CSC311: Introduction to Machine Learning

Project Report

Submitted by

Benjamin Gavriely (1009970015), Christopher Marrella (1008277152),
Wilton Miller (1009976171)

University of Toronto
Fall 2025

1 Executive Summary

The model families we explored were logistic regression, a decision tree classifier and a multilayer perceptron (MLP). Logistic regression emerged as the strongest performer and achieved an accuracy of 73%. This was likely due to the structure of the data. The open-ended responses produced high-dimensional TF-IDF vectors. TF-IDF creates a numeric indicator of how relevant that term is to a document. This could to lead to linear separability between classes on the decision boundary, which logistic regression models efficiently and reliably. The model's L2 regularization also helps smooth out noise from less common words and typos, which are common in open-ended text responses.

2 Data Exploration

2.1 Data summary

The data has three main feature types: Text, numerical on a 1-5 scale, and multi-select.

Text: There are three open-ended text features, each a short response.

Numerical: There are 4 numerical features on a scale 1 to 5. The distribution is balanced, with roughly a normal distribution centered at 3. These features are most likely going to be very predictive for our models, since there are clear correlations between labels.

Multi-select: There are two "select all that apply" responses, with 8 possible options to choose each. These are also balanced, with the most common selections as "Explaining complex concepts simply" and "math computations" respectively.

2.2 Issues found

Some of the open ended response cells are left blank, or contain placeholder strings such as #NAME? (there are 4 instances of such). The majority of the blank cells in the responses appear to arise because a specific student has given the same answer across all three models, the first cell will contain the answer itself with the other two left blank to signify that it is the same. This pattern is also used in the Supporting Evidence column, resulting in a high number of NaN responses. Another problem was with the text field containing typos, which influence any text processing that can be done and decrease the efficacy of these techniques.

2.3 Preprocessing

The first step was to rename all of the column names to be lowercase and use underscores instead of spaces to make this data more usable in future code. In this step, we also replaced all instances of "#NAME?" with an empty string, and replaced all null values with the empty string.

Then, we processed all numerical features by first filling all null values with the mode of that respective column, which was done to preserve distribution of the values. We stripped the column of everything except for the integer value, and performed a linear scaling to fit the values into the range [0,1] to avoid these features dominating the model.

After this, the categorical features were each encoded into a one-hot vector, where each possible selection became an indicator feature.

The most complex transformation was done for the text features, where we used TF-IDF to vectorize the responses. To do this, we first replaced all placeholders of [THIS MODEL] and [ANOTHER MODEL] with a lowercase text equivalent without brackets. Then, we converted all characters to lowercase, stripped punctuation, and removed any numbers. This was all done to create usable tokens for TF-IDF. After this, we removed stop words using scikit-learn, and lemmatized all text using the spacy library. These are both done in an effort to improve the accuracy of the TF IDF encoding. We used scikit-learn to encode a value for each data point for each word in the column, making a minimum threshold of 3 for the frequency of any word included in this process to eliminate the presence of typos or rare words.

Lastly, we converted the labels to be a one-hot encoding of the three models for classification.

We also saved all fitted preprocessing objects (TF–IDF vectorizers, categorical encoders, and feature order) so that `pred.py` could reproduce the exact feature space required by the trained models.

2.4 Data splitting

We split our data before preprocessing, using above the 80th percentile of student ids for tests and below 80 for training. Because this split was done by student id, it means that all of one student's data can only be in one of the splits. We did not randomize the data because we reasoned that there is no leakage that occurs from keeping the data in its original order.

For preprocessing, the only steps that required care were filling N/A values for the numerical columns and TF-IDF vectorizing. For numerical columns, we took the mode of only the training set and propagated that mode to the test set. Similarly, we fit our TF-IDF vectorizer to the training data and only applied it to the training and test data. This ensured no data leakage and that the test set was not used during any exploration or processing.

3 Methodology

3.1 Model Families

We evaluated three model families chosen to reflect different tradeoffs between linearity, interpretability, and the ability to model non-linear structure.

Multiclass Logistic Regression
Logistic Regression is a strong baseline, especially in our case when we have chosen to use high-dimensional TF-IDF features that dominate the feature space. Because the dataset contains three open-response text fields, the TF-IDF representations creates a sparse, approximately linear structure (see Figure 1) that this model preforms well on using multinational softmax. Since we used L2 regularization, it helped reduce the models sensitivity to noise, and reduced overfitting given $\approx$ 5,000 features.

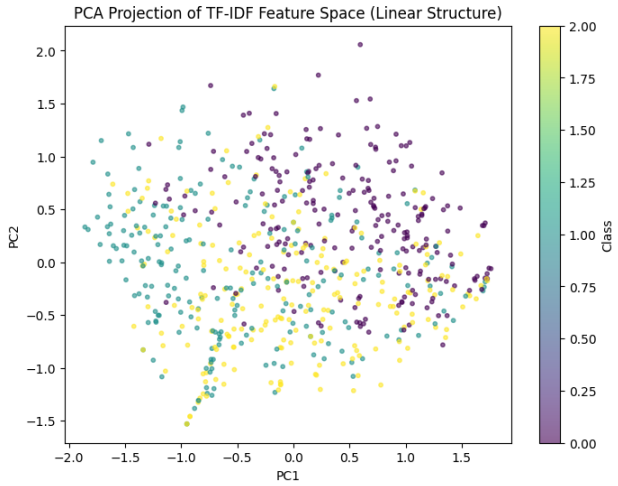

Figure 1: PCA Projection of Feature Space

Decision Tree Classifier
Decision Trees were included because they provide interpretability, handle mixed feature types well, and can capture non-linear interactions between the numeric and categorical fields. While trees struggle with sparse text features, they still provide a useful point of comparison to other models and highlight whether non-linear structure improves accuracy.

MLP
We trained a fully connected neural network with ReLU activations, and a softmax classifier. MLPs can model complex interactions between the TF-IDF features, numerical fields, and the one-hot categorical fields. However, because our feature space is extremely sparse and largely linear in structure, the added nonlinearity of the MLP did not provide an advantage. So in training, this led the neural network to perform worse than our simpler multinomial logistic regression model.

3.2 Optimization

Each model family used a specific set of optimizers.

Logistic Regression
We used LBFGS optimizer, which performs well for multinomial classification and con-

verges reliably within a moderate number of iterations. The regularization strength was controlled by the parameter C, with larger values reducing the magnitude of the L2 penalty, and vice versa.

Decision Tree
The tree was optimized using Gini impurity to select the splits. Hyperparameters such as maximum depth, min samples split, and min samples leaf controlled the model complexity and prevented overfitting.

MLP
Our MLP was trained using the Adam optimizer, which adaptively adjusts learning rates and is well-suited for high-dimensional sparse data. We used ReLU activations in the hidden layers and a softmax output. L2 weight decay was applied through the `alpha` parameter to regularize the network.

No early stopping or learning-rate schedules were used, as the models converged consistently within the training limits.

3.3 Validation Method

Because our dataset was medium in size, and the class distribution balanced, we used **10-fold StratifiedKFold cross-validation** on the training split only (we did an 80/20 split). Stratification preserved the proportions of class labels in every fold.

This choice reduces variance in the model estimates, and prevents our hyperparameter tuning from overfitting to a single validation split. The test set was held out strictly for final evaluation and was never used during hyperparameter tuning. preprocessing, or model selection.

Each hyperparameter configuration was trained independently on 9 folds and validated on the remaining fold, with the mean Macro-F1 used as the selection criterion.

3.4 Hyperparameter Tuning and Ranges

The following grids were explored during cross-validation:

Logistic Regression Grid

- $C \in \{0.01, 0.1, 1, 10, 100\}$

Decision Tree Grid

- `max_depth` $\in \{5, 10, 15, \text{None}\}$
- `min_samples_split` $\in \{2, 5, 10\}$
- `min_samples_leaf` $\in \{1, 2, 5\}$
- `max_features` $\in \{\text{"sqrt"}, \text{"log2"}\}$

Neural Network Grid

- `hidden_layer_sizes` $\in \{(64,), (128,), (128, 64)\}$
- `alpha` $\in \{10^{-5}, 10^{-4}, 10^{-3}\}$
- `batch_size` $= 64$
- `max_iter` $= 300$

All combinations were evaluated for each model family to avoid "winner-only tuning". This resulted in a fair comparison between the linear, tree-based, and neural models.

3.5 Evaluation Metrics

We evaluated models using two primary metrics:

1. **Accuracy:** which measures the overall proportion of correct predictions across all classes.
2. **Macro-averaged F1 score:** which was our main comparison metric. This metric computes the F1 score independently for each class and then averages them, assigning equal weight to ChatGPT, Claude, and Gemini. This is appropriate for our dataset, where response similarities between Claude and Gemini may cause unequal model performance across classes. Macro-F1 also penalizes models that perform well only on the majority classes, to ensure balanced evaluation.

In addition to these quantitative metrics, we analyze model behavior through confusion matrices in the Results section.

3.6 Implementation Details

All preprocessing steps, including TF-IDF vectorization of text responses, one-hot encoding of the categorical fields, and normalization of numerical features, were implemented using `pandas`, `numpy`, and `scikit-learn`.

The full training pipeline, which includes hyperparameter search, 10-fold cross-validation, and final model evaluation, was implemented in `src/train/train.py`. Cross-validation was performed using `StratifiedKFold` to preserve class balance across folds. Model families were trained using `scikit-learn`'s `LogisticRegression`, `DecisionTreeClassifier`, and `MLPClassifier`, with hyperparameter configurations determined by the grid search.

After identifying the best hyperparameters for each model family and selecting the best overall model, we saved the relevant artifacts to the `models/` directory:

- `best_model_type.txt` indicating the top-performing model family,
- `best_model_params.json` for Logistic Regression or MLP, containing extracted weights and biases,
- `best_model.pkl` for Decision Trees (when applicable),
- `metrics.json` containing test accuracy, macro-F1, confusion matrix, and all per-model scores.

These saved files are then used to perform inference in `pred.py` without relying on scikit-learn.

4 Results

We assessed each model using macro F1 score, accuracy, confusion matrices, and ROC curves. Logistic regression achieved the strongest overall performance with a macro F1 of 0.731, followed by MLP at 0.650 and decision tree at 0.470. The consistent ranking across validation runs and the test plot indicated that logistic regression is the most stable and generalizable model among the three.

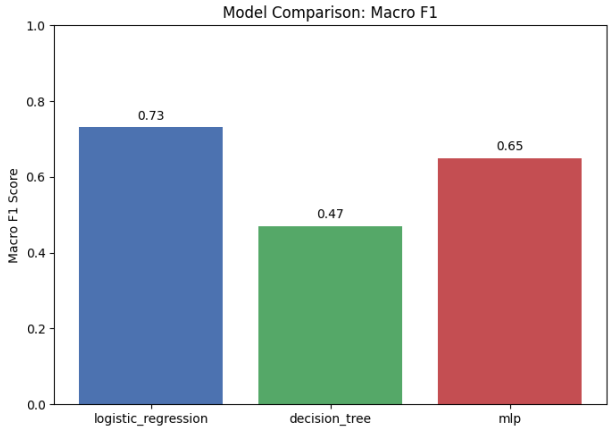

Figure 2: Bar graph comparing F1 scores of each model.

The model performed well on all three classes, correctly identifying 47 correct ChatGPT responses, 38 Claude responses, and 36 Gemini responses out of our 165-response test set. Most errors occurred between Claude and Gemini, where some responses share similar patterns in their response.

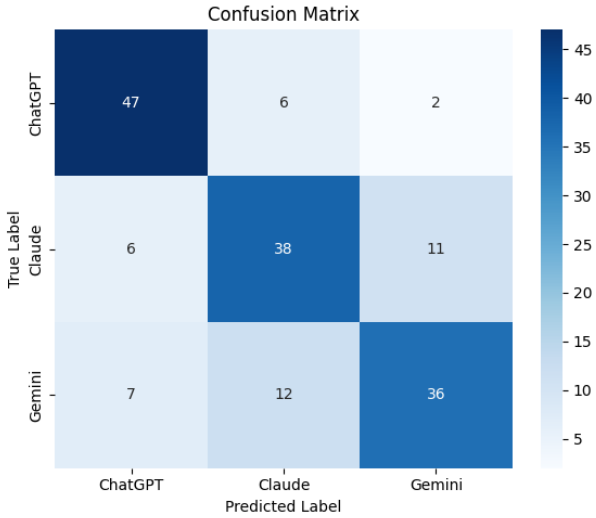

Figure 3: Confusion matrix for logistic regression.

To further analyze model discrimination, we plotted the one-vs-rest ROC curve for all three classes. Each curve rises above the chance line, and the macro ROC AUC is approximately 0.78. This confirms that the model meaningfully separates the classes.

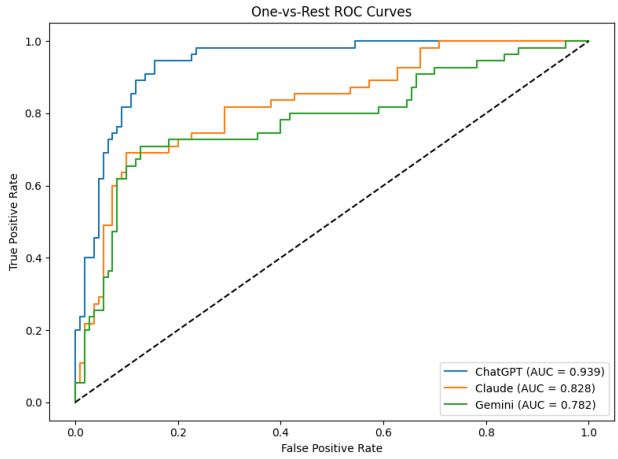

Figure 4: ROC curve for all three classes.

ChatGPT has the strongest performance, with a precision of 0.783 and a recall of 0.855. This indicates that the model not only correctly identifies most ChatGPT responses (high recall) but also rarely mislabels other classes (high precision). Claude showed a weaker performance, with a precision 0.679 and a recall 0.691. Gemini displays an inverse pattern. Its precision is a strong 0.735, but its recall is lower 0.655.

5 Contributions and Learning

- Christopher Marrella's contribution: I worked on the final Pred.py function, as well as the reconstruction of the TF-IDF vector without the use of machine learning libraries. Furthermore, I also worked the executive summary and results section of the report. The most important lesson I learned was theres a lot more to building a good model than just feature selection and data. A small inconsistency in cleaning or the vectorization can completely halt your progress or change a prediction.
- Benjamin's contribution: I focused on data processing and feature extraction. I split the data into training and test sets, and wrote scripts to pre-process based on the inconsistencies found while exploring the data and extract features. The biggest thing that I learned was how important it is to put time and care into data processing, as there were multiple times where I followed our plan but still noticed small but meaningful problems that arose from cleaning. Because data quality can make or break a model's performance, it was important to evaluate it after each step and adjust the processing accordingly.
- Wilton's contribution: I implemented all model training and selection code, including the Logistic Regression, Decision Tree, and MLP files, as well as the full `train.py` pipeline for cross-validation, hyperparameter tuning, and final model evaluation. I also saved all model parameters and preprocessing objects required for `pred.py`. As well as set up the GitHub repository structure, and wrote the Methodology section of the report. The most important lesson I learned is that effectively breaking up a project (both in what group members are going to do, and creating a clear project structure for you to do your work in) can make things much easier.