Categorization and Sub-Categorization of Email-Chains Using Deep Learning

Arvind Nandigam

JULY 2025

Abstract

In this project, I have used Deep Learning neural networks to categorize and further sub-categorize a email chain dataset, which consists of 5438 rows of email chain data spread across 5 columns namely, Email subject, Email Query Discerption, Query Category, Query Item and Action Owner.

I have employed the usage of hugging-face transformers and knowledge distillation along with employing nlpaug for data augmentation. I have use BERT-base-uncased for data augmenting using the nlpaug library and used BERT-base-multilingual-cased as the teacher model that trained on the augmented data.

The student model in this case is distil-Bert-cased which learns from the teacher model by the process of knowledge distillation for a more efficient and light-weight model solution and enables efficient inference while retaining its high accuracy.

1.Introduction

During my internship in FLEX , I was given a dataset to work on which consists of Email Chains that I had to categorize and sub-categorize. This required the utilization of libraries such as NumPy, pandas, matplotlib, collections, sklearn, torch, transformers, tqdm, re and nlpaug. Re, pandas, NumPy, tqdm, and matplotlib were used to organize, clean, and visualize the data. I used data augmentation techniques with the nlpaug library to address data imbalance issues, which improved the robustness of the model. I used transformer-based architectures to train the model, using 80% of the data that was randomly chosen for training and the remaining 20% for testing. Following the first training phase, the pretrained model and the original dataset were used to refine a second transformer model by the process of knowledge distillation . During real-time inference, this method sought to minimize resource consumption (such as memory space and computation time) while optimizing model performance.

2.Theory

2.1 Data Augmentation using nlpaug

We used data augmentation with the nlpaug library to fix the problem of class imbalance in the training data by substituting words with words which matched the context. This method uses a pretrained BERT model (bert-base-uncased) to make new sentences by swapping out some words for ones that fit the context better.

For instance, "The delivery was late because of traffic" could be changed to "The shipment was delayed because of congestion." These added examples keep the same meaning but add more words to the mix. The model can learn from all categories better by adding these kinds of variations to classes that are not well represented. This makes generalization better and performance more balanced during inference.

2.2 Knowledge Distillation with BERT

To reduce model size while maintaining predictive performance, knowledge distillation is employed. This technique transfers knowledge from a large, expressive teacher model (BERT-base) to a smaller student (distil-BERT) model, making the latter more suitable for deployment in resource-constrained environments.

Unlike traditional supervised learning, where models are trained solely on hard labels, distillation leverages the soft probability distributions (logits) produced by the teacher model.

A temperature-scaled SoftMax function is applied to the logits of both the teacher and student models to smooth the output distributions, capturing richer inter-class relationships.

The student model is trained using a composite loss function that includes both the standard cross-entropy loss with the true labels and the Kullback–Leibler (KL) divergence between the teacher's and student's softened output probabilities. This dual-supervision framework enables the student model to not only learn the correct class but also approximate the confidence levels and decision boundaries of the teacher.

The result is a light-weight model that offers improved efficiency in terms of memory and computation, while preserving much of the predictive power of the original large-scale model.

This makes it ideal for real-time applications or scenarios where computational resources are limited.

3.Methodology

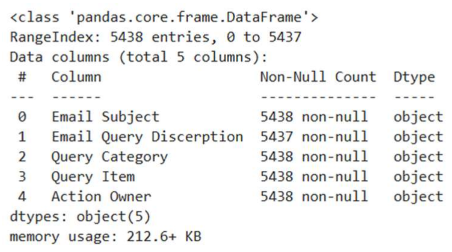
We use 2 different jupyter notebooks – one for main categorization and another for sub-categorization.

3.1 Main categorization Notebook

3.1.1 Pre-processing – EDA, cleaning of data

We begin with reading "AG – AI 2.xlsx" using read_excel using pandas.

We proceed to concatenate both columns Email Subject and Email Query Discerption into a new text column.


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5438 entries, 0 to 5437
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   Email Subject           5438 non-null   object
 1   Email Query Discerption 5437 non-null   object
 2   Query Category          5438 non-null   object
 3   Query Item              5438 non-null   object
 4   Action Owner            5438 non-null   object
dtypes: object(5)
memory usage: 212.6+ KB
```

We observe there are some missing values, so we also drop columns with NaN values.

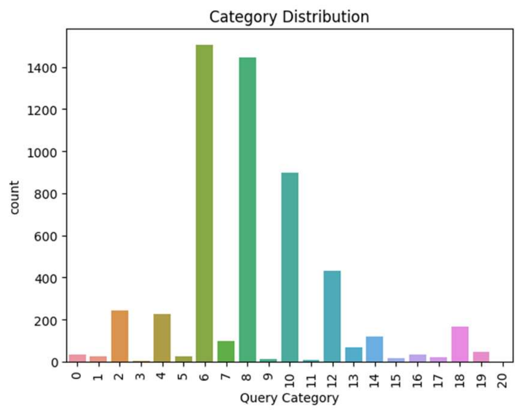
I will also drop the column "Query Item" which consists of subcategories. "Query Category" is preserved as it consists of main categories.

I am going to proceed with cleaning the data in "text" column such as irregular Unicode, links, phone numbers, from, to, thank you, etc… and keep only relevant information.

I am observing that many categories in "Query category" differ in cases only such as "Payments, payments" so we map all of them into a common case.

I am going to proceed with label Encoding the "Query category" column and visualize it.

The below is the class distribution of the column “Query Category”: -



3.1.2 Data Augmentation

To address class imbalance within the Query Category labels, contextual text augmentation was performed using the ContextualWordEmbsAug module from the nlpaug library. The augmentation was guided by a pretrained bert-base-uncased model, which substitutes selected words in a sentence with contextually relevant alternatives based on BERT’s masked language modelling.

A class-wise balancing strategy was used. First, the maximum class frequency across all Query Category labels was identified. Then, for each minority class, a sufficient number of samples were synthetically generated to match the maximum class size. Specifically, sentences from the underrepresented class were duplicated and passed through the BERT-based augmenter until the class was upsampled to the desired count.

Before Augmentation :

```
Query Category
6      1505
8      1447
10      896
12      431
2       243
4       225
18      168
14      120
7        97
13       68
19       48
0        36
16       36
5        26
1        25
17       23
15       15
9        14
11        8
3         5
20        2
Name: count, dtype: int64
```

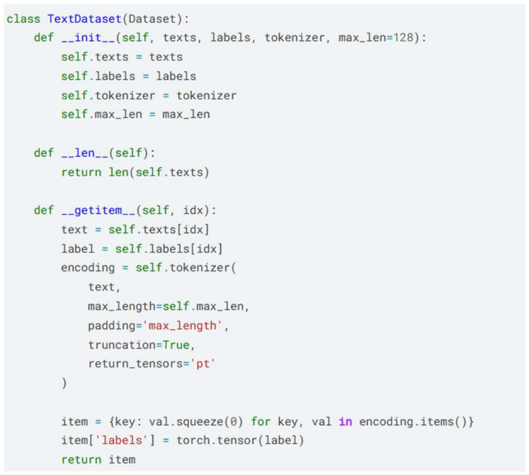
After data Augmentation :

```
Query Category
12     1505
8      1505
6      1505
10     1505
3      1505
14     1505
4      1505
19     1505
2      1505
16     1505
18     1505
15     1505
13     1505
0      1505
5      1505
1      1505
11     1505
17     1505
9      1505
7      1505
20     1505
Name: count, dtype: int64
```

We will now concatenate the newly generated rows in to the existing data and save it as "aug_df.csv" for future reference if necessary.

3.1.3 Dataset creation and division

We are going to store all the texts in "text" column as a list and doing the same with "Query Category" column. We are then going to proceed with splitting the data in 80:20 ratio for training and testing purposes respectively. We also use stratify to make sure all the classes are represented in the training data.



```
class TextDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=128):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = self.texts[idx]
        label = self.labels[idx]
        encoding = self.tokenizer(
            text,
            max_length=self.max_len,
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )

        item = {key: val.squeeze(0) for key, val in encoding.items()}
        item['labels'] = torch.tensor(label)
        return item
```

TextDataset Class

We are then going to form a TextDataset class that inputs texts, labels and tokenizer (We are going to use teacher tokenizer).

We are now going to have a train dataset and validation dataset; we are going to use DataLoader from torch to now input datasets and batch sizes of 16 and 32 respectively for train and validation .

3.1.4 Model Training (Teacher)

Load the teacher model (bert-base-multilingual-cased) from hugging face transformers

To train the teacher model, we used the AdamW optimizer with a learning rate of 5e-5 and a weight decay of 0.0005.

To help the model adjust its learning rate based on performance, we added a ReduceLROnPlateau scheduler. This scheduler monitors the validation loss. If there is no improvement for a full epoch, it reduces the learning rate by a factor of 0.001. We also set a very low limit for the learning rate at 1e-50 to stop it from getting too small, which could lead to instability.

The model was trained for up to 10 epochs, with early stopping in place to avoid overfitting. Training would stop if the validation loss did not improve by at least 0.0001 for 10 straight epochs.

Each epoch involved looping over batches from the training set. For every batch, the model entered training mode and processed input tokens, attention masks, and ground-truth labels, all loaded onto the GPU. It generated predictions, known as logits, which we compared to the true labels using a cross-entropy loss function. This loss was then backpropagated, and the model's parameters were updated with the optimizer.

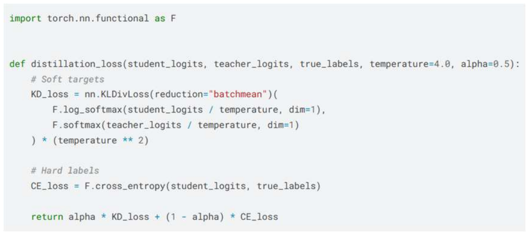
We tracked the loss and accuracy for each batch to assess overall training performance for the epoch. After each epoch, we switched the model to evaluation mode and tested it on a separate validation set. During this phase, we did not compute gradients to save memory and speed up inference. We calculated the validation loss and accuracy across all batches. This information helped adjust the learning rate and determine if early stopping should be triggered.

we logged key metrics such as training loss, training accuracy, validation loss, and validation accuracy at the end of each epoch.

3.1.5 Data Loading and loss (Student)

We are then going to form a Datasets (train and test) that inputs texts, labels and tokenizer (We are going to use student tokenizer).

We are going to load up the dataset using DataLoader and set the teacher to eval mode.



```python
import torch.nn.functional as F


def distillation_loss(student_logits, teacher_logits, true_labels, temperature=4.0, alpha=0.5):
    # Soft targets
    KD_loss = nn.KLDivLoss(reduction="batchmean")(
        F.log_softmax(student_logits / temperature, dim=1),
        F.softmax(teacher_logits / temperature, dim=1)
    ) * (temperature ** 2)

    # Hard labels
    CE_loss = F.cross_entropy(student_logits, true_labels)

    return alpha * KD_loss + (1 - alpha) * CE_loss
```

Distillation loss function

We are going to define distillation loss as a combination of KDDivLoss of teacher logits,student logits and cross entropy loss of hard labels and student logits.

3.1.6 Model Training (Student)

Load the teacher model (distil-bert) from hugging face transformers

To train the student model , we are going to use the same AdamW optimizer and same ReduceLROnPlateau scheduler as the teacher's.

The loss is defined as newly defined distillation loss .

The Knowledge distillation take places between Teacher and student over 10 epochs.

Each epoch is going to involve looping over batch from the training set teacher outputs are then converted into teacher logits. Using the same input ids and attention masks , student outputs are extracted and converted to student logits.

The loss is calculated. This loss was then backpropagated, and the model's parameters were updated with the optimizer.

As training continued, we tracked the loss and accuracy for each batch to assess overall training performance for the epoch. After each epoch, we switched the model to evaluation mode and tested it on a separate validation set.

3.1.7 Final Evaluation and Model Saving

A final evaluation loop is set to compare the accuracy of both teacher and student model using the respective data loading variable.

The final Accuracy and loss is logged.

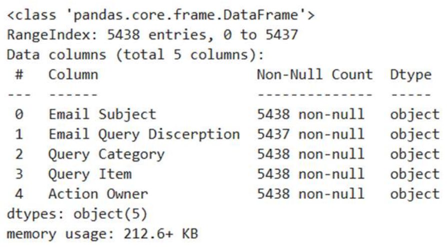
The final Student model is saved using saved_pretrained function.

3.2 Sub categorization Notebook

3.2.1 Pre-processing – EDA, cleaning of data

We begin with reading "AG – AI 2.xlsx" using read_excel using pandas.

We proceed to concatenate both columns Email Subject and Email Query Discerption into a new text column.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5438 entries, 0 to 5437
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   Email Subject           5438 non-null   object
 1   Email Query Discerption 5437 non-null   object
 2   Query Category          5438 non-null   object
 3   Query Item              5438 non-null   object
 4   Action Owner            5438 non-null   object
dtypes: object(5)
memory usage: 212.6+ KB
```

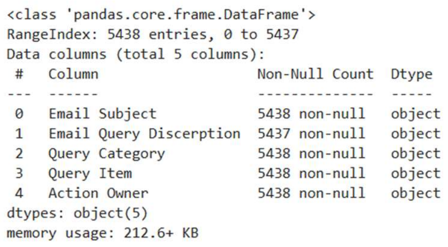
We observe there are some missing values, so we also drop columns with NaN values.

I will also drop the column "Query Category" which consists of main categories. "Query Item" is preserved as it consists of sub-categories.

I am going to proceed with cleaning the data in "text" column such as irregular Unicode, links, phone numbers, from, to, thank you, etc… and keep only relevant information.

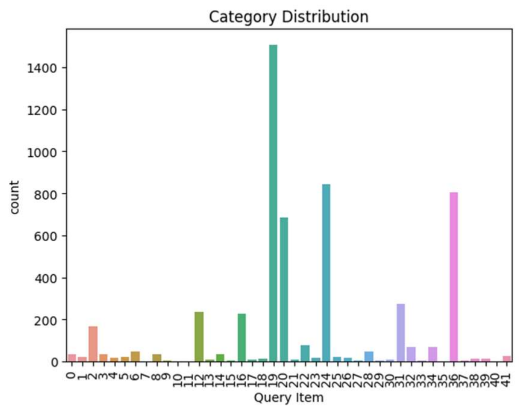
The values in "Query Item" is in the form of "main category-subcategory" I will extract subcategory and discard the rest in each of the "Query Item" column.

I am going to proceed with label Encoding the "Query Item" column and visualize it.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5438 entries, 0 to 5437
Data columns (total 5 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Email Subject            5438 non-null   object
 1   Email Query Discerption  5437 non-null   object
 2   Query Category           5438 non-null   object
 3   Query Item               5438 non-null   object
 4   Action Owner             5438 non-null   object
dtypes: object(5)
memory usage: 212.6+ KB
```

The below is the class distribution of the column "Query Item": -

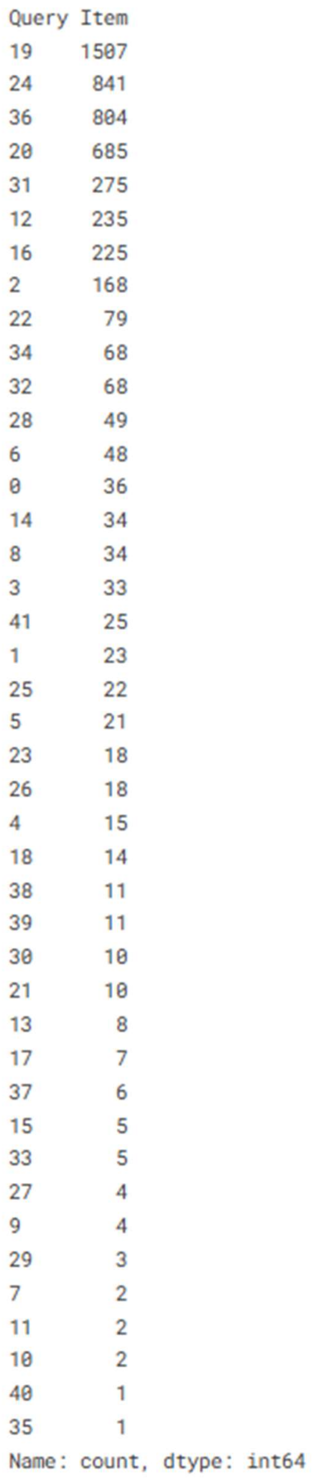


3.1.2 Data Augmentation

To address class imbalance within the Query Item labels, contextual text augmentation was performed using the ContextualWordEmbsAug module from the nlpaug library. The augmentation was guided by a pretrained bert-base-uncased model, which substitutes selected words in a sentence with contextually relevant alternatives based on BERT's masked language modelling.

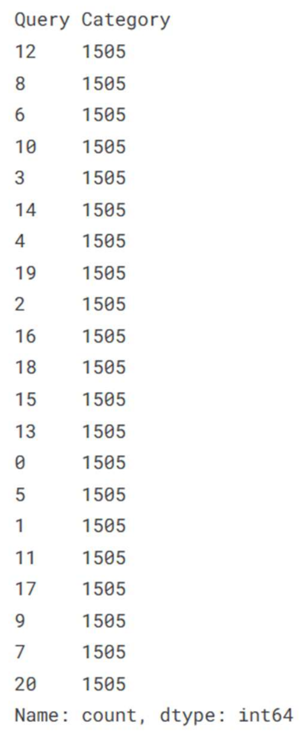
A class-wise balancing strategy was used. First, the maximum class frequency across all Query Item labels was identified. Then, for each minority class, a sufficient number of samples were synthetically generated to match the maximum class size. Specifically, sentences from the underrepresented class were duplicated and passed through the BERT-based augmenter until the class was up sampled to the desired count.

Before Augmentation :



```
Query Item
19    1507
24     841
36     804
20     685
31     275
12     235
16     225
2      168
22      79
34      68
32      68
28      49
6       48
0       36
14      34
8       34
3       33
41      25
1       23
25      22
5       21
23      18
26      18
4       15
18      14
38      11
39      11
30      10
21      10
13       8
17       7
37       6
15       5
33       5
27       4
9        4
29       3
7        2
11       2
10       2
40       1
35       1
Name: count, dtype: int64
```

After data Augmentation :



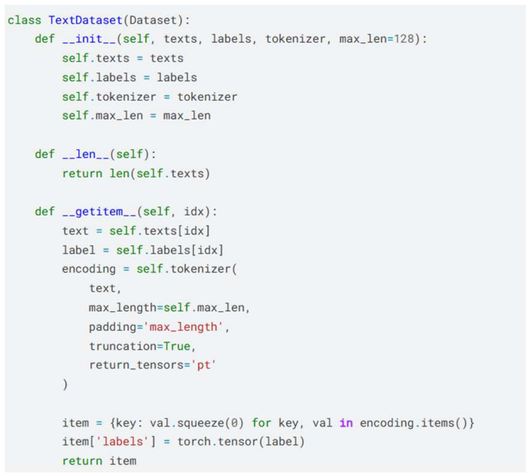
```
Query Category
12    1505
8     1505
6     1505
10    1505
3     1505
14    1505
4     1505
19    1505
2     1505
16    1505
18    1505
15    1505
13    1505
0     1505
5     1505
1     1505
11    1505
17    1505
9     1505
7     1505
20    1505
Name: count, dtype: int64
```

We will now concatenate the newly generated rows in to the existing data and save it as “aug_df.csv” for future reference if necessary.

3.2.3 Dataset creation and division

We are going to save columns “text” and “Query Item” in the form of a list and split it in the 80:20 ratio for training and validation purposes .

We are now going to define a TextDataset class that holds texts,labels and tokenizers.



```python
class TextDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=128):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = self.texts[idx]
        label = self.labels[idx]
        encoding = self.tokenizer(
            text,
            max_length=self.max_len,
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )

        item = {key: val.squeeze(0) for key, val in encoding.items()}
        item['labels'] = torch.tensor(label)
        return item
```

TextDataset Class

We define train TextDataset and validation TextDataset using respective texts and labels and teacher tokenizer because we are going to train teacher model.

We are further going to load the dataset into DataLoader with batch size 16 and 32 respectively.

3.2.4 Model Training (Teacher)

Load the teacher model (bert-base-multilingual-cased) from hugging face transformers

To train the teacher model, we used the AdamW optimizer with a learning rate of 5e-5 and a weight decay of 0.0005.

To help the model adjust its learning rate based on performance, we added a ReduceLROnPlateau scheduler. This scheduler monitors the validation loss. If there is no improvement for a full epoch, it reduces the learning rate by a factor of 0.001. We also set a very low limit for the learning rate at 1e-50 to stop it from getting too small, which could lead to instability.

The model was trained for only 2 epochs due to notebook time constraints of 12hours , we are going to get input ids , attention masks and labels of each batch. We are going to zero out previous gradients if any , calculate outputs and convert it to logits. Loss is calculated and back propagated with optimizer taking a step towards convergence.

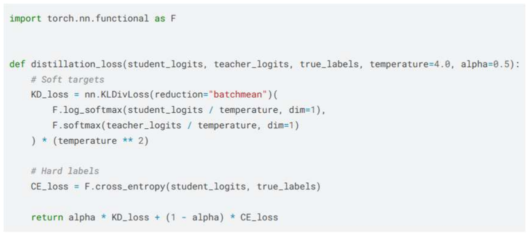
The final training loss, training accuracy are calculated. The model is set to eval mode to calculate validation loss and validation accuracy. All of these are logged. If there is no improvement in validation loss then scheduler take a step in reducing the learning rate.

If there is no improvement in 10 epochs in validation loss at all then Early stopping is triggered. We reset the model to training model at the end of each epoch.

3.2.5 Data Loading and loss (Student)

We are then going to form a Datasets (train and test) that inputs texts, labels and tokenizer (We are going to use student tokenizer).

We are going to load up the dataset using DataLoader and set the teacher to eval mode.



```
import torch.nn.functional as F


def distillation_loss(student_logits, teacher_logits, true_labels, temperature=4.0, alpha=0.5):
    # Soft targets
    KD_loss = nn.KLDivLoss(reduction="batchmean")(
        F.log_softmax(student_logits / temperature, dim=1),
        F.softmax(teacher_logits / temperature, dim=1)
    ) * (temperature ** 2)

    # Hard labels
    CE_loss = F.cross_entropy(student_logits, true_labels)

    return alpha * KD_loss + (1 - alpha) * CE_loss
```

Distillation loss function

We are going to define distillation loss as a combination of KDDivLoss of teacher logits,student logits and cross entropy loss of hard labels and student logits.

3.2.6 Model Training (Student)

Load the teacher model (distil-bert) from hugging face transformers

To train the student model , we are going to use the same AdamW optimizer and same ReduceLROnPlateau scheduler as the teacher's.

The loss is defined as newly defined distillation loss .

The Knowledge distillation take places between Teacher and student over 2 epochs.

Each epoch is going to involve looping over batch from the training set teacher outputs are then converted into teacher logits. Using the same input ids and attention masks , student outputs are extracted and converted to student logits.

The loss is calculated. This loss was then backpropagated, and the model's parameters were updated with the optimizer.

As training continued, we tracked the loss and accuracy for each batch to assess overall training performance for the epoch. After each epoch, we switched the model to evaluation mode and tested it on a separate validation set.

3.2.7 Final Evaluation and Model Saving

A final evaluation loop is set to compare the accuracy of both teacher and student model using the respective data loading variable.

The final Accuracy and loss is logged.

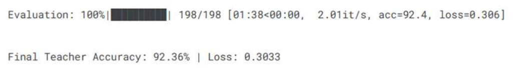
The final Student model is saved using saved_pretrained function.

4.Results

Now the final results in the notebook during testing is :

4.1 Main Categorization Results

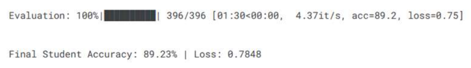
Teacher model accuracy and loss:



```
Evaluation: 100%|██████████| 198/198 [01:38<00:00,  2.01it/s, acc=92.4, loss=0.306]

Final Teacher Accuracy: 92.36% | Loss: 0.3033
```

Student model accuracy and loss:

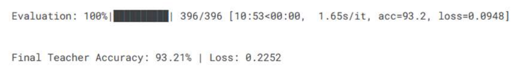


```
Evaluation: 100%|██████████| 396/396 [01:30<00:00,  4.37it/s, acc=89.2, loss=0.75]

Final Student Accuracy: 89.23% | Loss: 0.7848
```

4.2 Sub Categorization Results

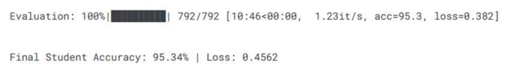
Teacher model accuracy and loss:



```
Evaluation: 100%|██████████| 396/396 [10:53<00:00,  1.65s/it, acc=93.2, loss=0.0948]

Final Teacher Accuracy: 93.21% | Loss: 0.2252
```

Student model accuracy and loss:



```
Evaluation: 100%|██████████| 792/792 [10:46<00:00,  1.23it/s, acc=95.3, loss=0.382]

Final Student Accuracy: 95.34% | Loss: 0.4562
```

5.Conclusion

The results of this project lead to the following conclusions:

- Sub categorization results are more accurate than main categorization results due to exponential data augmentation of the sub categorization data.
- The action owner column is irrelevant and can be dropped.

Acknowledgements

I would like to thank Mr. Manoj Ravunni for his exceptional mentorship and guidance during my internship in FLEX. His support and encouragement made a meaningful impact in the successful completion of this project and in my professional learning journey.

My heartfelt thanks to FLEX for I am grateful for this opportunity to grow professionally in this such a supportive environment. I sincerely thank my colleagues for their exceptional guidance throughout the internship.