Final Project Report: Classification of Mathematical Problems

1. Introduction

The ability to automatically categorize textual content is crucial in many domains. This project focuses on the specific challenge of classifying mathematical word problems into their respective subject areas, such as Algebra, Geometry, Calculus, Number Theory, Combinatorics, Probability & Statistics, Linear Algebra, and Discrete Mathematics. Classifying these problems can aid in educational resource management, targeted learning recommendations, and organizing large repositories of mathematical questions.

This report details the methodologies employed, experiments conducted, and findings from applying various Natural Language Processing (NLP) techniques to this classification task. We explored both traditional machine learning algorithms relying on TF-IDF features and modern deep learning approaches based on fine-tuning pre-trained transformer models.

This report is structured as follows:

- **Section 2:** Describes the dataset used for training and evaluation.
- **Section 3:** Details of the NLP models implemented, covering both classical algorithms and transformer architecture, including relevant background and formulations.
- **Section 4:** Outlines the experimental setup, including data handling, implementation frameworks, and performance evaluation metrics.
- **Section 5:** Discusses the hyperparameters tuned and the strategies used to mitigate overfitting.
- **Section 6:** Presents the results observed during the experiments, focusing on the types of comparisons made and metrics tracked (as specific numerical results are not available in the provided code outputs).
- **Section 7:** Summarizes the project, discusses key learnings, and suggests potential avenues for future work.
- **Section 8:** Provides references for algorithms, tools, and external resources used.

2. Description of the Data Set

The primary dataset for this project appears to originate from a Kaggle competition titled "Classification of Math Problems by Kasut Academy," as indicated by file paths like `/kaggle/input/classification-of-math-problems-by-kasut-academy/` found in the scripts. The dataset consists of:

- A training set (`train.csv`) containing mathematical questions (`Question` column) and their corresponding category labels (`label` column). The labels represent 8 distinct mathematical categories (0-7).
- A test set (`test.csv`) containing questions for which predictions are to be made, identified by an `id` column.
- A sample submission file (`sample_submission.csv`) showing the expected format for predictions.

Initial analysis of the training data revealed potential quality issues, such as questions starting with extraneous numbers or text (e.g., "Example...", "task..."), containing URLs or author names, suggesting they might have been scraped or automatically generated rather than manually curated.

Additionally, experiments were conducted using a paraphrased version of the dataset, indicated by filenames `train_pp.csv` and `test_pp.csv`. This suggests an effort to augment the data or test model robustness against variations in question phrasing. The paraphrased dataset required minimal cleaning compared to the original.

The distribution of labels in the training set was checked, and techniques to handle potential class imbalance (e.g., `class_weight="balanced"`, `is_unbalance = True`) were employed in some models.

3. Description of the NLP Model

Two main families of NLP models were implemented and evaluated:

a) Classical Machine Learning Models with TF-IDF Features:

These models represent text using statistical features derived from word frequencies.

- **TF-IDF (Term Frequency-Inverse Document Frequency):** This technique converts text documents into numerical vectors. The core idea is to assign weights to words based on their frequency within a document (Term Frequency) and their rarity across the entire corpus (Inverse Document Frequency). Words that are frequent in a specific document but rare overall are considered more informative and receive higher weights.
    - *Term Frequency (TF)*:
      <0>TF(t,d)=Total number of terms in document dNumber of times term t appears in document d
    - *Inverse Document Frequency (IDF)*:
      IDF(t,D)=log(Number of documents containing term t+1Total number of docume nts |D|) (The +1 avoids division by zero)
    - *TF-IDF Weight*: TFIDF(t,d,D)=TF(t,d)×IDF(t,D)
    - In this project, TF-IDF was implemented using Scikit-learn's `TfidfVectorizer` or a combination of `CountVectorizer` and `TfidfTransformer`. Variations included using word n-grams (e.g., unigrams and bigrams, `ngram_range=(1, 2)`) and character n-grams (e.g., `ngram_range=(3, 5)`, `analyzer='char_wb'`), as well as options like `max_features` to limit vocabulary size and `sublinear_tf` for scaling term frequency.
- **Classification Algorithms:** The TF-IDF vectors were fed into standard classification algorithms:
    - **Multinomial Naive Bayes:** A probabilistic classifier based on Bayes' Theorem, assuming independence between features (words). It's often effective for text classification.

- **Logistic Regression:** A linear model adapted for classification tasks, using a sigmoid (or softmax for multi-class) function. Implemented with One-vs-Rest (OvR) for multi-class classification.
- **Random Forest:** An ensemble method that builds multiple decision trees and aggregates their predictions (e.g., by voting) to improve robustness and accuracy.
- **XGBoost (Extreme Gradient Boosting):** An efficient and powerful implementation of gradient boosted decision trees, known for its performance in competitions.
- **LightGBM (Light Gradient Boosting Machine):** Another high-performance gradient boosting framework, often faster than XGBoost.

b) Transformer-Based Models:

These models leverage deep learning, specifically the transformer architecture, which has revolutionized NLP.

- **Transformer Architecture:** Introduced in the paper "Attention Is All You Need" (Vaswani et al., 2017), the transformer relies heavily on the **self-attention mechanism**. This allows the model to weigh the importance of different words in the input sequence when representing a specific word, capturing contextual relationships effectively. It consists of an encoder (to process the input text) and a decoder (often used for generation tasks, but only the encoder or the encoder plus a classification head is typically used for classification).

  (Placeholder: A figure illustrating the Transformer architecture, typically showing multi-head attention and feed-forward layers within encoder/decoder stacks, would be included here in a full report.)

  The self-attention calculation is often represented as: Attention(Q,K,V)=softmax(dk QKT)V where Q (Query), K (Key), and V (Value) are matrices derived from the input embeddings, and dk is the dimension of the keys.

- **Pre-training and Fine-tuning:** Transformer models like BERT (Bidirectional Encoder Representations from Transformers) are pre-trained on massive amounts of text data using self-supervised objectives (like Masked Language Modeling and Next Sentence Prediction). This allows them to learn rich language representations. For specific downstream tasks like classification, the pre-trained model is **fine-tuned**: a task-specific layer (e.g., a linear layer for classification) is added on top, and the entire model (or parts of it) is trained for a few epochs on the task-specific labeled dataset.
- **Models Used:**
  - **DistilBERT (`distilbert-base-uncased`)**: A distilled (smaller, faster) version of BERT, retaining much of its performance.
  - **DeBERTa (`microsoft/deberta-v3-base`)**: Incorporates architectural improvements like disentangled attention and an enhanced mask decoder, often achieving state-of-the-art results.

- **MathBERT (`tbs17/MathBERT`)**: A BERT model specifically pre-trained on a large corpus of mathematical texts (papers from arXiv, textbooks), making it potentially more suitable for this domain. It uses special tokens like `[MATH]` to handle mathematical expressions.

These models were implemented using the Hugging Face `transformers` library, utilizing `AutoTokenizer` for text tokenization (including handling special tokens and padding/truncation to `MAX_LENGTH`) and `AutoModelForSequenceClassification` which provides the pre-trained model base with a sequence classification head.

4. Experimental Setup

- **Data Splitting:** The labeled data (`train.csv` or `train_pp.csv`) was consistently split into training, validation, and test sets. Common splits observed were 80% train / 20% validation (in baseline scripts) or 80% train / 10% validation / 10% test (in transformer scripts, achieved via two splits). The splits were performed using `train_test_split` from Scikit-learn, often with stratification (`stratify=y`) to ensure that the proportion of classes in each split mirrored the original dataset.
- **Implementation Frameworks:**
  - Classical models were implemented using **Scikit-learn**, leveraging its tools for feature extraction (`TfidfVectorizer`, `CountVectorizer`, `TfidfTransformer`), modeling (`MultinomialNB`, `LogisticRegression`, `RandomForestClassifier`), and evaluation (`classification_report`, `f1_score`). Scikit-learn `Pipeline` objects were used to chain preprocessing and modeling steps.
  - Transformer models were implemented using the **Hugging Face `transformers` library** built on **PyTorch**. Key components included `Dataset` and `DatasetDict` from the `datasets` library for data handling, `AutoTokenizer` for tokenization, `AutoModelForSequenceClassification` for model loading, and the `Trainer` API for managing the training loop, evaluation, and model saving. The `model_utils.py` script provided helper functions for loading, preprocessing, and predicting with MathBERT. A Streamlit application (`app.py`) was built for demonstration.
- **Performance Evaluation:** Model performance was primarily judged based on standard classification metrics calculated on the validation or test sets:
  - **Accuracy:** The overall proportion of correctly classified problems.
  - **F1-Score:** The harmonic mean of precision and recall. Different averaging methods were used:
    - *Macro F1:* Calculates F1 independently for each class and averages them (treating all classes equally).
    - *Weighted F1:* Averages per-class F1 scores, weighted by the number of true instances for each class (accounting for class imbalance).
    - *Micro F1:* Calculates metrics globally by counting total true positives, false negatives, and false positives (equivalent to accuracy in multi-class classification).
  - **Classification Report:** Provides precision, recall, and F1-score for each class individually.

    - Metrics were computed using Scikit-learn functions or the `evaluate` library integrated with the Hugging Face `Trainer`.

5. Hyperparameters and Overfitting Prevention

Several hyperparameters were configured or implicitly tuned during the experiments:

- **General:**
    - `RANDOM_STATE`: Used for reproducibility in data splitting and model initialization.
- **TF-IDF:**
    - `max_features`: Maximum number of features (vocabulary size), e.g., 5000, 7000, 10000.
    - `ngram_range`: Range of n-grams to consider, e.g., (1, 2) for words, (3, 5) for characters.
    - `analyzer`: Feature type ('word' or 'char_wb').
    - `sublinear_tf`: Boolean flag to apply sublinear TF scaling.
- **Classical Models:**
    - `class_weight='balanced'`: Used in Logistic Regression and Random Forest to address class imbalance.
    - `multi_class='ovr'`: Strategy for Logistic Regression.
    - `n_estimators`: Number of trees in Random Forest (e.g., 100) and XGBoost (e.g., 100).
    - XGBoost specific: `max_depth` (e.g., 6), `learning_rate` (e.g., 0.1), `subsample` (e.g., 0.8), `colsample_bytree` (e.g., 0.8).
- **Transformer Models (`TrainingArguments`):**
    - `MAX_LENGTH`: Maximum token sequence length (e.g., 512 for DeBERTa/DistilBERT, 256 for MathBERT).
    - `LEARNING_RATE`: Initial learning rate for the optimizer (e.g., 2e-5).
    - `BATCH_SIZE`: Number of samples per batch during training and evaluation (e.g., 8 for DeBERTa, 64 for DistilBERT).
    - `EPOCHS`: Number of training epochs (e.g., 3 for DistilBERT, 10 for DeBERTa).
    - `WEIGHT_DECAY`: L2 regularization strength (e.g., 0.01).

Overfitting Detection and Prevention:

- **Validation Set:** The primary method for detecting overfitting was monitoring performance on the validation set during training. A significant gap between training performance and validation performance indicates overfitting.
- **Early Stopping:** For transformer models trained with the `Trainer`, the `load_best_model_at_end=True` argument, combined with `eval_strategy="epoch"` and `metric_for_best_model` (e.g., "f1_micro"), effectively implements early stopping. The training process saves the model checkpoint with the best validation score and restores it at the end, preventing the model from continuing to train past the point of optimal validation performance.
- **Regularization:**

    - `WEIGHT_DECAY` (L2 regularization) was applied during transformer fine-tuning to penalize large weights.
    - XGBoost includes inherent regularization parameters like `subsample` (fraction of samples used per tree) and `colsample_bytree` (fraction of features used per tree).
- **Handling Imbalance:** Using `class_weight='balanced'` or `is_unbalance=True` helps prevent the model from simply predicting the majority class, which can be a symptom related to overfitting on biased data.
- **Data Splitting:** The fundamental use of separate train, validation, and test sets ensures that final model performance is evaluated on unseen data, providing a more realistic estimate of generalization ability.

6. Results

(Note: This section describes the experimental outcomes based on the analyses performed in the code. Specific numerical results, tables, and figures comparing models are inferred from the evaluation procedures but were not available in the provided file outputs.)

The project involved training and evaluating a diverse set of models for math problem classification. Key observations from the experimental process include:

- **Baseline Performance:** Classical models like Naive Bayes, Logistic Regression, Random Forest, XGBoost, and LightGBM established baseline performance levels using TF-IDF features. Evaluation reports including per-class precision, recall, F1-score, and overall macro F1 were generated on validation sets for these models.
- **Impact of Preprocessing:** For Logistic Regression and LightGBM, experiments were conducted both with raw text and text processed by the `clean_math_text_final` function. The results indicated that applying this cleaning step improved performance, as measured by accuracy on the validation set. The cleaned data was subsequently used for generating final predictions with the LGBM model.
- **Feature Engineering:** Different TF-IDF strategies were employed. Word n-grams (1-2) were used for Naive Bayes, Random Forest, Logistic Regression, and LightGBM, while XGBoost utilized character n-grams (3-5). While direct comparison figures are absent, this suggests exploration into different feature representations for classical models.
- **Transformer Model Fine-tuning:** Transformer models (DistilBERT, DeBERTa, MathBERT) were successfully fine-tuned on the classification task. The Hugging Face `Trainer` API was configured to log metrics (accuracy, F1-weighted, F1-micro) during training on both the training and validation sets epoch-by-epoch. The best performing model checkpoint based on the validation set metric (`f1_micro`) was saved.
- **Evaluation on Test Set:** After fine-tuning, the best transformer models were evaluated on the held-out test set, and the final test metrics (e.g., `test_results`) were logged. This provides an unbiased estimate of the final model's generalization performance.
- **Data Augmentation Experiment:** The DeBERTa model was trained separately on original and paraphrased data. This allows for a comparison (though numerical results are missing) of how data augmentation impacts model performance on this specific task.

- **Demonstration:** The Streamlit application successfully integrates a trained model (likely MathBERT based on `model_utils.py`) to classify user-provided questions in real-time, displaying the predicted category and confidence scores.

In a complete report, this section would include tables comparing the validation and test set scores (Accuracy, F1-scores) across all major models (Naive Bayes, RF, XGBoost, LGBM, DistilBERT, DeBERTa, DeBERTa-Paraphrased, MathBERT). Figures such as confusion matrices or plots of training/validation loss/metrics over epochs for the transformer models would also be presented to visualize performance and overfitting trends.

7. Summary and Conclusions

This project successfully explored the classification of mathematical problems using a range of NLP techniques. We began by establishing baselines with classical machine learning models (Naive Bayes, Random Forest, XGBoost, Logistic Regression, LightGBM) using TF-IDF features, finding that text preprocessing significantly impacted performance. Subsequently, we implemented and fine-tuned advanced transformer models, including DistilBERT, DeBERTa, and the domain-specific MathBERT, leveraging the Hugging Face ecosystem. An experiment involving data augmentation through paraphrasing was also conducted using the DeBERTa model. Evaluation involved standard metrics like accuracy and F1-score on validation and test sets, with strategies employed to monitor and prevent overfitting.

Key Learnings:

- Both classical and transformer-based models can be effectively applied to math problem classification.
- Text preprocessing is a critical step, and domain-specific considerations (like handling mathematical notation) are important.
- Transformer models, especially those pre-trained on relevant corpora (like MathBERT), offer a powerful approach due to their ability to capture complex contextual information.
- The Hugging Face library provides a robust framework for implementing and evaluating transformer models.
- Data augmentation techniques like paraphrasing are viable exploration paths for potentially improving model robustness.

Future Work: Several directions could be pursued to potentially improve upon the results obtained:

- **Hyperparameter Optimization:** Conduct a more systematic search for optimal hyperparameters (e.g., learning rate, batch size, weight decay, TF-IDF parameters) using techniques like grid search, random search, or Bayesian optimization.
- **Model Exploration:** Experiment with larger or more recent transformer architectures.
- **Ensemble Methods:** Combine predictions from multiple models (e.g., averaging predictions from the best classical model and the best transformer model) to potentially achieve better performance.

- **Advanced Data Augmentation:** Explore other augmentation techniques beyond paraphrasing, such as back-translation or synonym replacement.
- **Error Analysis:** Perform a detailed analysis of the errors made by the best models to understand their weaknesses and identify specific types of problems that are difficult to classify.
- **Handling Mathematical Notation:** Investigate more sophisticated methods for representing mathematical expressions within the models, potentially moving beyond simple placeholders or special tokens.

8. References

Libraries and Frameworks:

- **Scikit-learn:** Pedregosa, F. et al. (2011). Scikit-learn: Machine Learning in Python. JMLR 12, pp. 2825-2830. (Used for classical ML models, TF-IDF, evaluation)
- **Hugging Face Transformers:** Wolf, T. et al. (2020). Transformers: State-of-the-Art Natural Language Processing. EMNLP 2020. (Used for transformer models, tokenizers, Trainer API)
- **PyTorch:** Paszke, A. et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. NeurIPS 2019. (Backend for Hugging Face models)
- **Pandas:** McKinney, W. (2010). Data Structures for Statistical Computing in Python. Proc. 9th Python in Science Conf. (Used for data loading and manipulation)
- **NumPy:** Harris, C.R. et al. (2020). Array computing with NumPy. Nature 585, 357–362. (Used for numerical operations)
- **XGBoost:** Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. KDD '16.
- **LightGBM:** Ke, G. et al. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. NeurIPS 2017.
- **Datasets:** Lhoest, Q. et al. (2021). Datasets: A Community Library for Natural Language Processing. EMNLP 2021. (Used with Hugging Face Trainer)
- **Evaluate:** von Werra, L. et al. (2022). Evaluate: A Library for Evaluating Natural Language Processing Models. (Used for metrics with Hugging Face Trainer)
- **Streamlit:** (Used for creating the demo application `app.py`) - `https://streamlit.io/`
- **gdown:** (Used in `model_utils.py` for downloading model checkpoint from Google Drive)

Models and Concepts:

- **Transformer:** Vaswani, A. et al. (2017). Attention Is All You Need. NeurIPS 2017.
- **BERT:** Devlin, J. et al. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL-HLT 2019.
- **DistilBERT:** Sanh, V. et al. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. ArXiv:1910.01108.
    - Hugging Face Hub: `https://huggingface.co/distilbert-base-uncased`
- **DeBERTa:** He, P. et al. (2021). DeBERTa: Decoding-enhanced BERT with Disentangled Attention. ICLR 2021.

    - Hugging Face Hub: `https://huggingface.co/microsoft/deberta-v3-base`
- **MathBERT:** Peng, Z. et al. (2021). MathBERT: A Pre-Trained Model for Mathematical Formula Understanding. ArXiv:2110.05379.
    - Hugging Face Hub: `https://huggingface.co/tbs17/MathBERT`
- **TF-IDF:** Salton, G. & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. Information Processing & Management.

Dataset Source (Inferred):

- Kaggle Competition: Classification of Math Problems by Kasut Academy (Based on paths like `/kaggle/input/classification-of-math-problems-by-kasut-academy/`).

Code Snippets/Workarounds:

- Streamlit PyTorch Class Loading Workaround: `https://github.com/VikParuchuri/marker/issues/442#issuecomment-2636393925`