COMPSCI 4NL3 Group 6 Project Step 3 Report

Ahsan Muzammil muzammia
Muhammad Jawad jawadm1
Sufyan Motala motalas

March 2025

Annotation Analysis

Agreement Calculations:

After collecting the data annotated by our peers, we combined all the annotated files into a single dataset. However, before merging, we assessed the agreement metrics on the duplicated data, which constituted approximately 15% of the entire dataset. This evaluation was necessary to determine the consistency of the annotations. To measure agreement, we employed Cohen's Kappa coefficient, as suggested in the assignment. This metric is well-suited for evaluating pairwise agreement between two annotators. Although we had four annotators, the dataset was split as follows:

- Annotator 1 labeled `data1.csv`
- Annotator 2 labeled `data2.csv`
- Annotator 3 labeled `data3.csv`
- Annotator 4 labeled `data4.csv`

The duplicated data was distributed across these files as follows:

- `data1.csv` and `data3.csv` contained approximately 300 duplicate instances.
- `data2.csv` and `data4.csv` contained approximately 100 duplicate instances.

For analysis, we grouped the duplicates as follows:

- **Bag A**: `data1.csv` + `data2.csv`
- **Bag B**: `data3.csv` + `data4.csv`

This grouping allowed for a more meaningful Cohen's Kappa evaluation.

Computing Cohen's Kappa Score

Using the `sklearn.metrics` library, we imported the `cohen_kappa_score` function to compute the agreement scores between annotators.

Overall Agreement Score

We first computed Cohen's Kappa between **Bag A** and **Bag B**, obtaining:

$$\text{Cohen's Kappa for Bag A and Bag B} = 0.3251 \tag{1}$$

This result indicates some level of consistency in labeling between annotators 1 and 2 compared to annotators 3 and 4.

Individual File Comparisons

To further investigate, we computed the Cohen's Kappa score separately for:

- `data1.csv` vs. `data3.csv`
- `data2.csv` vs. `data4.csv`

The results were:

$$\text{Cohen's Kappa for data1.csv and data3.csv} = 0.3213 \tag{2}$$
$$\text{Cohen's Kappa for data2.csv and data4.csv} = 0.1103 \tag{3}$$

Analysis of Results

The significantly lower Kappa score (0.1103) for `data2.csv` and `data4.csv` suggests greater inconsistencies in labeling between annotators 2 and 4. Possible explanations include:

- Differences in annotation guidelines or interpretation.
- The smaller size of `data2.csv` and `data4.csv` (100 duplicate instances) compared to `data1.csv` and `data3.csv` (300 duplicate instances), which may impact the statistical reliability of the score.

Ground Truth Labels

After evaluating annotation consistency using agreement metrics, the next step is to establish the ground truth labels for the dataset. The labeling approach we used varied based on whether the data we are dealing with is duplicated or unique.

For datasets `data5.csv` through `data8.csv`, the annotations provided by the respective annotators are directly assigned as the ground truth labels. However, for the duplicated data found in `data1.csv` through `data4.csv`, a systematic approach is required to determine the ground truth labels.

Majority Vote Approach

To resolve inconsistencies in the duplicated dataset, we employed a **majority voting** strategy. This method ensures that if a common label is assigned to a particular data point by multiple annotators, it is selected as the ground truth. In cases where disagreement arises, a manual decision process is implemented to assign the final label.

Manual Conflict Resolution

To handle conflicting labels, we developed a Python script that automates the label selection process:

- The script iterates through each row of `data1.csv` and `data3.csv`, checking for agreement in labels.
- If the labels match, the agreed-upon label is assigned as the ground truth.
- If a disagreement is detected, the script prompts the user to manually select the correct label while displaying the corresponding text.
- A similar procedure is applied to `data2.csv` and `data4.csv`.

Python Implementation

The following Python program was used to automate this process:

```
import pandas as pd

def perform_truth_table_analysis():
    file1 = "annotator3.csv"
    file2 = "annotator4.csv"
    output_file = "merged2.csv"

    df1 = pd.read_csv(file1)
    df2 = pd.read_csv(file2)

    assert set(df1.columns) == set(df2.columns)

    merged_data = []

    for (_, row1), (_, row2) in zip(df1.iterrows(), df2.iterrows()):
        if row1['id'] != row2['id']:
            print(f"ID mismatch: {row1['id']} != {row2['id']}")
            continue

        if row1['label'] == row2['label']:
            merged_data.append(row1.tolist())
        else:
            print("\nConflict detected:")
            print(f"ID: {row1['id']}")
            print(f"Text: {row1['body']}")
            print(f"Label in File 1: {row1['label']}")
            print(f"Label in File 2: {row2['label']}")

            while True:
                try:
                    new_label = int(input("Enter the correct label (-1, 0, or 1): "))
                    if new_label in [-1, 0, 1]:
                        break
                    else:
                        print("Invalid input. Please enter -1, 0, or 1.")
                except ValueError:
                    print("Invalid input. Please enter an integer.")

            row1['label'] = new_label
            merged_data.append(row1.tolist())

    merged_df = pd.DataFrame(merged_data, columns=df1.columns)
    merged_df.to_csv(output_file, index=False)
    print(f"\nMerged CSV saved as {output_file}")
```

Conclusion

This approach ensures a systematic and reliable way to assign the ground truth labels to the dataset. The combination of majority voting and manual intervention allows for a good way to resolve annotation disagreements.

Ground Truth Labels Distribution

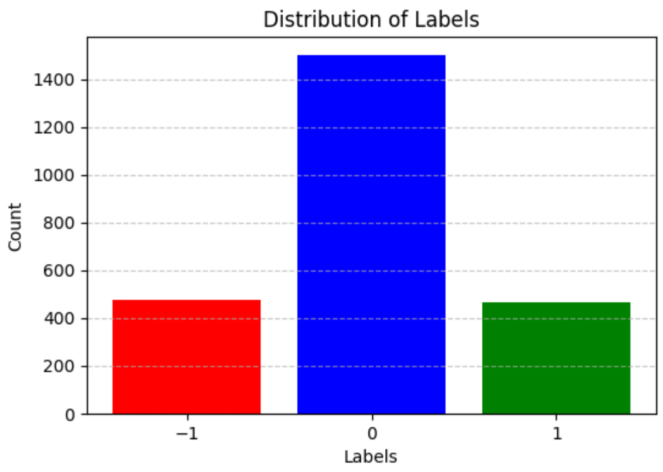

Figure 1: Distribution of labels.

This visualization showcases the distribution of data points across each ground truth label. It is evident that there is a noticeable imbalance between the neutral label and the positive and negative labels. The positive and negative labels contain approximately the same number of data points, with 468 and 476, respectively. In contrast, the neutral label has a significantly higher count of 1,504 data points.

Create Baseline

As a group, we chose to use randomly assigned labels as our baseline, as this approach yielded better overall results. This outcome is reasonable given the class imbalance in our dataset. Had we opted for the majority-class baseline, which assigns the most frequent label to each data file, all instances would have been labeled as neutral (zero) since it is the dominant class across our dataset. The majority vote baseline would introduce bias and skew the results. Since most of our labels are neutral, a model that simply predicts the majority class would appear more effective than it actually is, giving a misleading impression of its performance.

To evaluate model performance, we trained a Feedforward Neural Network on both human-annotated data and randomly labeled data. The results demonstrated a significant performance gap. When trained on human-annotated data, the model achieved an accuracy of approximately 0.64. In contrast, training on randomly labeled data resulted in a much lower accuracy score of 0.33.

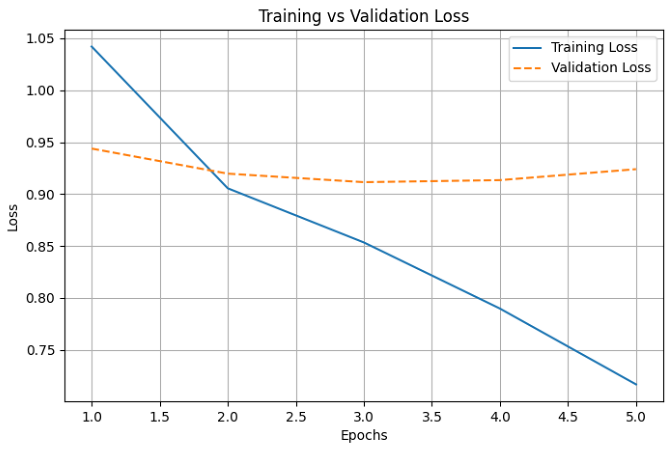

Figure 2: Training vs. Validation Loss for the Trained Model.

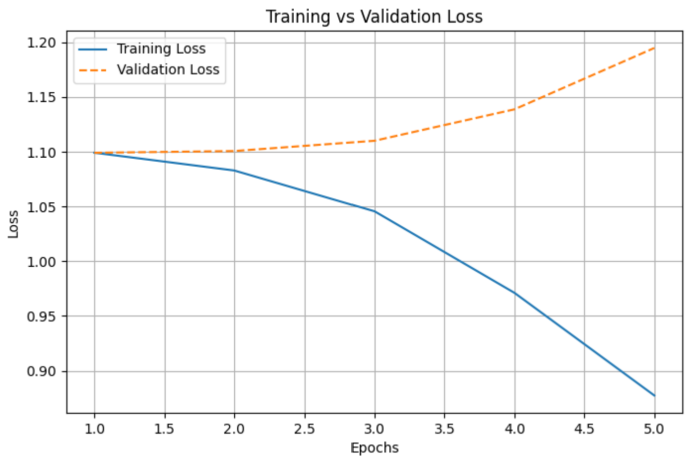

Figure 3: Training vs. Validation Loss for the Baseline Model.

As observed from the training vs. validation loss graphs, the trained model significantly outperforms the baseline model.

Conclusion

To further improve the performance of our Feedforward Neural Network in the text classification task, we could explore alternative feature extraction techniques. Currently, we use TF-IDF for feature extraction, but in the future, implementing a word embedding model with mean-pooled embeddings could yield better results.

Additionally, further tuning of the neural network's hyperparameters—such as the number of hidden dimensions, epochs, and learning rate—could enhance model performance.

Google Slide Link

Use this link to view the google slide: here

Codabench Link

The following Codabench link allows you to run a benchmark using your model's output labels for the test set. **No actual model is required to run the benchmark; only the label outputs generated by running the model locally are needed.**

- **Benchmark Using Model Output:**
  Use this link to run a benchmark on the human-annotated data against your own model outputs for the test set: here

If you wish to run the benchmark for either dataset (human annotated or random baseline) using a `model.py` file, you can use the following link:

- **Benchmark with a `model.py` file:**
  Use this link to run a benchmark with your `model.py` file for either the model predicted data or random baseline predicted data: here