Individual Final Report: Math Problem Classification using Transformer Models

Name: Prudhvi Chekuri

Group: 8

1. Introduction

This report details work undertaken as part of the NLP DATS_6312 final project. The project aimed to tackle the challenge of classifying math problems into one of eight distinct categories (Algebra, Geometry & Trigonometry, Calculus & Analysis, Probability & Statistics, Number Theory, Combinatorics & Discrete Math, Linear Algebra, Abstract Algebra & Topology) based on their textual content. The primary dataset was sourced from the Kaggle competition "Classification of Math Problems by Kasut Academy".

The project explored multiple Natural Language Processing (NLP) techniques and transformer architectures. Key areas of exploration included:

- Getting the best out of classical methods for math problem classification.
- Fine-tuning standard sequence classification models (DistilBERT, DeBERTa-V3).
- Framing the classification task as sequence-to-sequence text generation (T5).
- Fine-tuning a decoder-only Large Language Model (LLM) using instruction prompting (Llama 3.2 1B with Unsloth/LoRA).
- Augmenting the training data using generative AI (AWS Bedrock with Claude 3.5 Sonnet).
- Combining predictions from multiple models using an ensemble approach (Hard Voting).
- Developing and deploying a Streamlit application on AWS that allows users to test our models and use for Math Problem Classification.

This report focuses on the individual contributions made towards these explorations, particularly concerning **data augmentation, different transformer architectures, ensemble implementation.**

2. Description of Individual Work (Background & Methodology)

This section provides background on the core techniques and models employed in my individual contributions to the project. The work spanned several approaches to tackle the math problem classification task.

2.1. Sequence Classification (DeBERTa-V3)

- **Background:** Sequence classification is a common NLP task where the goal is to assign a predefined category label to a sequence of text (in this case, a math problem). Models typically process the input text and output probabilities for each potential class.
- **Methodology (DeBERTa):** The microsoft/deberta-v3-base model was employed. DeBERTa (Decoding-enhanced BERT with Disentangled Attention) is an advanced encoder-only transformer architecture that builds upon models like BERT and RoBERTa. Its key innovation is the **disentangled attention mechanism**, which represents each word using separate vectors for content and relative position, allowing the attention mechanism to explicitly model interactions between content and position.

2.2. Sequence-to-Sequence Formulation (T5)

- **Background:** The Text-to-Text Transfer Transformer (T5) model frames all NLP tasks as a text-to-text problem. Given an input text (often prefixed with task instructions), the model generates a target text sequence as output.
- **Methodology (T5):** The t5-base model, featuring an encoder-decoder architecture, was used. The classification task was reformulated: the input was the math problem prefixed with "Classify this math problem: ", and the model was trained to generate the *textual name* of the correct category (e.g., "Algebra", "Number Theory") as the target sequence. Fine-tuning utilized the

Hugging Face Seq2SeqTrainer, optimizing the model to produce the correct label text.

2.3. Instruction Fine-tuning Decoder-Only LLMs (Llama 3.2 1B)

- **Background:** Decoder-only Large Language Models (LLMs) like Llama excel at text generation and following instructions. They can be adapted for classification by formulating the task as instruction following within a specific prompt template, where the model generates the class label as part of its response.
- **Methodology (Llama):**
    - The unsloth/Llama-3.2-1B model was fine-tuned.
    - **Efficient Fine-tuning:** The unsloth library was used alongside Parameter-Efficient Fine-Tuning (PEFT) via Low-Rank Adaptation (LoRA, r=16) and 4-bit quantization to enable fine-tuning on available hardware with reduced memory and computation requirements.
    - **Supervised Fine-tuning (SFT):** The trl library's SFTTrainer was used to fine-tune the model on a dataset formatted with a specific instruction prompt (containing instruction, input math problem, and target response label name).

2.4. Data Augmentation using Generative AI (AWS Bedrock)

- **Background:** Class imbalance in the original dataset motivated the use of data augmentation. Generative LLMs were employed to create synthetic examples for underrepresented categories.
- **Methodology:** AWS Bedrock provided access to the Claude 3.5 Sonnet model. Scripts (data_generation_*.ipynb) were developed to prompt this model, providing examples from a target category and requesting the generation of new, diverse problems matching the topic, complexity, and (in the "specific" version) the mathematical notation style (LaTeX).

2.5. Ensemble Methods (Hard Voting)

- **Background:** Ensembling combines predictions from multiple diverse models, often leading to improved generalization and robustness compared to single models.
- **Methodology:** A Hard Voting strategy was implemented (nlp-final-project-ensemble-model.ipynb). This involved taking the predictions from the fine-tuned DeBERTa, T5, and Llama models and selecting the label predicted most frequently. A tie-breaking rule defaulting to the DeBERTa prediction was used when all three models disagreed because of its performance.

3. Detailed Description of Individual Work Portion

- **Data Generation (data_generation_specific.ipynb):**
  - Developed the Python script utilizing boto3 to interact with the AWS Bedrock API (Claude 3.5 Sonnet).
  - Designed and refined the prompt structure, specifically adding the constraint to preserve LaTeX notation based on observing outputs from the generic version.
  - Implemented logic for calculating needed samples, selecting examples, parallel generation (ThreadPoolExecutor), and API error handling/retries.
  - Wrote code to parse delimited API output and save the augmented dataset (augmented_train_*.csv).
- **DeBERTa-V3 (nlp-final-project-nn-deberta-v3-base-augmented.ipynb):**
  - Implemented the sequence classification pipeline using Hugging Face transformers.
  - Wrote the clean_math_text_final function for preprocessing.

    - Trained the model on the original training data and the augmented data.
    - Configured Dataset and DatasetDict creation with stratified train/validation/test splits (80/10/10) from train.csv.
    - Set up AutoTokenizer and AutoModelForSequenceClassification for microsoft/deberta-v3-base.
    - Defined the compute_metrics function using the evaluate library for accuracy and F1 scores.
    - Configured TrainingArguments including 10 epochs, learning rate 2×10^{-5}, batch size 8, and FP16 mixed precision.
    - Executed the fine-tuning process using the Trainer.
    - Implemented the code to run predictions on the competition test set (test.csv) and generate the submission.csv file.
- **T5 Fine-tuning (nlp-final-project-t5.ipynb):**
    - Implemented the sequence-to-sequence classification approach.
    - Set up the mapping between numeric labels and their string names.
    - Prepared Dataset objects from train.csv with a 90/10 train/validation split. Trained on just the original training data.
    - Wrote the preprocess_function to add the task prefix ("Classify this math problem: ") and tokenize both input questions and target label names using the t5-base tokenizer.
    - Configured DataCollatorForSeq2Seq.
    - Implemented the custom compute_metrics function involving decoding generated text labels, mapping them back to IDs, and calculating accuracy.

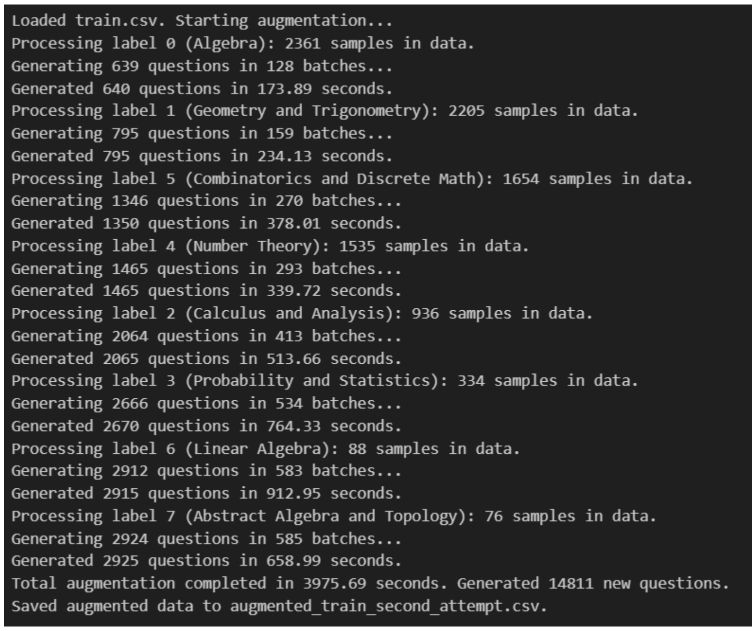
    - Configured Seq2SeqTrainingArguments including 10 epochs, learning rate 5×10−5, batch size 8, FP16 mixed precision, and enabling predict_with_generate.
    - Executed the fine-tuning using Seq2SeqTrainer.
    - Implemented the prediction logic on the competition test set using a text2text-generation pipeline.
    - Wrote the code to parse the generated label names, convert them back to numeric IDs (handling unknowns), and generate submission.csv.
- **Llama 1B Fine-tuning (LLAMA_1B.ipynb):**
    - Set up the fine-tuning environment using unsloth.
    - Configured FastLanguageModel for loading unsloth/Llama-3.2-1B with 4-bit quantization and adding LoRA adapters (r=16).
    - Implemented the formatting_prompts_func to structure the combined training data (original subset + augmented data) into the required instruction format, ensuring the inclusion of EOS_TOKEN.
    - Configured TrainingArguments for the SFTTrainer (max_steps=640, learning rate 2×10−4, effective batch size 32, 8-bit AdamW optimizer).
    - Executed the SFT training process.
    - Implemented the inference loop for the internal hold-out set and developed the parse_output regex function to extract predictions.
    - Calculated accuracy on the hold-out set.
    - Adapted the inference process for the final competition test set and generated predictions.
    - Implemented logic to convert generated label names to numeric IDs for submission.

- **Ensemble Implementation (nlp-final-project-ensemble-model.ipynb):**
    - Structured the notebook to sequentially load each required model (Llama, T5, DeBERTa) from pre-saved checkpoints.
    - Implemented the model-specific preprocessing and prediction pipelines within this notebook for the competition test set.
    - Wrote the Python code to perform the hard voting logic, combining predictions from the three models and implementing the DeBERTa tie-breaker rule.
    - Generated the final submission.csv based on the ensemble results.

4. Results

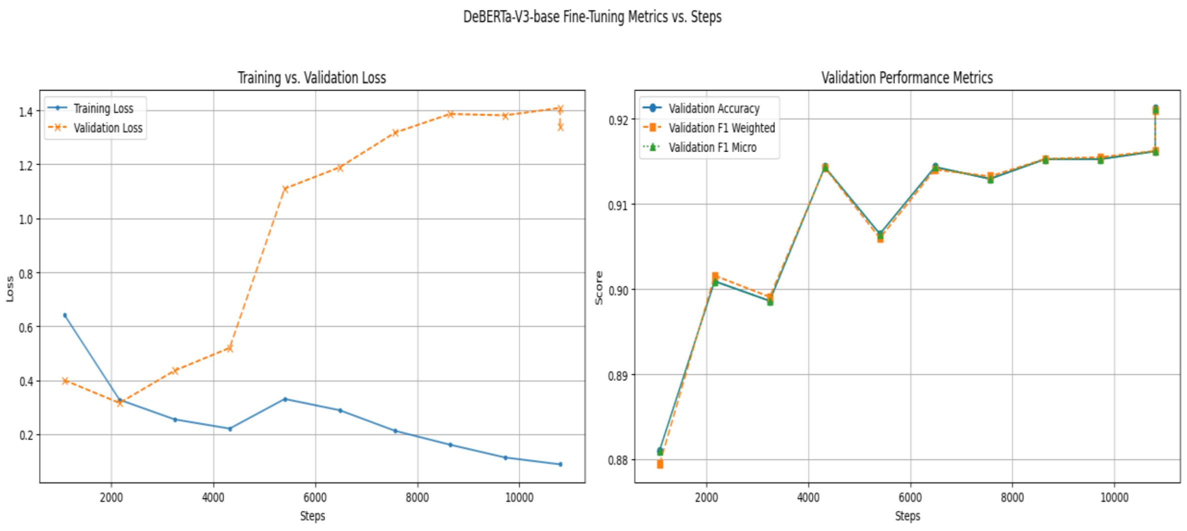
The experiments yielded the following key results:

- **Data Augmentation:** The data generation scripts successfully produced [Number, e.g., ~11k-14k] synthetic samples using AWS Bedrock, helping to balance the training dataset classes towards the target of 3000 samples each.
- **Output of** data_generation_specific.ipynb:



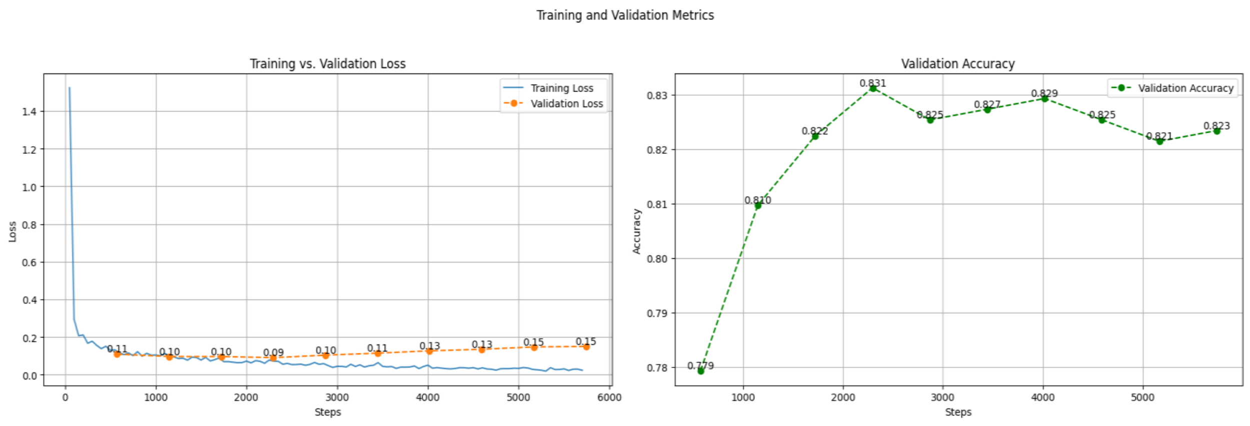
```
Loaded train.csv. Starting augmentation...
Processing label 0 (Algebra): 2361 samples in data.
Generating 639 questions in 128 batches...
Generated 640 questions in 173.89 seconds.
Processing label 1 (Geometry and Trigonometry): 2205 samples in data.
Generating 795 questions in 159 batches...
Generated 795 questions in 234.13 seconds.
Processing label 5 (Combinatorics and Discrete Math): 1654 samples in data.
Generating 1346 questions in 270 batches...
Generated 1350 questions in 378.01 seconds.
Processing label 4 (Number Theory): 1535 samples in data.
Generating 1465 questions in 293 batches...
Generated 1465 questions in 339.72 seconds.
Processing label 2 (Calculus and Analysis): 936 samples in data.
Generating 2064 questions in 413 batches...
Generated 2065 questions in 513.66 seconds.
Processing label 3 (Probability and Statistics): 334 samples in data.
Generating 2666 questions in 534 batches...
Generated 2670 questions in 764.33 seconds.
Processing label 6 (Linear Algebra): 88 samples in data.
Generating 2912 questions in 583 batches...
Generated 2915 questions in 912.95 seconds.
Processing label 7 (Abstract Algebra and Topology): 76 samples in data.
Generating 2924 questions in 585 batches...
Generated 2925 questions in 658.99 seconds.
Total augmentation completed in 3975.69 seconds. Generated 14811 new questions.
Saved augmented data to augmented_train_second_attempt.csv.
```

- **Llama 1B Fine-tuning:**
    - The unsloth/Llama-3.2-1B model fine-tuned with LoRA on the combined original and augmented dataset achieved an accuracy of **85.1%** on the internal hold-out set (last 1000 samples of train.csv).
    - The public Kaggle score for this model varied slightly depending on the run/hardware, achieving **~0.820 - 0.835**.
    - These scores are consistent across datasets which shows that the model is not overfitting and generalizing well.
- **DeBERTa-V3 (nlp-final-project-nn-deberta-v3-base-augmented.ipynb):**
    - The microsoft/deberta-v3-base model, fine-tuned for 10 epochs on the original train.csv using a sequence classification approach, achieved strong performance.



    - These training clearly shows this extended training for 10 epochs is not necessary and the model is overfitting after 4000 steps based on the loss curves.
    - Stopping the training at epoch 4 can still achieve an accuracy >91, which also reduces the training time by half.
    - The submission generated from this notebook achieved a public Kaggle score of **0.851**, making it the best-performing single model in this project.

- **T5 Fine-tuning (nlp-final-project-t5.ipynb):**
  - The t5-base model, fine-tuned using a sequence-to-sequence approach on the original train.csv to generate label names, showed reasonable performance.
  - On the internal validation set, the best accuracy achieved was approximately **83.1%**.



- These training curves shows the model is not improving after ~2500 stepos as the validation loss seems to go up and even the accuracy degrades after that point. Stopping the training at epoch 4 will prevent the overfitting and also saves 2X time.
- The submission based on this model yielded a public Kaggle score of **0.824**. This demonstrates the viability of the text-generation approach for this classification task, although it didn't outperform the dedicated sequence classifier (DeBERTa) in this instance.

- **Ensemble Model:**
  - The hard voting ensemble combined predictions from DeBERTa-v3-base, T5-base, and Llama-3.2-1B.
  - This ensemble achieved a public Kaggle score of **0.8588**.
  - This is our top submission on Kaggle.

5. Summary and Conclusions

- **Summary:** This project successfully explored multiple transformer-based approaches for classifying math problems. Techniques included standard sequence classification (DistilBERT, DeBERTa), sequence-to-sequence generation (T5), and instruction fine-tuning of a decoder-only LLM (Llama 1B with Unsloth/LoRA). Data augmentation using AWS Bedrock was implemented to address class imbalance. A hard voting ensemble of the DeBERTa, T5, and Llama models achieved the best performance with a public Kaggle score of 0.8588.
- **Learnings:**
  - Different model architectures and task framings (classification vs. generation) yield varying performance levels on this task.
  - Instruction fine-tuning decoder-only models like Llama is a viable approach for classification, achieving competitive results even with a relatively small model (1B parameters).
  - Libraries like unsloth coupled with techniques like LoRA and quantization significantly lower the barrier for fine-tuning LLMs on accessible hardware.
  - Generative AI (Bedrock/Claude) can be effectively prompted to create domain-specific synthetic data for augmentation, which actually improved score for us.
  - Ensembling diverse models remains a powerful technique for boosting performance, often exceeding the score of the best individual model. Although we made only a small improvement with our ensemble, making other two weak model (t5, llama 1b) can show a significant improvement in the ensemble predictions.

I have learnt a great deal from this project, so I have shared my work on Kaggle to help others.

- **Future Improvements:**
  - **Hyperparameter Tuning:** Systematically tune hyperparameters (learning rates, batch sizes, LoRA ranks, training steps/epochs) for each individual model.
  - **Cross-Validation:** Employ k-fold cross-validation during training and evaluation for more robust performance estimates.
  - **Preprocessing:** Develop a more sophisticated preprocessing pipeline specifically targeting mathematical notation (e.g., standardizing LaTeX, handling symbols).
  - **Model Exploration:** Experiment with larger models (e.g., DeBERTa-large, T5-large, larger Llama variants) if computational resources permit.
  - **Data Augmentation Validation:** Rigorously evaluate the quality of the generated data and its actual impact on model performance compared to training only on original data. Explore different generation models or prompting strategies.
  - **Reproducibility:** Implement consistent random seed setting across all notebooks from the beginning.
  - **Ensemble Strategy:** Explore weighted averaging or stacking instead of hard voting for the ensemble.

6. Code Percentage Calculation

Percentage = (Lines Copied-Lines Modified)(Lines Copied+Lines Added)×100

- For the Bert, T5 related codes, I have followed class materials, official hugging face documentation.
- But still the notebooks for those models has some codes that are taken from ChatGpt etc.

- I have extensively used GenAI for the data generation codes, since I don't know much about boto3.
- So, 20-30% of my code has AI generated code.
- For the Unsloth notebook, I have used official demo notebook, from which I have used around 55 lines of code, have added around 60 lines of code, and modified around 10 lines, so the calculation will be:

  (55-10) / (55+60) * 100 = 39

7. References

- **Competition:** Classification of Math Problems by Kasut Academy. (https://www.kaggle.com/competitions/classification-of-math-problems-by-kasut-academy/overview)
- **Models:**
  - Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.
  - He, P., Liu, X., Gao, J., & Chen, W. (2020). DeBERTa: Decoding-enhanced BERT with Disentangled Attention. *arXiv preprint arXiv:2006.03654*.
  - Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research, 21*(140), 1-67.
  - https://www.kaggle.com/code/danielhanchen/kaggle-llama-3-1-8b-unsloth-notebook

- **Libraries & Tools:**
  - Hugging Face Transformers: Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Brew, J. (2020). Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (pp. 38-45). (https://huggingface.co/docs/transformers)
  - Hugging Face Datasets: (https://huggingface.co/docs/datasets/)
  - Hugging Face TRL (Transformer Reinforcement Learning Library): (https://huggingface.co/docs/trl)
  - Hugging Face PEFT (Parameter-Efficient Fine-Tuning): (https://huggingface.co/docs/peft)
  - Unsloth AI: (https://github.com/unslothai/unsloth)
  - AWS Bedrock: (https://aws.amazon.com/bedrock/)
  - PyTorch: Paszke, A., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*.
  - Pandas: McKinney, W. (2010). Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference* (Vol. 445, pp. 51-56).
  - NumPy: Harris, C. R., et al. (2020). Array programming with NumPy. *Nature, 585*(7825), 357-362.
- **Techniques:**
  - LoRA: Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2021). LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685*.