LLM Source Classification Model

Project Report

Egor Dolgov, Cedric Jizmejian, Kirill Utyashev

Executive Summary

In this project, we compared softmax regression, random forests, and feed-forward neural networks for predicting which LLM a student used, and selected the best model using group-aware cross-validation with student-level splits. Softmax regression ultimately performed the best, achieving about 71% accuracy and macro-F1 on both validation and the 10% test set. This strong performance comes from the high-dimensional space created by the text and categorical features, where linear decision boundaries tend to generalize more reliably than the more complex models we evaluated.

Data Exploration

Dataset Summary. The dataset includes 825 rows and 11 columns, with 10 input features: 3 categorical (columns 1, 4, 6), 4 numerical (3, 5, 8, 9), and 3 textual (2, 7, 10). Classes are perfectly balanced, each representing one-third of the observations. Among the numerical features, "academic use of the LLM" and "frequency of suboptimal responses" show the most uneven label distributions (Figures 1a–b), while others are more uniform and thus less discriminative (Figure 1c). For categorical features (excluding *student_id*), "math computations" and "drafting professional text" show the lowest entropy across labels, indicating stronger predictive power.

Issues Identified. Missing or corrupted entries like "#NAME?" were removed. Outliers, most notably in "verification frequency" ($\approx$ 12%), were retained since they represent genuine variability rather than noise. Categorical variables showed no outliers. After cleaning, 822 complete rows remained for modeling.

Data Splitting. We randomly partitioned the dataset into a 90% training set and a 10% test set using a fixed seed. We then further split the training set into folds for cross-validation, as explained in the *Methodology* section. To prevent information leakage, we ensured that no data points with the same *student_id* appeared across multiple splits.

Pre-processing. Categorical features (columns 4 and 6) were encoded using indicator variables. Text features (columns 2, 7, 10) were cleaned by removing stop-words and vectorized using a bag-of-words model for each column. We tune the size of the vocabulary as discussed in the next section. Each word's presence was encoded as an indicator variable, preserving interpretability and keeping the feature space compact. Because the resulting features are categorical indicators, empty rows are handled naturally, as they simply correspond to zeros across all dimensions. Lastly, missing values in numerical features were imputed using the median, which is robust to outliers and straightforward to implement.

Key Insights. We derive these insights within training set only. Students selected ChatGPT more often for academic tasks, where it also produced more suboptimal responses, making these features highly predictive of the ChatGPT label. Survey responses about best tasks (Tables 1–3) provide the most informative categorical predictors, with low conditional entropy values indicating strong alignment with assigned labels, particularly for distinguishing ChatGPT and Claude. Text-derived features (Tables 4–6) reinforce these trends: writing and explanation terms are most predictive of ChatGPT, coding-related terms align with Claude, and Gemini exhibits relatively weaker associations.

Label	0	1
ChatGPT	62	185
Claude	181	64
Gemini	160	86

Table 1: Professional text.

Label	0	1
ChatGPT	139	108
Claude	82	163
Gemini	172	74

Table 2: Coding

Label	0	1
ChatGPT	35	212
Claude	162	83
Gemini	123	123

Table 3: Explaining

Label	0	1
ChatGPT	167	80
Claude	233	12
Gemini	228	18

Table 4: Concept

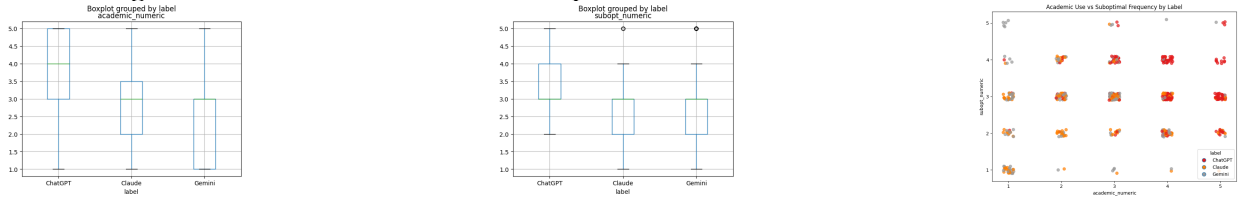
Label	0	1
ChatGPT	238	9
Claude	245	0
Gemini	201	45

Table 5: Google

Label	0	1
ChatGPT	178	69
Claude	118	127
Gemini	198	48

Table 6: Coding (text)

Data Patterns and Model Selection. Across the most predictive numerical features (Figure 1), we observe substantial overlap among classes, indicating that the data is not linearly separable. To capture these patterns, we employ three model families: softmax regression as a linear baseline, random forests for nonlinear feature interactions and variance reduction, and feed-forward neural networks for higher-level nonlinear relationships and feature construction.



(a) Academic use (b) Suboptimal responses (c) Feature overlap

Figure 1: Feature distributions and overlap across labels.

Methodology

Model Families. We analyze three model families: softmax regression, random forests, and feed-forward neural networks.

We begin with softmax regression, which learns linear decision boundaries between the three class labels. Although two of the most predictive features are not linearly separable on their own, separability can improve once the data is represented in a higher-dimensional space through categorical and text indicators. From of this, we expect softmax to serve as a solid linear baseline rather than performing extremely poorly.

We include random forests because they handle categorical features effectively (which make up the majority of our dataset) and tend to produce more stable predictions, especially when the data is noisy, as our exploratory analysis suggests. Ensembling allows the model to mitigate variance and capture nonlinear interactions.

Finally, we select feed-forward neural networks, which can learn complex nonlinear relationships and construct higher-level representations. For example, a neural network may capture intricate interactions between text features and numerical responses that would be difficult or impossible to manually engineer.

We selected these three model families because together they cover the full range of model capacity that would be appropriate for this dataset: a simple linear baseline (softmax), a nonlinear but interpretable ensemble (random forest), and a flexible high-capacity model (neural network), allowing us to evaluate performance across different model strengths.

Optimization. For the softmax regression model, we use the built-in `lbfgs` solver from scikit-learn. We don't have to select a learning rate because this solver efficiently performs full-batch gradient descent with second-order updates. Using full batches is effective and avoids the additional noise associated with SGD or mini-batch updates because the dataset is relatively small. An L2 penalty is used to apply regularization, which we adjust by experimenting with various C values. Additionally, the solver stops when it reaches the iteration limit or converges; neither a learning-rate schedule nor any explicit early-stopping mechanism is used.

We use the `MLPClassifier` with the Adam optimizer for the Neural Network. Adam is more stable than simple SGD because it employs mini-batches and automatically modifies its effective learning rate during training. We use `alpha` to modify L2 regularization. We don't add any extra learning-rate schedule or early-stopping mechanism; instead, we rely on Adam's built-in convergence behaviour and a maximum iteration cap.

For the random forest model, a greedy splitting strategy is used to construct each tree, with each split selected to maximize impurity reduction (testing both Gini and Entropy). Structural limitations such as the maximum depth, minimum samples needed to split or form a leaf, the number of features taken into account at each split, and the limit on leaf nodes are the source of regularization. These factors regulate the trees' potential for complexity.

Validation. We use a seeded, group-aware split with `GroupShuffleSplit` using *student_id* so that each student only shows up in one place. Group-aware splitting treats all responses from the same student as a single unit and never lets those responses get split across train and test, which prevents any leakage. This gives us a 90% training set and a 10% test set. All validation is done inside the training split using group-aware cross-validation instead of making a fixed validation set.

During model selection, we run `GroupKFold`, which builds fixed folds based on *student_id* and keeps all responses from the same student together so they never appear in both the training and validation

folds. Because these folds are fixed and reused across all models and all runs, the comparisons we make between models or hyperparameters are fair and not just a result of different random splits. After picking the best hyperparameters based on the average validation accuracy across folds, we retrain the model on the full 90% training split and then evaluate it once on the 10% held-out test set. As part of this process, we also introduce our own hyperparameter, k, which controls how many text features we include; we tune k using the same group-aware cross-validation so that the number of text features is picked in a consistent and data-driven way.

This approach makes sure that the test set does not affect preprocessing, vocabulary selection, hyperparameter tuning, or model selection, preventing any form of information leakage. Because GroupKFold reuses fixed folds, the evaluation across different models is directly comparable, and so there are no favourable splits that could influence conclusions.

Hyperparameter Tuning. Our hyperparameter tuning strategy involved extensive hyperparameter grid search. Table 7 includes all hyperparameters and their values we considered for each model.

For hyperparameter tuning, we compared three models: softmax regression, a Neural Network, and a random forest. For softmax regression, we tried different penalties, solvers, and a range of C values, along with trying it with and without class balancing. For the random forest, we varied the number of trees, the splitting criterion, the max depth, and the minimum number of samples needed to split. For the Neural Network, we tested different hidden-layer sizes, learning rates, and α values.

Further, we varied the size of the bag-of-words representation for each of the three text columns, using $k \in \{0, 3, 15, 30, 60, 90, 120, 180, 210, 300, 450\}$. We treated this as a standard hyperparameter and included it in our grid search. Next we outline how the hyperparameter was implemented and how information leakage was avoided.

We first applied `CountVectorizer` with `binary=True` to obtain one-hot indicator features for all unique words in each column's vocabulary. To select the most informative words, we trained a `LogisticRegression` model with solver="saga" and max_iter=1000 on the training fold using only these one-hot text features and ranked features by the magnitude of their learned weights. Because all features were on the same scale, larger coefficients correspond to more influential predictors. For each column, we then kept the top k features as the vocabulary and used this restricted vocabulary when vectorizing both the training and validation sets.

Crucially, this feature-selection procedure was performed within each cross-validation fold, ensuring that vocabulary construction depended only on the training fold and did not leak information from the validation fold into the training process. This required us to implement a custom preprocessor in `scikit-learn`.

After finding the best hyperparameters for each model, we retrained each one several times with different random initializations to average out the noise from initialization, and then selected the model with the highest mean validation performance. These choices are reasonable because they cover the ranges where the models typically change behaviour, and going beyond this would require an exhaustive search.

Implementation Details. We implemented all models and preprocessing using `scikit-learn`: `LogisticRegression` for softmax regression, `RandomForestClassifier` for random forest, MLPClassifier for feed-forward neural network. We implemented grid search with `GridSearchCV`.
These implementation choices make sure the library can reproduce the same test each time, allowing the full pipeline to be checked consistently across folds.

Table 7: Hyperparameter Search Space

Softmax Regression	
Penalty	L1, L2
Solver	lbfgs (L2), saga (L1/L2)
C values	0.01, 0.03, 0.1, 0.3, 1, 3, 10
Tolerance (tol)	10^{-3}
Class weight	None, balanced
Fit intercept	True, False

Random Forest	
# trees ($n_{\text{estimators}}$)	50, 100, 150, 200, 300, 500
Criterion	gini, entropy
Max depth	None, 10, 20, 30, 50, 70, 100, 150
Min samples split	2, 8, 32, 128, 512

Neural Network	
Hidden layer sizes	all architectures with 1–3 hidden layers and units $\{16, 32, 64, 128, 256\}$
Learning rate	1×10^{-4}, 3×10^{-4}, 1×10^{-3}, 3×10^{-3}
α (L2 reg.)	0, 1×10^{-7}, 1×10^{-6}, 1×10^{-5}, 1×10^{-4}
Max iterations	10000
Solver	adam
Activation	relu

Text Feature Selection	
Top-k values	0, 3, 15, 30, 60, 90, 120, 180, 210, 300, 450

Performance Evaluation. We used macro F1 score as our main metric. It is a harmonic mean of precision and recall (measure of false positives and false negatives). This metric helps understand how accurately can the model differentiate between different LLM labels and will be a good add to accuracy. This ensures that the model must perform well on ChatGPT, Claude, and Gemini individually, rather than relying on strong performance on only one or two labels.

Results

Model Families Results. Softmax reaches its highest validation accuracy at $k = 210$, random forest at $k = 300$, neural network at $k = 180$.

Softmax Regression

k	Mean	Std	F1	F2	F3	F4	F5
0	0.68	0.04	0.62	0.71	0.67	0.66	0.73
30	0.69	0.03	0.64	0.70	0.68	0.69	0.73
60	0.69	0.03	0.66	0.69	0.67	0.70	0.76
90	0.70	0.03	0.69	0.72	0.67	0.66	0.73
120	0.70	0.02	0.68	0.72	0.69	0.68	0.74
180	0.70	0.03	0.66	0.73	0.71	0.67	0.72
210	**0.71**	**0.02**	**0.68**	**0.72**	**0.72**	**0.68**	**0.73**
300	0.71	0.03	0.68	0.74	0.69	0.68	0.74
450	0.71	0.03	0.68	0.73	0.69	0.68	0.75

Random Forest

k	Mean	Std	F1	F2	F3	F4	F5
0	0.69	0.02	0.69	0.68	0.67	0.71	0.71
30	0.70	0.03	0.70	0.66	0.71	0.74	0.71
60	0.70	0.02	0.70	0.72	0.67	0.72	0.71
90	0.71	0.03	0.70	0.68	0.67	0.76	0.74
120	0.72	0.04	0.72	0.66	0.69	0.76	0.75
180	0.71	0.03	0.71	0.67	0.69	0.76	0.74
210	0.72	0.02	0.73	0.70	0.69	0.73	0.74
300	**0.72**	**0.02**	**0.71**	**0.71**	**0.69**	**0.73**	**0.75**
450	0.71	0.03	0.72	0.68	0.66	0.76	0.73

Neural Network

k	Mean	Std	F1	F2	F3	F4	F5
0	0.66	0.02	0.62	0.64	0.65	0.67	0.69
30	0.65	0.02	0.64	0.69	0.63	0.63	0.67
60	0.64	0.02	0.60	0.66	0.64	0.64	0.65
90	0.64	0.02	0.61	0.64	0.64	0.67	0.66
120	0.66	0.03	0.60	0.69	0.69	0.65	0.65
180	**0.66**	**0.03**	**0.63**	**0.71**	**0.66**	**0.66**	**0.65**
210	0.66	0.02	0.66	0.70	0.65	0.67	0.64
300	0.66	0.02	0.66	0.70	0.63	0.67	0.66
450	0.65	0.02	0.62	0.69	0.65	0.65	0.64

We then selected the best hyperparameters for each model based on the highest performance, shown in Table 8. Next, we re-evaluated them using five cross-validation re-runs with different initialization seeds (Table 9). Softmax Regression produced the same results as the setup we used was deterministic. Random Forest had a larger standard deviation than Neural Network, yet it achieved a higher mean accuracy. Softmax has the highest mean CV accuracy and macro F1. Thus, Softmax Regressions is the model we chose for the final test set.

Softmax (k=210)	RF (k=300)	NN (k=180)
L2 penalty	Trees: 300	Layers: (256)
Solver: lbfgs	Criterion: gini	$\alpha = 10^{-5}$
C = 0.1	Max depth: 30	LR = 10^{-4}
Class wt: None	Min split: 2	Activation: relu
Tol = 10^{-3}		Solver: adam
Intercept: Yes		Iters: 10000

Table 8: Best hyperparameters

Model	Acc	Std	Macro F1	Std
Softmax (k=210)	0.707	0.000	0.704	0.000
Random Forest (k=300)	0.699	0.009	0.695	0.009
Neural Net (k=180)	0.643	0.007	0.641	0.006

Table 9: Performance over 5 random seeds

Error Analysis. To analyze the model's errors, we constructed a confusion matrix. The model identifies ChatGPT most accurately, but it struggles the most with Claude, followed closely by Gemini. This occurs because the categorical and textual indicators for Claude and Gemini have higher conditional entropy, meaning the predictive signal is weaker. As a result, the model finds it more difficult to distinguish between these two classes. In contrast, ChatGPT exhibits more distinctive patterns, making it easier to classify. Additionally, entries labeled as Claude and Gemini had disproportionately more empty fields, weakening the signal and leading to more misclassifications between them in the confusion matrix.

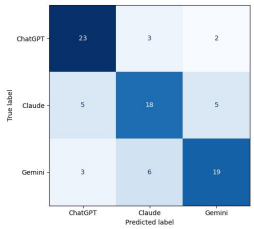


Figure 2: Confusion matrix for the final Softmax Regression model.

Test Set Performance. We trained Softmax with $k = 210$ on the 90% training split and evaluated its performance on the 10% test set. **The test accuracy is 0.7143** (our single-number estimate), and for our second metric, the test macro F1 score is 0.7126. Both values are very close to our cross-validation results, indicating that the estimates are stable and the model generalizes well. Macro-F1 averages performance equally across all three labels, so the close match between test and validation macro-F1 shows that the model is not overfitting to any single class and maintains balanced performance across ChatGPT, Claude, and Gemini.

References

[1] Scikit learn. Tfidfvectorizer, 2018.

[2] Artem Ryblov. Text classification: Baseline with tf-idf and logistic regression, 12 2022.

[3] scikit learn. 3.1. cross-validation: Evaluating estimator performance — scikit-learn 0.21.3 documentation, 2009.

[4] scikit learn. Logisticregression, 2014.

[5] scikit learn. 1.17. neural network models (supervised) — scikit-learn 0.23.1 documentation, 2025.