takes a little bit, should only happen once per dataset)
/content/fp-dataset-artifacts/train/run.py:189: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.
  trainer = trainer_class(
{'loss': 0.9469, 'grad_norm': 19.131834030151367, 'learning_rate': 1.9467733333333333e-05, 'epoch': 0.08}
{'loss': 0.7125, 'grad_norm': 6.712376594543457, 'learning_rate': 1.8934400000000003e-05, 'epoch': 0.16}
{'loss': 0.6375, 'grad_norm': 10.302647590637207, 'learning_rate': 1.8401066666666666e-05, 'epoch': 0.24}
{'loss': 0.6004, 'grad_norm': 5.404224872589111, 'learning_rate': 1.7867733333333335e-05, 'epoch': 0.32}
{'loss': 0.5675, 'grad_norm': 18.229394912719727, 'learning_rate': 1.73344e-05, 'epoch': 0.4}
{'loss': 0.5613, 'grad_norm': 18.186208724975586, 'learning_rate': 1.6801066666666667e-05, 'epoch': 0.48}
{'loss': 0.5453, 'grad_norm': 23.964557647705078, 'learning_rate': 1.6267733333333337e-05, 'epoch': 0.56}
```

```
{'loss': 0.5165, 'grad_norm': 6.462014198303223, 'learning_rate': 1.4667733333333334e-05, 'epoch': 0.8}
{'loss': 0.5065, 'grad_norm': 7.721761226654053, 'learning_rate': 1.4134400000000001e-05, 'epoch': 0.88}
{'loss': 0.508, 'grad_norm': 13.849309921264648, 'learning_rate': 1.3601066666666668e-05, 'epoch': 0.96}
{'loss': 0.477, 'grad_norm': 6.8057379722595215, 'learning_rate': 1.3067733333333334e-05, 'epoch': 1.04}
{'loss': 0.4732, 'grad_norm': 14.111554145812988, 'learning_rate': 1.2534400000000001e-05, 'epoch': 1.12}
{'loss': 0.4544, 'grad_norm': 12.841914176940918, 'learning_rate': 1.2001066666666668e-05, 'epoch': 1.2}
{'loss': 0.4496, 'grad_norm': 11.674468040466309, 'learning_rate': 1.1467733333333335e-05, 'epoch': 1.28}
{'loss': 0.4543, 'grad_norm': 14.737910270690918, 'learning_rate': 1.09344e-05, 'epoch': 1.36}
{'loss': 0.4554, 'grad_norm': 9.141331672668457, 'learning_rate': 1.0401066666666668e-05, 'epoch': 1.44}
{'loss': 0.4361, 'grad_norm': 9.355502128601074, 'learning_rate': 9.867733333333334e-06, 'epoch': 1.52}
{'loss': 0.4425, 'grad_norm': 8.792309761047363, 'learning_rate': 9.334400000000002e-06, 'epoch': 1.6}
{'loss': 0.4573, 'grad_norm': 8.223569869995117, 'learning_rate': 8.801066666666668e-06, 'epoch': 1.68}
{'loss': 0.4303, 'grad_norm': 10.115617752075195, 'learning_rate': 8.267733333333334e-06, 'epoch': 1.76}
{'loss': 0.4388, 'grad_norm': 11.57454776763916, 'learning_rate': 7.7344e-06, 'epoch': 1.84}
{'loss': 0.4394, 'grad_norm': 7.7778778076171875, 'learning_rate': 7.201066666666667e-06, 'epoch': 1.92}
{'loss': 0.4284, 'grad_norm': 16.731571197509766, 'learning_rate': 6.667733333333334e-06, 'epoch': 2.0}
{'loss': 0.383, 'grad_norm': 13.084661483764648, 'learning_rate': 6.1344e-06, 'epoch': 2.08}
{'loss': 0.3919, 'grad_norm': 26.146312713623047, 'learning_rate': 5.601066666666667e-06, 'epoch': 2.16}
{'loss': 0.3999, 'grad_norm': 8.977935791015625, 'learning_rate': 5.067733333333333e-06, 'epoch': 2.24}
{'loss': 0.4115, 'grad_norm': 15.60965347290039, 'learning_rate': 4.5344e-06, 'epoch': 2.32}
{'loss': 0.3956, 'grad_norm': 7.38207483291626, 'learning_rate': 4.001066666666667e-06, 'epoch': 2.4}
{'loss': 0.3835, 'grad_norm': 14.191218376159668, 'learning_rate': 3.4677333333333337e-06, 'epoch': 2.48}
{'loss': 0.3978, 'grad_norm': 8.772391319274902, 'learning_rate': 2.9344000000000002e-06, 'epoch': 2.56}
{'loss': 0.3952, 'grad_norm': 12.477909088134766, 'learning_rate': 2.4010666666666668e-06, 'epoch': 2.64}
{'loss': 0.3924, 'grad_norm': 7.550316333770752, 'learning_rate': 1.8677333333333335e-06, 'epoch': 2.72}
{'loss': 0.3944, 'grad_norm': 7.706876277923584, 'learning_rate': 1.3344e-06, 'epoch': 2.8}
{'loss': 0.396, 'grad_norm': 12.262116432189941, 'learning_rate': 8.010666666666667e-07, 'epoch': 2.88}
{'loss': 0.3793, 'grad_norm': 9.734000205993652, 'learning_rate': 2.6773333333333335e-07, 'epoch': 2.96}
{'train_runtime': 905.2011, 'train_samples_per_second': 331.418, 'train_steps_per_second': 20.714, 'train_loss': 0.47823227986653644, 'epoc
100% 18750/18750 [15:05<00:00, 20.71it/s]
100% 616/616 [00:10<00:00, 57.57it/s]
Evaluation results:
{'eval_loss': 0.40020179748535156, 'eval_accuracy': 0.8577525019645691, 'eval_runtime': 10.727, 'eval_samples_per_second': 917.501, 'eval_s
```

```
# Check baseline results
import json
with open(os.path.join('outputs', 'evaluations', 'baseline_100k', 'eval_metrics.json'), 'r') as f:
    baseline_metrics = json.load(f)

print("=" * 80)
print("Baseline Model Results")
print("=" * 80)
print(f"Accuracy: {baseline_metrics['eval_accuracy']:.4f} ({baseline_metrics['eval_accuracy']*100:.2f}%)")
print(f"Eval Loss: {baseline_metrics.get('eval_loss', 'N/A')}")
```

```
================================================================================
Baseline Model Results
================================================================================
Accuracy: 0.8578 (85.78%)
Eval Loss: 0.40020179748535156
```

Part 1.2: Artifact Detection - Hypothesis-Only Model

Train a model that only sees the hypothesis (not the premise) to detect dataset artifacts.
If this model achieves >33.33% accuracy (random baseline), it indicates strong artifacts exist.

```
!python train/train_hypothesis_only.py
```

```
 92% 567/616 [00:09<00:00, 59.71it/s]
 93% 574/616 [00:09<00:00, 60.06it/s]
 94% 581/616 [00:09<00:00, 60.14it/s]
 95% 588/616 [00:09<00:00, 59.71it/s]
100% 18750/18750 [14:44<00:00, 21.75it/s]
 97% 600/616 [00:10<00:00, 59.43it/s]
 98% 606/616 [00:10<00:00, 59.53it/s]

{'eval_loss': 0.7663306593894958, 'eval_accuracy': 0.6719163060188293, 'eval_runtime': 10.3934, 'eval_samples_per_second': 946.947, 'eval_s
100% 18750/18750 [14:44<00:00, 21.75it/s]
100% 616/616 [00:10<00:00, 59.83it/s]
{'train_runtime': 956.4607, 'train_samples_per_second': 313.656, 'train_steps_per_second': 19.604, 'train_loss': 0.7894001281738281, 'epoch
100% 18750/18750 [14:45<00:00, 21.18it/s]

================================================================================
EVALUATING HYPOTHESIS-ONLY MODEL...
================================================================================
100% 616/616 [00:10<00:00, 59.50it/s]
100% 616/616 [00:10<00:00, 58.75it/s]

================================================================================
HYPOTHESIS-ONLY MODEL RESULTS
================================================================================
Accuracy: 0.6719 (67.19%)
Loss:     0.7663
================================================================================

Model saved to: ../outputs/evaluations/hypothesis_only_model/
Metrics saved to: ../outputs/evaluations/hypothesis_only_model/eval_metrics.json

Saving predictions...
Predictions saved to: ../outputs/evaluations/hypothesis_only_model/eval_predictions.jsonl

✓ Training complete!
wandb:
wandb: You can sync this run to the cloud by running:
wandb: wandb sync /content/fp-dataset-artifacts/wandb/offline-run-20251206_015606-n3wr77jh
wandb: Find logs at: wandb/offline-run-20251206_015606-n3wr77jh/logs
```

```
# Check hypothesis-only results
with open(os.path.join('outputs', 'evaluations', 'hypothesis_only_model', 'eval_metrics.json'), 'r') as f:
        hyp_metrics = json.load(f)

hyp_accuracy = hyp_metrics['eval_accuracy']
random_baseline = 1.0 / 3.0
above_random = hyp_accuracy - random_baseline

print("=" * 80)
print("Hypothesis-Only Model Results (Artifact Detection)")
print("=" * 80)
print(f"Accuracy: {hyp_accuracy:.4f} ({hyp_accuracy*100:.2f}%)")
print(f"Random Baseline: {random_baseline:.4f} ({random_baseline*100:.2f}%)")
print(f"Above Random: {above_random:.4f} ({above_random*100:.2f}%)")
print(f"\n{'STRONG ARTIFACTS DETECTED!' if above_random > 0.2 else 'Weak artifacts detected' if above_random > 0.1 else 'No
```

```
================================================================================
Hypothesis-Only Model Results (Artifact Detection)
================================================================================
Accuracy: 0.6080 (60.80%)
Random Baseline: 0.3333 (33.33%)
Above Random: 0.2747 (27.47%)

STRONG ARTIFACTS DETECTED!
```

Part 1.3: Baseline Error Analysis

Analyze the baseline model's errors, confusion patterns, and identify artifact-related mistakes.

```
!python analyze/error_analysis.py
```

```
Error 3:
  Premise: At an outdoor event in an Asian-themed area, a crowd congregates as one person in a yellow Chinese dragon costume confronts the
  Hypothesis: A crowd is dancing
  True: Neutral, Predicted: Contradiction

2. Examples where TRUE=Contradiction but PREDICTED=Neutral:
Error 1:
  Premise: Two young children in blue jerseys, one with the number 9 and one with the number 2 are standing on wooden steps in a bathroom a
  Hypothesis: Two kids in jackets walk to school.
  True: Contradiction, Predicted: Neutral

Error 2:
  Premise: A taxi SUV drives past an urban construction site, as a man walks down the street in the other direction.
  Hypothesis: A man is chasing an SUV that is going in the same direction as him.
  True: Contradiction, Predicted: Neutral

Error 3:
  Premise: A person rides his bicycle in the sand beside the ocean.
  Hypothesis: A person is falling off his bike.
  True: Contradiction, Predicted: Neutral

3. Examples where TRUE=Entailment but PREDICTED=Neutral:
Error 1:
  Premise: A man in a black shirt is playing a guitar.
  Hypothesis: He is playing a song.
  True: Entailment, Predicted: Neutral

Error 2:
  Premise: Girl plays with colorful letters on the floor.
  Hypothesis: The girl is having fun learning her letters.
  True: Entailment, Predicted: Neutral

Error 3:
  Premise: A man wandering in the desert as the clouds roll in.
  Hypothesis: A man wonders in the desert.
  True: Entailment, Predicted: Neutral

================================================================================

=== SUMMARY ===
Total errors: 1400
Most common error types:
  True=Neutral       -> Predicted=Contradiction:  340 (24.3%)
  True=Contradiction -> Predicted=Neutral      :  306 (21.9%)
  True=Neutral       -> Predicted=Entailment   :  273 (19.5%)
  True=Entailment    -> Predicted=Neutral      :  232 (16.6%)
  True=Contradiction -> Predicted=Entailment   :  131 (9.4%)

================================================================================
```

Part 1.4: Visualizations - Baseline Model

Create visualizations to show error patterns and confusion matrices.

```
!python  analyze/visualize_baseline.py
```

```
Loading baseline predictions...
Creating confusion matrix...
Confusion matrix saved to: /content/fp-dataset-artifacts/outputs/evaluations/baseline_confusion_matrix.png
Creating per-class accuracy chart...
Per-class accuracy chart saved to: /content/fp-dataset-artifacts/outputs/evaluations/baseline_per_class_accuracy.png
Baseline visualizations completed!
```

Part 2: Fix - Debiasing Implementation

Part 2.1: Train Debiased Model

Train a debiased model using confidence-based reweighting.
Examples where the hypothesis-only model is confident (likely artifacts) are downweighted.

```
!python  train/train_debiased.py
```

```
 86% 527/616 [00:14<00:02, 35.31it/s]
 86% 531/616 [00:14<00:02, 35.46it/s]
 87% 535/616 [00:15<00:02, 35.68it/s]
 88% 539/616 [00:15<00:02, 35.45it/s]
 88% 543/616 [00:15<00:02, 35.14it/s]
 89% 547/616 [00:15<00:01, 35.43it/s]
 89% 551/616 [00:15<00:01, 35.02it/s]
 90% 555/616 [00:15<00:01, 35.28it/s]
 91% 559/616 [00:15<00:01, 35.46it/s]
 91% 563/616 [00:15<00:01, 34.81it/s]
 92% 567/616 [00:15<00:01, 34.37it/s]
 93% 571/616 [00:16<00:01, 34.70it/s]
 93% 575/616 [00:16<00:01, 35.03it/s]
 94% 579/616 [00:16<00:01, 34.40it/s]
 95% 583/616 [00:16<00:00, 34.70it/s]
 95% 587/616 [00:16<00:00, 34.46it/s]
 96% 591/616 [00:16<00:00, 34.54it/s]
 97% 595/616 [00:16<00:00, 34.62it/s]
 97% 599/616 [00:16<00:00, 34.49it/s]
 98% 603/616 [00:17<00:00, 34.70it/s]
 99% 607/616 [00:17<00:00, 34.79it/s]
 99% 611/616 [00:17<00:00, 34.26it/s]

{'eval_loss': 0.25410524010658264, 'eval_accuracy': 0.8569396734237671, 'eval_runtime': 17.4354, 'eval_samples_per_second': 564.484, 'eval_
100% 18750/18750 [18:35<00:00, 17.96it/s]
100% 616/616 [00:17<00:00, 34.84it/s]
{'train_runtime': 1345.485, 'train_samples_per_second': 222.968, 'train_steps_per_second': 13.935, 'train_loss': 0.30209944132486977, 'epoc
100% 18750/18750 [18:35<00:00, 16.81it/s]

================================================================================
EVALUATING DEBIASED MODEL...
================================================================================
100% 616/616 [00:17<00:00, 35.83it/s]

================================================================================
DEBIASED MODEL RESULTS
================================================================================
Accuracy: 0.8569 (85.69%)
Loss:     0.2541
================================================================================

Debiased model saved to: ../outputs/evaluations/debiased_model/
Metrics saved to: ../outputs/evaluations/debiased_model/eval_metrics.json

Saving predictions for comparison...
100% 616/616 [00:17<00:00, 36.19it/s]
Predictions saved to: ../outputs/evaluations/debiased_model//eval_predictions.jsonl
Total predictions: 9842

✓ Training complete!
wandb:
wandb: You can sync this run to the cloud by running:
wandb: wandb sync /content/fp-dataset-artifacts/wandb/offline-run-20251206_021531-ue78kzvq
wandb: Find logs at: wandb/offline-run-20251206_021531-ue78kzvq/logs
```

```
# Check debiased results
import json
with open(os.path.join('outputs', 'evaluations', 'debiased_model', 'eval_metrics.json'), 'r') as f:
        debiased_metrics = json.load(f)

print("=" * 80)
print("Debiased Model Results")
print("=" * 80)
print(f"Accuracy: {debiased_metrics['eval_accuracy']:.4f} ({debiased_metrics['eval_accuracy']*100:.2f}%)")
print(f"Eval Loss: {debiased_metrics.get('eval_loss', 'N/A')}")
```

```
================================================================================
Debiased Model Results
================================================================================
Accuracy: 0.8642 (86.42%)
Eval Loss: 0.24399055540561676
```

```
!python analyze/compare_results.py
```

```
================================================================================
Results Comparison - Baseline vs Debiased
================================================================================

Random Baseline:        0.3333 (33.33%)
```

```
Hypothesis-Only:        0.6080 (60.80%) [Above random: +27.47%]
Baseline (Full Model):  0.8578 (85.78%)
Debiased:               0.8642 (86.42%) [Change: +0.64%]

================================================================================
Key Findings:
================================================================================
1. Hypothesis-Only model achieves 60.80%, proving strong artifacts exist!
2. Debiasing maintains performance: 86.42% vs 85.78%
3. Debiasing preserved performance

================================================================================
Per-Class Accuracy Comparison
================================================================================
Entailment     : Baseline=89.49%, Debiased=89.31%, Change=-0.18%
Neutral        : Baseline=81.05%, Debiased=82.38%, Change=+1.33%
Contradiction  : Baseline=86.67%, Debiased=87.46%, Change=+0.79%

================================================================================
Prediction Changes
================================================================================
Total predictions changed: 647 (6.6%)
Baseline wrong -> Debiased correct (FIXES): 327
Baseline correct -> Debiased wrong (BREAKS): 264
Net improvement: +63

Top 10 fixes saved to: /content/fp-dataset-artifacts/outputs/evaluations/fixes_examples.json
```

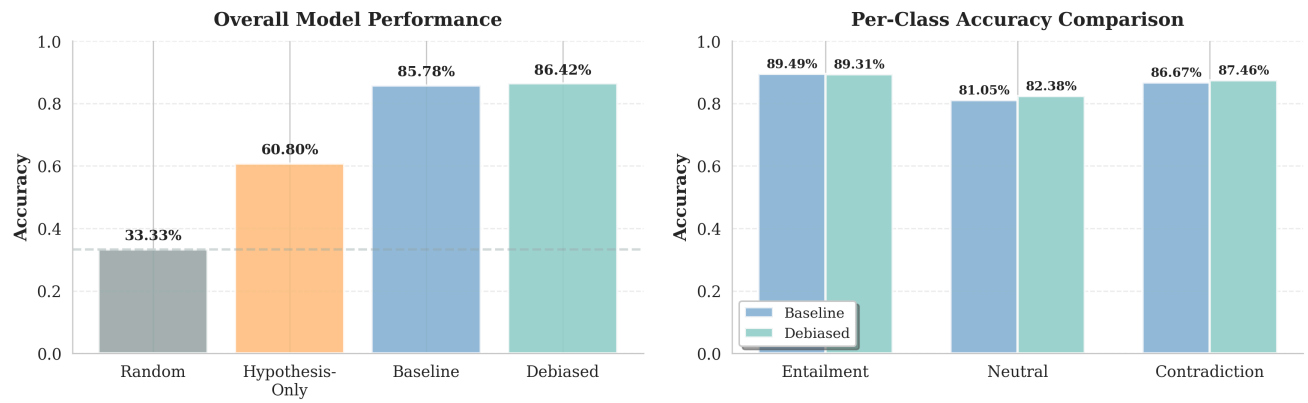
Part 2.3: Visualizations - Comparison

Create visualizations comparing baseline and debiased models.

```
!python  analyze/visualize_comparison.py
```

```
Loading metrics...
Loading predictions...
Creating comparison charts...
Comparison chart saved to: /content/fp-dataset-artifacts/outputs/evaluations/baseline_vs_debiased_comparison.png
Comparison visualizations completed!
```

```
from  IPython.display  import  Image
display(Image('outputs/evaluations/baseline_vs_debiased_comparison.png'))
```




```
!python  analyze/show_fixes.py
```

```
================================================================================
Examples Where Debiasing Fixed Baseline Errors
================================================================================

Fix Example 1:
  Premise: A man and a woman are looking at produce on display.
  Hypothesis: Two men throw produce at each other for fun.
  True Label: Contradiction
  Baseline Predicted: Neutral [WRONG]
```

```
  Debiased Predicted: Contradiction [CORRECT]
--------------------------------------------------------------------------------

Fix Example 2:
  Premise: People are throwing tomatoes at each other.
  Hypothesis: The men are covered in tomatoes.
  True Label: Neutral
  Baseline Predicted: Entailment [WRONG]
  Debiased Predicted: Neutral [CORRECT]
--------------------------------------------------------------------------------

Fix Example 3:
  Premise: A small ice cream stand with two people standing near it.
  Hypothesis: Two people in line to buy icecream.
  True Label: Neutral
  Baseline Predicted: Contradiction [WRONG]
  Debiased Predicted: Neutral [CORRECT]
--------------------------------------------------------------------------------

Fix Example 4:
  Premise: Number 916 is hoping that he is going to win the race.
  Hypothesis: A person is betting that he will win  the race.
  True Label: Neutral
  Baseline Predicted: Entailment [WRONG]
  Debiased Predicted: Neutral [CORRECT]
--------------------------------------------------------------------------------

Fix Example 5:
  Premise: An older gentleman enjoys a scenic stroll through the countryside.
  Hypothesis: An old man searches for a good place to die.
  True Label: Neutral
  Baseline Predicted: Contradiction [WRONG]
  Debiased Predicted: Neutral [CORRECT]
--------------------------------------------------------------------------------
```

Part 2.4: Negation Word Analysis and Visualization

Analyze the correlation between negation words and model predictions.
This helps identify if models learn spurious correlations (e.g., negation → contradiction).

```
# Comprehensive Negation Word Analysis with Visualizations
import json
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import os

# Load predictions
print("Loading predictions...")
baseline_predictions = []
with open(os.path.join('outputs', 'evaluations', 'baseline_100k', 'eval_predictions.jsonl'), 'r', encoding='utf-8') as f:
    for line in f:
        baseline_predictions.append(json.loads(line))

debiased_predictions = []
with open(os.path.join('outputs', 'evaluations', 'debiased_model', 'eval_predictions.jsonl'), 'r', encoding='utf-8') as f:
    for line in f:
        debiased_predictions.append(json.loads(line))

label_names = {0: "Entailment", 1: "Neutral", 2: "Contradiction"}

# Define negation words
negation_words = ['no', 'not', 'never', 'nobody', 'nothing', 'nowhere', 'neither', 'none', "n't", 'nor']

def has_negation(text):
    """Check if text contains negation words."""
    text_lower = text.lower()
    return any(neg in text_lower for neg in negation_words)

# Analyze negation for baseline
baseline_with_neg = [p for p in baseline_predictions if has_negation(p['hypothesis'])]
baseline_without_neg = [p for p in baseline_predictions if not has_negation(p['hypothesis'])]

# Analyze negation for debiased
debiased_with_neg = [p for p in debiased_predictions if has_negation(p['hypothesis'])]
debiased_without_neg = [p for p in debiased_predictions if not has_negation(p['hypothesis'])]
```

```
print("=" * 80)
print("NEGATION WORD ANALYSIS")
print("=" * 80)
print(f"\nTotal examples: {len(baseline_predictions)}")
print(f"Examples with negation: {len(baseline_with_neg)} ({len(baseline_with_neg)/len(baseline_predictions):.1%})")
print(f"Examples without negation: {len(baseline_without_neg)} ({len(baseline_without_neg)/len(baseline_predictions):.1%})")

# Calculate accuracy on negation examples
baseline_neg_acc = sum(1 for p in baseline_with_neg if p['label'] == p['predicted_label']) / len(baseline_with_neg)
debiased_neg_acc = sum(1 for p in debiased_with_neg if p['label'] == p['predicted_label']) / len(debiased_with_neg)

baseline_no_neg_acc = sum(1 for p in baseline_without_neg if p['label'] == p['predicted_label']) / len(baseline_without_neg)
debiased_no_neg_acc = sum(1 for p in debiased_without_neg if p['label'] == p['predicted_label']) / len(debiased_without_neg)

print(f"\n{'='*80}")
print("ACCURACY ON NEGATION EXAMPLES")
print("=" * 80)
print(f"Baseline - With negation: {baseline_neg_acc:.2%}")
print(f"Baseline - Without negation: {baseline_no_neg_acc:.2%}")
print(f"Debiased - With negation: {debiased_neg_acc:.2%}")
print(f"Debiased - Without negation: {debiased_no_neg_acc:.2%}")
print(f"\nChange on negation examples: {(debiased_neg_acc - baseline_neg_acc)*100:+.2f}%")

# Label distribution for negation examples
print(f"\n{'='*80}")
print("TRUE LABEL DISTRIBUTION (Hypotheses WITH Negation)")
print("=" * 80)
neg_true_labels = Counter(p['label'] for p in baseline_with_neg)
for label in [0, 1, 2]:
    count = neg_true_labels[label]
    pct = count / len(baseline_with_neg)
    print(f"{label_names[label]:15}: {count:4} ({pct:.1%})")

print(f"\n{'='*80}")
print("PREDICTED LABEL DISTRIBUTION (Hypotheses WITH Negation)")
print("=" * 80)
print("Baseline:")
baseline_neg_preds = Counter(p['predicted_label'] for p in baseline_with_neg)
for label in [0, 1, 2]:
    count = baseline_neg_preds[label]
    pct = count / len(baseline_with_neg)
    print(f"  {label_names[label]:15}: {count:4} ({pct:.1%})")

print("\nDebiased:")
debiased_neg_preds = Counter(p['predicted_label'] for p in debiased_with_neg)
for label in [0, 1, 2]:
    count = debiased_neg_preds[label]
    pct = count / len(debiased_with_neg)
    print(f"  {label_names[label]:15}: {count:4} ({pct:.1%})")

# Check if model over-predicts contradiction for negation
true_contrad_pct = neg_true_labels[2] / len(baseline_with_neg)
baseline_pred_contrad_pct = baseline_neg_preds[2] / len(baseline_with_neg)
debiased_pred_contrad_pct = debiased_neg_preds[2] / len(debiased_with_neg)

print(f"\n{'='*80}")
print("NEGATION → CONTRADICTION CORRELATION")
print("=" * 80)
print(f"True Contradiction rate (with negation): {true_contrad_pct:.1%}")
print(f"Baseline predicted Contradiction rate: {baseline_pred_contrad_pct:.1%}")
print(f"Debiased predicted Contradiction rate: {debiased_pred_contrad_pct:.1%}")
print(f"\nBaseline over-predicts Contradiction by: {(baseline_pred_contrad_pct - true_contrad_pct)*100:+.1f}%")
print(f"Debiased over-predicts Contradiction by: {(debiased_pred_contrad_pct - true_contrad_pct)*100:+.1f}%")
```

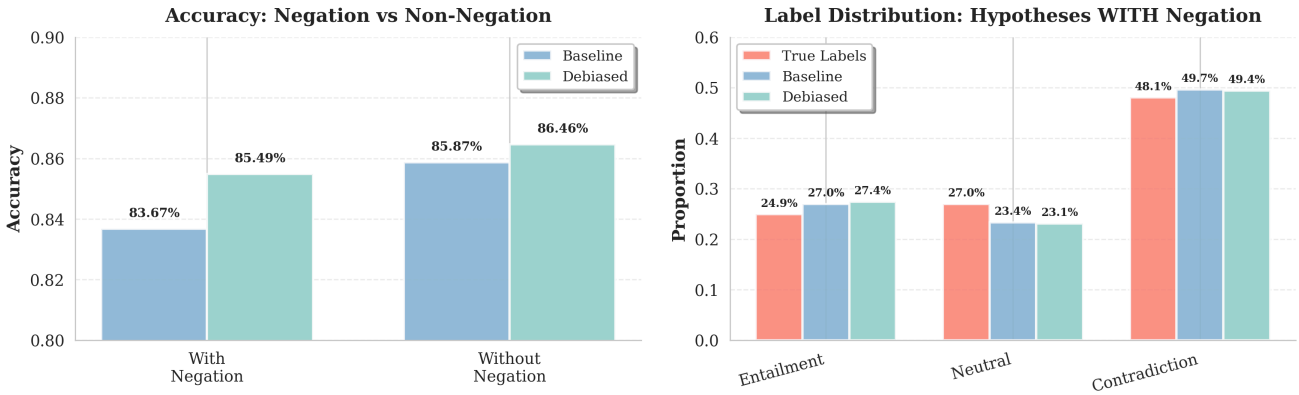
```
Loading predictions...
================================================================================
NEGATION WORD ANALYSIS
================================================================================

Total examples: 9842
Examples with negation: 441 (4.5%)
Examples without negation: 9401 (95.5%)

================================================================================
ACCURACY ON NEGATION EXAMPLES
================================================================================
Baseline - With negation: 83.67%
```

```
Baseline - Without negation: 85.87%
Debiased - With negation: 85.49%
Debiased - Without negation: 86.46%

Change on negation examples: +1.81%

================================================================================
TRUE LABEL DISTRIBUTION (Hypotheses WITH Negation)
================================================================================
Entailment     :  110 (24.9%)
Neutral        :  119 (27.0%)
Contradiction  :  212 (48.1%)

================================================================================
PREDICTED LABEL DISTRIBUTION (Hypotheses WITH Negation)
================================================================================
Baseline:
  Entailment     :  119 (27.0%)
  Neutral        :  103 (23.4%)
  Contradiction  :  219 (49.7%)

Debiased:
  Entailment     :  121 (27.4%)
  Neutral        :  102 (23.1%)
  Contradiction  :  218 (49.4%)

================================================================================
NEGATION → CONTRADICTION CORRELATION
================================================================================
True Contradiction rate (with negation): 48.1%
Baseline predicted Contradiction rate: 49.7%
Debiased predicted Contradiction rate: 49.4%

Baseline over-predicts Contradiction by: +1.6%
Debiased over-predicts Contradiction by: +1.4%
```

```
!python  analyze/visualize_negation.py
```

```
Loading predictions...

Creating negation analysis visualizations...
✓ Negation analysis chart saved to: /content/fp-dataset-artifacts/outputs/evaluations/negation_analysis.png
Negation analysis visualizations completed!
```

```
from  IPython.display  import  Image
display(Image('outputs/evaluations/negation_analysis.png'))
```



Update

```
!git  config  --global  user.name  "DinaberryPi"
!git  config  --global  user.email  "dinahenrykyy@gmail.com"
!git  status
```

```
On branch main
nothing to commit, working tree clean
```

```
!git add .
```

```
!git status
```

```
On branch main
nothing to commit, working tree clean
```

```
!git commit -m "update"
```

```
On branch main
nothing to commit, working tree clean
```

Gemini

```
!git push origin main --force
```

```
Enumerating objects: 473, done.
Counting objects: 100% (473/473), done.
Delta compression using up to 12 threads
```