LLM MODELS PERFORMANCE COMPARISON ANALYSIS

Akshara Sri Lakshmipathy
Data Science
University Of Colorado
Boulder,CO,USA
akla8196@colorado.edu

Harish Nandhan Shanmugam
Data Science
University Of Colorado
Boulder,CO,USA
hash1366@colorado.edu

Shivaraj Senthil Rajan
Data Science
University Of Colorado
Boulder,CO,USA
shse1502@colorado.edu

ABSTRACT

The project proposes a multi-class classification framework that predicts the most effective Large Language Model by analyzing task characteristics, including domain knowledge, creativity, problem-solving complexity, and code-related tasks. The contribution of this study is mainly to construct a model selection system that can suggest a proper language model for a given task and hence improve the effectiveness and efficiency of the task. We perform an ensemble learning strategy using a Voting Classifier, which integrates three effective models: XGBoost, LightGBM, and CatBoost, to optimize prediction accuracy. The model's performance was evaluated on accuracy, precision, recall, F1-score, and AUC-ROC. Besides, different strategies to handle class imbalance were implemented, such as oversampling and undersampling, to ensure stable performance for all classes. The results show that this Voting Classifier, using gentle voting, far outperformed the single models with its overall accuracy of 69%. This research provides information on developing a large language model recommendation system for appropriate selection and has shown ensemble learning to be effective for solving complex classification problems.

KEYWORDS

Large Language Model - Multi-Class Classification - Ensemble Learning - Voting Classifier - XGBoost - LightGBM - CatBoost - Class Imbalance

1 Introduction

Large Language Models or LLMs have rapidly transformed in natural language processing. They can be applied in tasks such as chatbots and code writing. However, selecting the appropriate LLM for a given task remains a challenge due to the large number of existing models and the specific requirements of each task. This project looks into this need by suggesting a way that the different LLMs can be classified to perform specific tasks. Topic knowledge, creativity, problem difficulty, and code-related tasks are some of the factors that influence how well different LLMs perform and are considered for improving the performance of how well and fast the task is done. Thus, there is a need to design which recommends the optimal LLM based on the task characteristics.

2 Related Work

Recent Large Language Models (LLMs) developments have motivated research in developing, fine-tuning, and a variety of applications. In Huang et al.[1], it is demonstrated that LLMs can self-improve by self-supervised learning without any additional human intervention, which enhances their performance. Alberts et al. [2] discuss the impact of LLMs on nuclear medicine, such as ChatGPT, and the potential consequences for clinical practice and research. In the recommendation systems area, Bao et al.[3] proposed TallRec, a tuning framework that aligns LLMs with recommendation tasks to improve both efficiency and effectiveness. Zhang et al. [4] explored the fairness of ChatGPT in recommendations regarding user demographics and content diversity. Ding et al.[5] focused on parameter-efficient fine-tuning approaches for large-scale pre-trained language models with the goal of saving computational resources while maintaining the performance. Chen et al.[6] explored meta-learning by way of in-context tuning where LLMs are capable of learning new tasks out of the few-shot examples in the input context.

Further studies have expanded the scope of LLM applications in diverse domains. Zheng et al.[7] worked efficiently in fine-tuning of multiple language models for better accessibility and performance. Luo et al.[8] introduced ChatKBQA, a framework for knowledge base question answering by combining generation and retrieval using fine-tuned LLMs. Dunn et al.[9] applied fine-tuned LLMs to structured information extraction from complex scientific texts in order to assist in knowledge discovery. Zhang et al.[10] introduce Video-LLAMA, an instruction-tuned audio-visual language model for video understanding, which integrates information from both audio and visual inputs for better understanding content. Singhal et al.[11] inquired into how large language models encode clinical knowledge and the possible applications of these structures in healthcare settings. Alternatively, Wang et al.[12] wrote a comprehensive review of pretrained language models and their application in different tasks falling under natural language processing. Other papers include Wu et al.[13], which introduces Next-GPT, a multi-modal LLM with the ability in any-to-any modality transformations; Zhuang et al.[14], who present ToolQA, a dataset for the question answering of LLMs requiring external tools, and Lyu et al.[15], who investigates personalized recommendation through prompting LLMs. Also, Li et al. [16] proposed prompt distillation for efficient LLM-based recommendation, Jin et al. [17] adapted LLMs for time series forecasting, Gao et al. [19] optimized attention mechanisms in LLMs for faster computation, and de Zarzà et al. [20] used LLMs in multimodal traffic accident forecasting to improve road safety.

3 Main Methods:

3.1 Data Exploration

Data source: Chatbot Arena Leaderboard - a Hugging Face Space by lmarena-ai.
DataAPI:
https://storage.googleapis.com/arena_external_data/public/clean_battle_20240814_public.json .
In this analysis, we've downloaded the most recent battle data from an external source using the requests library. First, it downloaded raw JSON format data and saved it in a local file called local_file_name.json. After the downloading of data completion, it loads the same into a pandas DataFrame to manipulate and analyze the data further. Data sorting was based on a timestamp, tstamp in an ascending order of time. This sorting allowed us to keep the right chronological sequence of records. After this, the DataFrame was then sent to a CSV format entitled battles_data.csv which now acts as our main data file for analysis. At last, a verification step confirmed that the data had indeed been stored correctly and thus was ready for further processing.

3.2 Data Cleaning and Preprocessing

First, we had the need to transform Unix timestamp seconds into a date format. In their raw form, the dataset showed the Unix timestamps in second form-for example, 1609459200 translated as 2021-01-01 in YYYY-MM-DD. We called pandas' pd.to_datetime() on the column 'tstamp' to transform the columns into proper datetime format. Other than that, the 'month_year' column was created by extracting the month and year from the timestamp as a period (e.g., '2024-08'). We filtered the unique dates from the column 'tstamp' and converted them into a list, to exclude certain dates such as August 1-3, 2024. Further, we excluded these filtered dates from the DataFrame so that our analysis would fall within August 4 to 14, 2024. Also, columns like judge and anony were not required for analysis; hence, we removed them to clean the dataset, thereby getting filtered_excluded_df.

A new column 'winner_model' was then created that identifies the winning model as specified in the 'winner' column, either as model_a or model_b. Now, to reduce the dataset further, the columns model_a, model_b, and winner were removed after the creation of the 'winner_model' column. To extract and enrich the dataset, a function called extract_conv_metadata was defined to parse the 'conv_metadata' column, which contained crucial information about user tokens, assistant tokens, and context tokens. This function was applied to each row of the DataFrame, and the extracted metadata was split into new columns. Next, the original column 'conv_metadata' was dropped, and it filtered the dataset for only those rows whose language was not marked as "unknown" so that only valid data concerning the language was considered. Finally, preprocessing involved extracting unique models used that were derived from the 'model' column and presented as a list for a better understanding of the different language models at work.

In the last stages of preprocessing, further column transformation and refinement were done. We transformed the 'tstamp' column into a period with a frequency of day and placed this in a new column named 'tstamp_period', such that this column would result in a more workable format for the temporal analysis. The column 'category_tag', initially stored as a string in JSON-like format, was parsed into a dictionary using the ast.literal_eval() function, enabling us to extract specific values such as 'information_fulfillment', 'Math', and 'specificity' into new columns. This transformation increased the granularity of the data, hence providing more actionable features for analysis. Finally, the columns 'category_tag', 'tstamp', 'dedup_tag', and 'month_year' were dropped off from the dataset. The cleaned DataFrame, now labeled df_cleaned, was then saved to a CSV file entitled "battles_data_cleaned.csv" for subsequent uses. Saving without the index would also ensure that the final dataset was clean and ready for any further analysis without additional metadata or index clutter.

3.3 Exploratory Data Analysis

In this work, we will analyze model distribution and different Boolean features of the dataset to be used for extracting the patterns. The distribution of models could be visualized in the form of a bar plot, which expresses the number of each model. Indeed, by this plot, it showed that some of the models were highly used, while some were little used, hence giving us an idea of their selection based on their usage rate. Besides that, we considered a number of Boolean features: is_code, is_refusal, and many others. For example, is_code mostly had instances marked as False; it means that most of them were not about technical content. At the same time, the is_refusal feature is mostly False, which means that most inputs were accepted. Information features such as information_fulfillment, math, and problem_solving had higher frequency in False, indicating areas where the models were less proficient to meet information needs, handle mathematical reasoning, and provide approaches to problem-solving. Whereas features like specificity, domain_knowledge, technical_accuracy, and real_world had higher counts for True, showing that models were more specific, domain-based, accurate, and relevant to the real world.

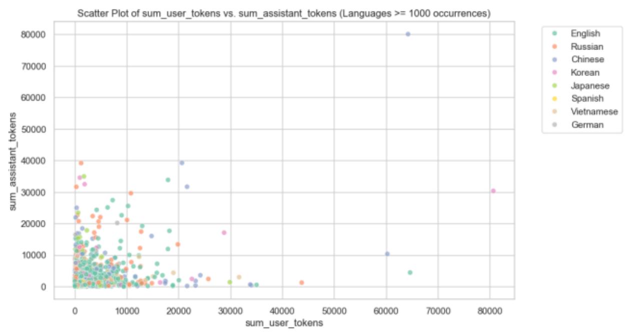

Figure 1: Scatter Plot of sum_user_tokens VS sum_assistant_tokens

We also analyzed the distribution of tokens used by users, assistants, and context. In general, the Kernel Density Estimates for user tokens, assistant tokens, and context tokens all shared the same trend: most values lay around zero, with some extending toward higher values. These demonstrated that most interactions involve relatively low-token usage, but there is a subset showing much higher usages. Finally, a pie chart of the proportion of languages with over 1000 occurrences showed that close to 70% was represented by the English language, while the next most frequent languages are Russian and Chinese. Such a distribution reflects the importance of optimization and selection of appropriate models depending on the languages in multilingual tasks.

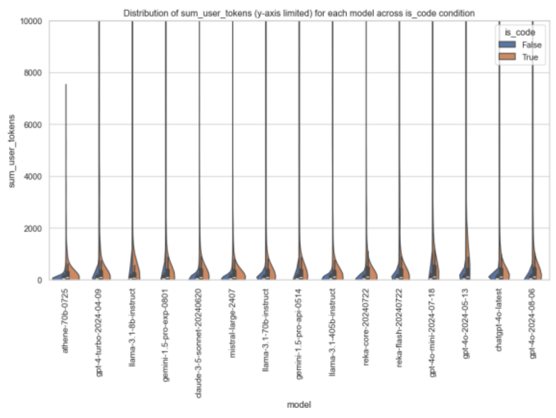

Figure 2: Distribution of sum_user_tokens(y-axis limited) for each model across is_code condition

This study proposes a variety of visualizations in the exploration and understanding of token distribution and language patterns across different models. Box plots of user and assistant tokens for models with more than 2000 occurrences underlined the dispersion of token usage, showing for each model the consistency and potential outliers. These are very important in model development and optimization, emphasizing the efficiency of the tokens. Likewise, violin plots for user and assistant tokens, segmented according to whether the input involved code, shed light on how models performed on token-heavy tasks involving and not involving code; these insights aided model selection when token-intensive tasks were considered.

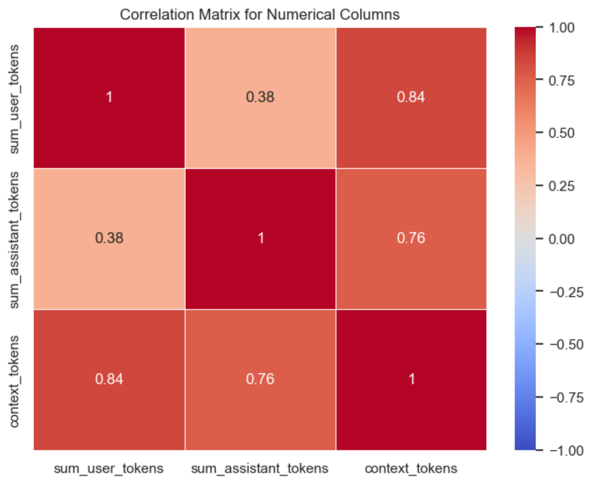

Figure 3: Correlation Matrix for Numerical columns

The correlation matrix heatmap showed a strong positive correlation between context tokens and user tokens, 0.84, with a moderate correlation with assistant tokens, highlighting how these token types interact across models. This will be useful in tuning models for optimization in handling tokens. Scatter plots comparing user and assistant tokens across models and languages further elucidated the pattern in token usage that helped identify efficient models and languages for specific tasks.

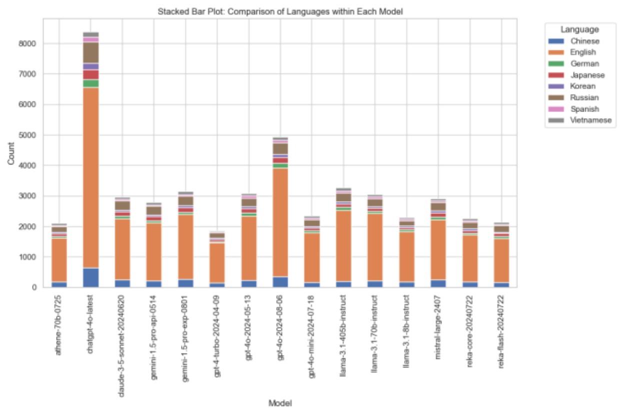

Figure 4: Stacked Bar Plot: Comparison of Languages within Each Model

The distribution of languages was visualized using a stacked bar plot, which showed the prevalence of languages across models. Dominating the majority of the interactions was English. Lastly, a count plot of model refusals helped to identify models with a higher refusal rate, thus guiding model selection for consistent performance. The word cloud visualization summarized the most common languages to aid in multilingual model evaluation. These analyses form the backbone of model optimization and development, ensuring efficiency at token usage and language handling.

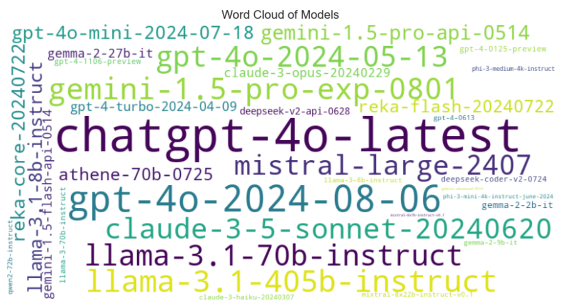

Figure 5: Word Cloud of Models

In this project, several visualizations were used to explore the model behavior and pattern of token usage. Model distribution visualization showed the frequency of different models, which showed that chatgpt-4.0-latest and gpt-4.0-2024-08-06 are very frequent, while others like gemini-1.5-pro-api-0514 are less frequent. Histograms with kernel density estimates show the distribution of user and assistant tokens across languages with over 1000 occurrences, where most interactions involved a low number of tokens with several outliers at higher counts. Bubble plots were used to visualize the relationship between token usage and models; the size of the bubbles indicated the volume of tokens used and helped in determining which models were performing well on token-heavy tasks. Joint plots were used further to delve deeper into how user and assistant tokens correlate to shed light on the efficiency of models in handling tokens. These visualizations are very important for the optimization of model selection and development, especially in token-intensive tasks.

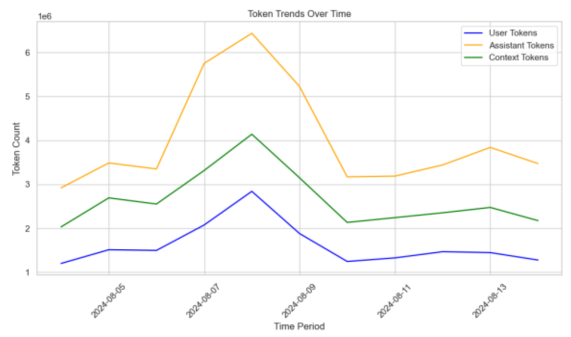

Figure 6: Token Trends Over Time

3.4 Handling Class Imbalance & Feature Selection

In this work, we tried to deal with class imbalance issues of this dataset, especially the 'model' target column. First of all, we examined model distribution by the help of value_counts() function that some models are far outnumbered compared to the other ones. In order to focus our analysis on models that were used frequently in the LLM landscape, we filtered out those that occurred less than 1000 times and kept only the models with more than 1000 occurrences. This step helped streamline the analysis and ensured that the dataset was representative of the models used in most real-world scenarios.

Balancing classes with resampling techniques came next. The nature of this dataset creates a need either to oversample the minority classes or undersample the majority classes for each model to have an equal number of instances. Precisely, we decided on 4000 samples per model, whether by random sampling of the data or replication for models that occurred less than 4000 times. In models that were over 4000 entries, we used undersampling to select a random 4000 samples to keep the balance across the dataset. This evaded the bias towards the model that appears frequently and gave a good chance of comparing models on an equal basis.

Finally, after resampling, we checked the value counts of the 'model' column again to verify the success of the balancing process. Indeed, all models now had 4000 samples, confirming the consistency of the class distribution. This balanced dataset is then saved to a new CSV that can be used for further analysis and model training. The multi-class classification task would, therefore, have a much fairer basis on which to evaluate the model performances.

3.5 Models Implemented

3.5.1 Random Forest Classifier

The random forest algorithm was then applied that can predict the best language model that would fit any particular task. Random Forest represents ensemble learning, which combines multiple decision trees to improve predictive accuracy with better control over overfitting. Each tree in the forest is trained on a random subset of the data and conducts the final prediction based on the majority vote across all trees. It has worked especially well in situations with complex datasets that have multiple features and categories, hence making it appropriate for our problem of multi-class classification.

We used GridSearchCV for hyperparameter tuning to enhance the performance of the Random Forest model. This involved tuning of hyperparameters for the best performance, which included, but was not limited to, number of estimators- n_estimators, maximum depth of tree-max_depth, minimum samples to split an internal node-min_samples_split, minimum samples to be at a leaf node-min_samples_leaf, and boostrap. The GridSearchCV method conducts an exhaustive search over the specified parameter grid and selects the best hyperparameter combination in terms of the best performance through cross-validation.

The optimized model of Random Forest was therefore trained on the dataset using the best hyperparameters identified, to make predictions for each class - a process which gave very vital insights into model performance on various tasks.

3.5.2 XGBoost

XGBoost (Extreme Gradient Boosting) was used in the implementation of the algorithm to predict the best-fitted language model for any given task. XGBoost is an efficient and scalable implementation of gradient boosting, which is a specific ensemble learning technique that creates a series of decision trees. It works by progressively adding trees to correct the mistakes made by previous trees; thus, it's usually effective in complex classification tasks and datasets. First off, XGBoost seemed popular because it can cope well with large datasets that can have missing values as well as robustness regarding the overfitting issues.

Hyperparameter tuning, regarding the best performance for XGBoost, will be done using RandomizedSearchCV. kyperparameters like n_estimators representing the number of boosting rounds, learning_rate relating to step size, max_depth referring to maximum depth of trees, and min_child_weight relating to the minimum sum of instance weight needed in a child, which were tuned for optimal modeling. Other parameters tested during these experiments included gamma as a regularization term, subsample as a fraction of data with which each tree trains, and colsample_bytree as a fraction of features to use in each tree. The RandomizedSearchCV method conducts a search over a random subset of the parameter space, which allows faster convergence and better optimization compared to exhaustive grid search. After finding the best parameters, the model trained with the optimal settings was used to make predictions over the test set.

3.5.3 CatBoost

We implemented the algorithm of CatBoost-an efficient gradient boosting adapted for categorical features. Thereafter, CatBoost is tailored for categorical features and enables working with categorical data without extensive preprocessing that is usually required by other machine learning algorithms, such as one-hot encoding. Basically, this algorithm builds sequential trees in such a way that each is correcting the errors of the previous one. The process is iteratively repeated with a focus on reducing the prediction error; thus, it performs extremely well on both regression and classification problems.

For tuning the optimum performance of the CatBoost model, hyperparameter tuning was done using RandomizedSearchCV. This enables the random search to take place over the hyperparameter space, hence a much faster way of doing the traditional grid search. The tuned hyperparameters include the following: iterations, which stands for the number of boosting rounds; learning_rate, the step size; depth, maximum depth of the trees; l2_leaf_reg, L2 regularization for leaf values; and bagging_temperature, a parameter controlling randomness of the algorithm. Besides, the problem of class imbalance was addressed by setting auto_class_weights to 'Balanced' to avoid over-favoring more frequent classes by the model. After selecting the best combination of hyperparameters, the optimized model was fitted and predictions were made on each class to afford an overall performance evaluation of the model.

3.5.4 LGBM Classifier

The LGBM classifier was implemented-a gradient boosting framework designed for efficiency and scalability. LightGBM is particularly known for its speed and effectiveness when handling large datasets with high-dimensional features. It works by building decision trees sequentially, where each tree corrects the errors made by the previous one. Key features of LightGBM are the ability to handle categorical features directly, parallel learning, and GPU learning. Its memory consumption is also lower than other gradient boosting models.

RandomizedSearchCV is a method used to perform hyperparameter tuning, where random samples from the hyperparameter tuning space are selected. The tuned hyperparameters included num_leaves, the number of leaves in one tree; max_depth, the maximum depth of the trees; learning_rate, the step size to minimize the loss function; and n_estimators, the number of boosting iterations. Other important parameters explored were reg_alpha and reg_lambda, which correspond to L1 and L2 regularization, respectively, and subsample, the fraction of data to be used for training each tree. The RandomizedSearchCV has been set to perform 150 iterations, including cross-validation to evaluate the best-performing hyperparameters. As seen, the best parameters are shown, and these are the ones on which the model should be trained-the best settings for the model concerning the dataset at hand.

3.5.5 Voting Classifier:

In order to combine the predictions of the three powerful machine learning models, namely LightGBM (LGBM), CatBoost, and XGBoost, a Voting Classifier was implemented. Voting Classifier is a form of ensemble learning where multiple models, often called classifiers, make the final prediction through aggregation. The concept followed here is soft voting where the class probabilities for every model are averaged, and a class with a maximum average probability is taken as a final prediction. The base models comprising the ensemble were LGBMClassifier, CatBoostClassifier, and XGBClassifier. In fact, these three were picked because of their efficiency on big and complex datasets. While LGBMClassifier is really fast and efficient, CatBoost handles categorical features nicely, and generally, XGBoost is well-regarded for its robust performance in classification tasks.

Hyperparameters of the adopted ensemble approach for each model had been pre-set by specifying some values such as number of estimators, learning rate, max_depth among other parameters.

All of them have been trained on the same training set, and the Voting Classifier aggregates their outputs; hence, a more robust and accurate prediction by leveraging each model's strengths. This approach considerably improves performance by reducing overfitting issues and hence increases model stability.

4 Evaluation

The evaluation involved the use of four important metrics to compare the performances of the models, namely: Accuracy, Classification Report, Confusion Matrix, and AUC ROC Curve. These metrics provided full insights into the overall effectiveness of the model, the balance between precision and recall, the performance across classes, and its ability to discriminate between positive and negative classes.

4.1 Classification Report

The Classification Report is a really extended evaluation metric since it gives, in an extremely comprehensible way, insights into the performance of this model by calculating the main metrics-precision, recall, and f1-score-each for each class. These will show how well the model is recognizing each class while balancing between false positives and false negatives. Precision measures the correctness of positive predictions, recall tells how well the model has captured all relevant instances, while f1-score is the harmonic mean of precision and recall, and hence a balanced metric on imbalanced datasets.

All the models-the Random Forest Classifier and XGBoost models-performed rather similarly. The Gemini models and GPT-4 variants present relatively high values of precision and recall, especially for models with more than 700 samples. Models such as chatgpt-4o-latest and gpt-4o-2024-08-06 had to struggle a lot while giving accurate predictions in the case of less frequent models, with lower values of precision and recall. Similarly, the performance for CatBoost and LGBM Classifier showed the same trend. The models on which the performance was seen to be quite high include the gemma and llama variants. On an overall level, the performance of Voting Classifier was somewhat better in balanced precision and recall. Certain models, like llama-3-70b-instruct, gave very high f1-score outputs. However, the relatively low f1-scores of the model variants of chatgpt and gpt-4o do suggest some complication in handling those particular models.

4.2 Confusion Matrix

The confusion matrix is one of the basic metrics to evaluate the performance of a classification model. It is a summarization of the number of predictions for each class : TP(True Positive), TN(True Negative), FP(False Positive), and FN(False Negative). Each row in the matrix represents an actual class, and each column represents a class prediction. The diagonal elements of the matrix, from top left to bottom right, show the correct predictions, while the off-diagonal elements represent misclassifications.

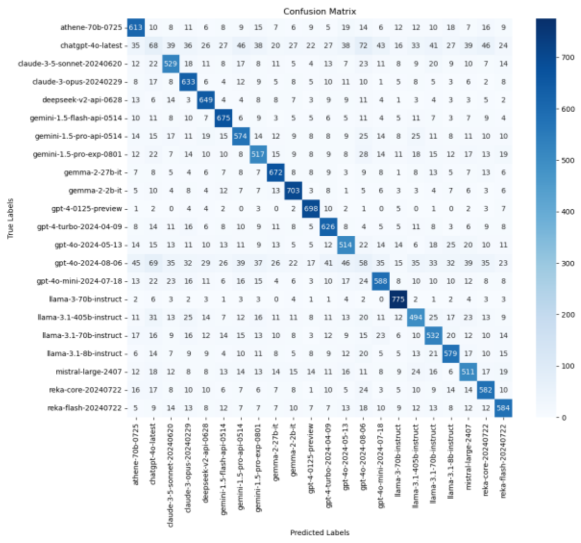

Figure 7: Confusion Matrix

Confusion matrices for all models (Random Forest, XGBoost, CatBoost, LGBM Classifier, and Voting Classifier) reflect the difficulty in predicting certain models. The Voting Classifier was the most balanced, having a higher number of correct predictions for the most frequent models, such as athene-70b-0725 and gpt-4-turbo-2024-08-06. The LightGBM and CatBoost also showed their performance for the models such as gemma-2-2b-it and llama-3-70b-instruct, whereas XGBoost struggled with a relatively poor performance for the models like chatgpt-4o-latest and gpt-4o-2024-08-06 due to lower representation in the dataset. Also, misclassifications for the rare models, including gpt-4o-2024-08-06 and chatgpt-4o-latest, were higher, further implying that these models face difficulties in predicting classes correctly, which have a lesser occurrence frequency. This again brings into focus the strengths of the models on common classes and their limitation in predicting instances that are rarer.

4.3 AUC ROC Curve

The AUC ROC Curve, or the Area Under the Receiver Operating Characteristic Curve, is a measure of performance for classification problems: it describes how well the model tells classes apart. The ROC curve plots the True Positive Rate against the False Positive Rate, with the diagonal line representing random guesses. AUC shows the whole capability of the model for distinguishing between the positive and negative classes. The greater the AUC, the better the quality of the model-the indication is that the model can have a good balance of both true positives and minimizing the false positives.

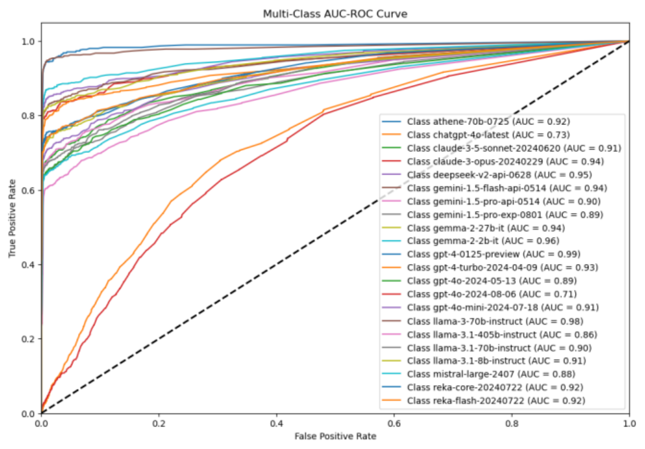

Figure 8: Multi-Class AUC-ROC Curve

From here, we can analyze further the AUC ROC Curves for different models to make out the performance difference exhibited by these models. LGBM, XGBoost, and CatBoost models keep a high AUC value consistently, with most classes above 0.9, which really means excellent performance in differentiating among the classes. This can also be seen from the Voting Classifier, with an AUC value for most classes nearing 1.0, hence combining several models seems to improve classification performance in general. ChatGPT-4o-latest consistently shows across all models lower AUC values, meaning it struggles in differentiating certain classes. Most notably, the Llama models have a high AUC, especially the llama-3-70b-instruct with a high value of 0.98, followed by the rest. However, in general, the Voting Classifier stands out when looking into the robustness section by being able to borrow the strengths of many other models to increase the final quality of the classification.

5 Result

The evaluation of several machine learning models was done: Random Forest, XGBoost, CatBoost, LGBM, and Voting Classifier. Their performance review gave quite impressive insights into their performance: while most of them, especially the LGBM, XGBoost, and CatBoost models, had very high classification AUC scores, a Voting Classifier outperformed others by combining the strong points of these models. This is to say that other models, for example ChatGPT-4o-latest, have demonstrated lower performances based on their AUC and precision score. These results make clear that model selection and ensemble techniques are of much importance and that a proper combination can improve classification results in tasks that are more challenging**.**

Model	Precision	Recall	F1-Score	Accuracy
Random Forest	0.67	0.69	0.68	0.69
XGBoost	0.66	0.68	0.66	0.68
CatBoost	0.64	0.66	0.65	0.66
LightGBM	0.67	0.68	0.68	0.68
Voting Classifier	0.67	0.69	0.67	0.69

Table 1: Performance Comparison of Different Models

6 Conclusions

This research came up with a methodology to classify different Large Language Models for specific tasks based on their features. Using XGBoost, LightGBM, and CatBoost together in a Voting Classifier, the system got an accuracy of 69%, which was better than all the individual models.

The steps to preprocess the data, including fixing class imbalance by adding or removing samples, were very important for training the models on a balanced dataset. Explored the data how features were distributed and how they related to different LLMs. Evaluation metrics such as accuracy, precision, recall, F1-score, and AUC-ROC proved that the proposed model was strong.

The findings are presented to improve the results of machine learning models on tasks using ensemble learning strategies. The suggested framework is one useful tool for researchers, helping them select the most suitable LLM according to a task to improve results.

7 Future Works

The proposed framework has shown large improvements in predicting the best Large Language Model for a given task. However, there are many areas for future research to make it even better and more useful. Firstly we can add more task features, like sentiment, sarcasm, or Variations in clarity which could help in understanding more subtle aspects of language. And also we can provide a real-time recommendation system that will offer instant suggestions for LLMs when current input of tasks.

Secondly, adding newer LLMs into the system would allow us to check how well they perform in different areas, staying up to date with quick changes in natural language processing. Another important area of exploration is deep learning approaches like neural networks or transformers for the classification task that might capture even more complex patterns in data.

Finally, transparency in the ways would show the users why a certain LLM is being suggested for a task, and that would build more trust and openness in the system. By looking at these areas, future research can add to the groundwork set by this study, making LLM recommendation systems more effective and useful in different natural language processing tasks.

REFERENCES

[1] [Huang, Jiaxin, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. "Large language models can self-improve." arXiv preprint arXiv:2210.11610 (2022).

[2] Alberts, Ian L., Lorenzo Mercolli, Thomas Pyka, George Prenosil, Kuangyu Shi, Axel Rominger, and Ali Afshar-Oromieh. "Large language models (LLM) and ChatGPT: what will the impact on nuclear medicine be?." European journal of nuclear medicine and molecular imaging 50, no. 6 (2023): 1549-1552.

[3] Bao, Keqin, Jizhi Zhang, Yang Zhang, Wenjie Wang, Fuli Feng, and Xiangnan He. "Tallrec: An effective and efficient tuning framework to align large language model with recommendation." In Proceedings of the 17th ACM Conference on Recommender Systems, pp. 1007-1014. 2023.

[4] Zhang, Jizhi, Keqin Bao, Yang Zhang, Wenjie Wang, Fuli Feng, and Xiangnan He. "Is chatgpt fair for recommendation? evaluating fairness in large language model recommendation." In Proceedings of the 17th ACM Conference on Recommender Systems, pp. 993-999. 2023.

[5] Ding, Ning, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu et al. "Parameter-efficient fine-tuning of large-scale pre-trained language models." Nature Machine Intelligence 5, no. 3 (2023): 220-235.

[6] Chen, Yanda, Ruiqi Zhong, Sheng Zha, George Karypis, and He He. "Meta-learning via language model in-context tuning." arXiv preprint arXiv:2110.07814 (2021).

[7] Zheng, Yaowei, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. "Llamafactory: Unified efficient fine-tuning of 100+ language models." arXiv preprint arXiv:2403.13372 (2024).----

[8] Luo, Haoran, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong et al. "Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models." arXiv preprint arXiv:2310.08975 (2023).

[9] Dunn, Alexander, John Dagdelen, Nicholas Walker, Sanghoon Lee, Andrew S. Rosen, Gerbrand Ceder, Kristin Persson, and Anubhav Jain. "Structured information extraction from complex scientific text with fine-tuned large language models." arXiv preprint arXiv:2212.05238 (2022).

[10] Zhang, Hang, Xin Li, and Lidong Bing. "Video-llama: An instruction-tuned audio-visual language model for video understanding." arXiv preprint arXiv:2306.02858 (2023).

[11] Singhal, Karan, Shekoofeh Azizi, Tao Tu, S. Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales et al. "Large language models encode clinical knowledge." arXiv preprint arXiv:2212.13138 (2022).---

[12] Wang, Haifeng, Jiwei Li, Hua Wu, Eduard Hovy, and Yu Sun. "Pre-trained language models and their applications." Engineering 25 (2023): 51-65.

[13] Wu, Shengqiong, Hao Fei, Leigang Qu, Wei Ji, and Tat-Seng Chua. "Next-gpt: Any-to-any multimodal llm." arXiv preprint arXiv:2309.05519 (2023).

[14] Zhuang, Yuchen, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. "Toolqa: A dataset for llm question answering with external tools." Advances in Neural Information Processing Systems 36 (2023): 50117-50143.

[15] Lyu, Hanjia, Song Jiang, Hanqing Zeng, Yinglong Xia, Qifan Wang, Si Zhang, Ren Chen, Christopher Leung, Jiajie Tang, and Jiebo Luo. "Llm-rec: Personalized recommendation via prompting large language models." arXiv preprint arXiv:2307.15780 (2023).

[16] Li, Lei, Yongfeng Zhang, and Li Chen. "Prompt distillation for efficient llm-based recommendation." In Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, pp. 1348-1357. 2023.

[17] Jin, Ming, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y. Zhang, Xiaoming Shi, Pin-Yu Chen et al. "Time-llm: Time series forecasting by reprogramming large language models." arXiv preprint arXiv:2310.01728 (2023).

[18] Gao, Yeqi, Zhao Song, Weixin Wang, and Junze Yin. "A fast optimization view: Reformulating single layer attention in llm based on tensor and svm trick, and solving it in matrix multiplication time." arXiv preprint arXiv:2309.07418 (2023).

[19] de Zarzà, Irene, Joachim de Curtò, Gemma Roig, and Carlos T. Calafate. "LLM multimodal traffic accident forecasting." Sensors 23, no. 22 (2023): 9225.

[20] Inan, Hakan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev et al. "Llama guard: Llm-based input-output safeguard for human-ai conversations." arXiv preprint arXiv:2312.06674 (2023).