MACHINE LEARNING
FINAL PROJECT
2025

PREPARED BY:
RACHEL BELOKOPYTOV
NOA FISHMAN

Introduction:

After a semester exploring various topics in Machine Learning (ML), a final project was assigned, allowing flexibility in structure and topic selection. Throughout the year, different projects were explored, covering areas such as NLP, clustering, and more. Therefore, to diversify our experience, this project followed the following structured approach:

1. Choosing an interesting dataset.
2. Raising several hypothetical questions.
3. Visualizing the data to better understand it and its connections.
4. Selecting tasks that can be addressed using different machine learning and deep learning models.
5. Designing a user interface.

The 'Leetcode Problem' Dataset from Kaggle was chosen because it is highly relevant to the next step after completing the computer science bachelor's degree - technical interviews. As students in the final year, preparing for job interviews and coding challenges is a major focus. Since Leetcode is widely used for interview preparation, analyzing its problems using machine learning seemed both useful and interesting.

This project explores problem characteristics, predicts difficulty levels, and finds patterns in related topics. It not only applies machine learning concepts but also provides insights that align with technical interview preparation.

1. Data:

To build an accurate and effective project, we used a dataset of 'Leetcode Problem Dataset' taken from Kaggle.com website [1]. This dataset contains entries, each representing a question (problem) from the Leetcode website. The dataset is rich with features that provide a complete view of each question. The dataset consists of 17 columns and 1,825 rows, with the following features:

- Id – problem id
- Title – problem name
- Description – problem description
- Is_premium – whether the question requires a premium account, 1 for yes, 0 for no
- Difficulty – easy, medium, hard
- Acceptance_rate - how often the answer submitted is correct
- Frequency - how often the problem is attempted
- Discuss_count - how many comments are submitted by users
- Accepted - how many times the answer was accepted

- Submissions - how many times the answer was submitted
- Companies - which companies were tagged as having asked this specific problem
- Related_topics - related topics to the current problem
- Likes – how many likes the problem got
- Dislikes - how many dislikes the problem got
- Rating – likes / (likes + dislikes)
- Asked_by_faang - whether the question was asked by Facebook, Apple, Amazon, Netflix, or Google - 1 for yes, 0 for no
- Similar_questions - similar problems with problem name, slug, and difficulty

After an initial look at the data, several questions about the data and the relationships between the features were raised.
Most of the questions were on the topic of whether the difficulty level of the question affects other features in one way or another. For example, do harder questions imply more discussion around the question (discuss_rate) (fig. 5.)? Do certain topics appear more in questions of a certain difficulty level (related_topics) (fig. 7.)? Do certain words in the question description indicate a certain difficulty level (fig. 8.)? etc. Additional questions arose on other topics, such as whether there is a connection between the rating and FAANG asking the question, whether certain companies prefer to ask certain topics (fig. 9.)? and so on. The answers to this type of questions were mostly inferable from the plots chosen to visualize the data (part 2). But there were more questions that required better models to answer them. Here are a few questions for which different models have been built to answer them:

- Is it possible to predict the number of likes/ dislikes of a problem?
- Is it possible to predict how many accepted submissions there'll be given a problem?
- Is it possible to predict the difficulty level of a problem given its description?
- Is it possible to predict the related topics of a question based on its description?

Deeper explanation and discussion of these questions and models will be shown in part 3.

To prepare the dataset for the machine learning models, several preprocessing steps were applied to convert raw data into numerical representations. These steps included encoding categorical variables, transforming numerical features, and extracting meaningful information from text descriptions:

- The related_topics and companies columns contained multiple values per entry, which were converted into a multi-hot encoded format using the MultiLabelBinarizer. This transformation enabled each question to have multiple associated topics and companies, represented as binary features.

- The difficulty and title columns were label-encoded, mapping text values to numerical representations to facilitate model training.
- The submissions and accepted columns included shorthand notations such as "K" for thousands and "M" for millions. These values were converted into standard numerical format to ensure consistency in calculations.
- The description column was processed using lemmatization with the spaCy library, standardizing words to their base forms. A manual word-mapping step replaced common programming-related shortenings like, "arr" to "array," "nums" to "numbers" etc. to improve text consistency. And in the end, TF-IDF vectorizer was applied to extract the 100 most relevant words from descriptions, converting them into numerical vectors that could be used as input features for models.

These preprocessing steps ensured that the data was structured and optimized for various machine learning tasks.

2. Data visualization:

Visualization of the data was needed to understand the data better and the connections between the features. Some of the questions raised could be answered by looking at some specific graphs. It is important to point out that these visualizations are based on this dataset specifically.

These first visualizations reveal two key insights about LeetCode's problem distribution in this data: First, medium difficulty problems dominate the platform at 53%, followed by easy (26%) and hard (21%) problems, showing a balanced progression in difficulty. Second, only 22% of problems are premium (requiring paid access), with the majority (78%) being

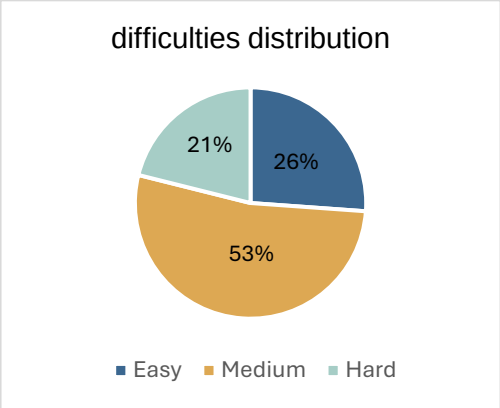


Fig. 1. Pie chart of the difficulties distribution

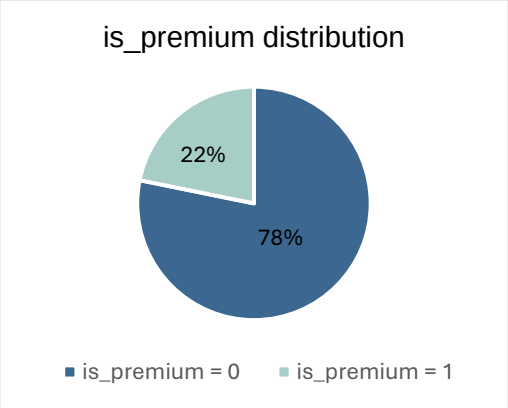


Fig. 2. Pie chart of the is_premium distribution

Is Premium vs. Difficulty Table

Is Premium	Easy	Hard	Medium	Total
0	370	313	744	1427
1	107	72	219	398
Total	477	385	963	1825

Fig. 3. Table showing the breakdown of problems by difficulty level and whether they are Premium or not

freely available, while the accompanying table breaks down how these premium vs. free problems are distributed across difficulty levels, showing that the ratio of free-to-premium problems remains fairly consistent across all difficulty categories.

This line plot (fig. 4.) demonstrates the relationship between problem difficulty and acceptance rates. The trend shows a clear inverse relationship - as difficulty increases, the acceptance rate tends to decrease. This pattern validates the difficulty classifications and provides users with expectations about success rates at different difficulty levels.

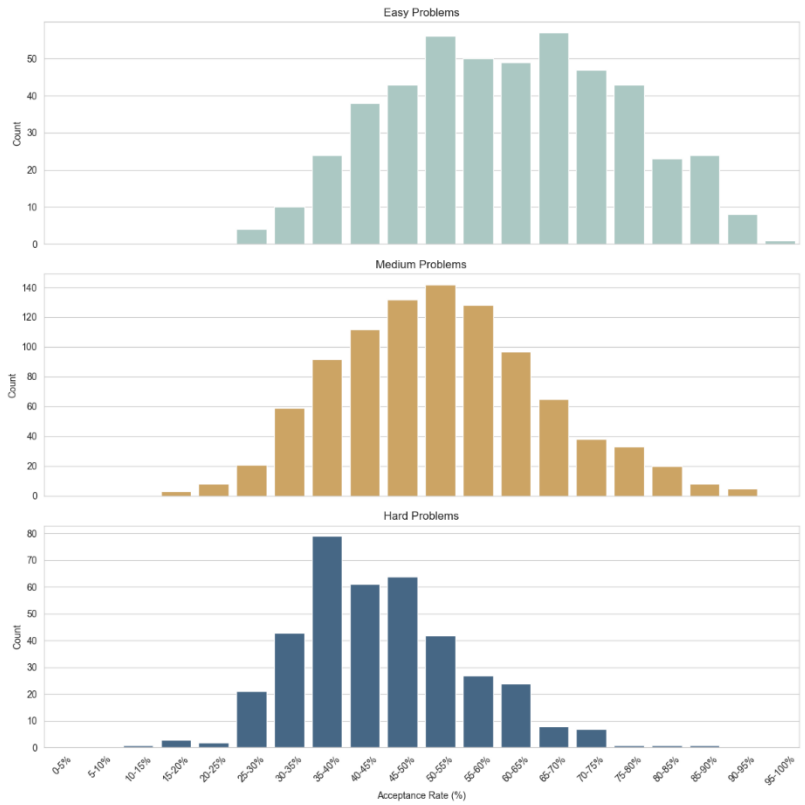


Fig. 4. Distribution of acceptance rate by difficulty level

This next visualization (fig. 5.) shows the distribution of discussion counts across easy, medium, and hard problems. The data reveals an interesting engagement pattern. This pattern suggests that medium difficulty problems tend to generate the most community

engagement, possibly because they strike a balance between being challenging enough to warrant discussion but not so difficult that fewer people attempt them.

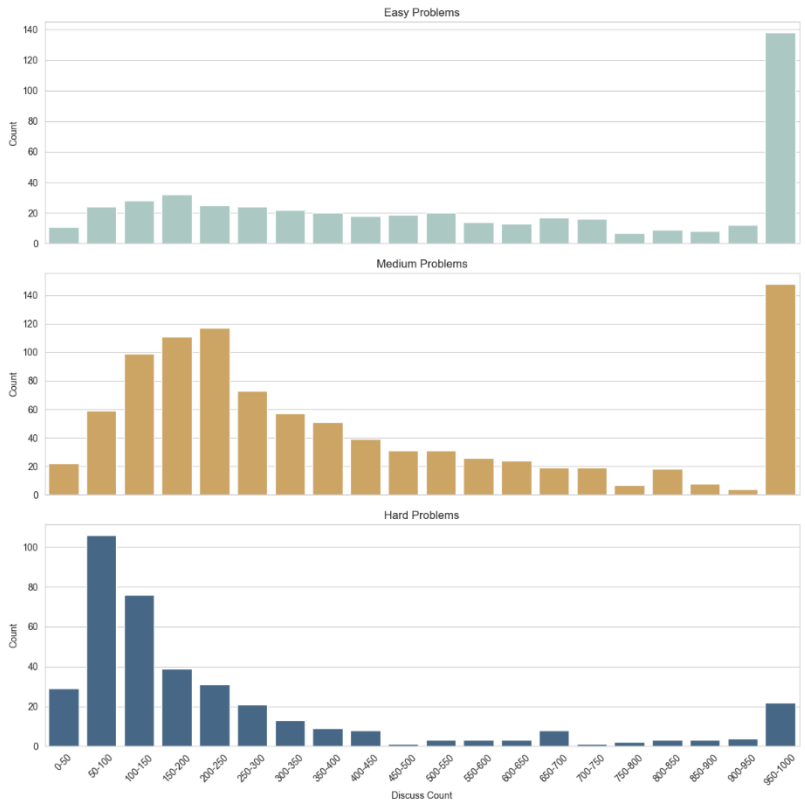


Fig. 5. Distribution of discussion count by difficulty level

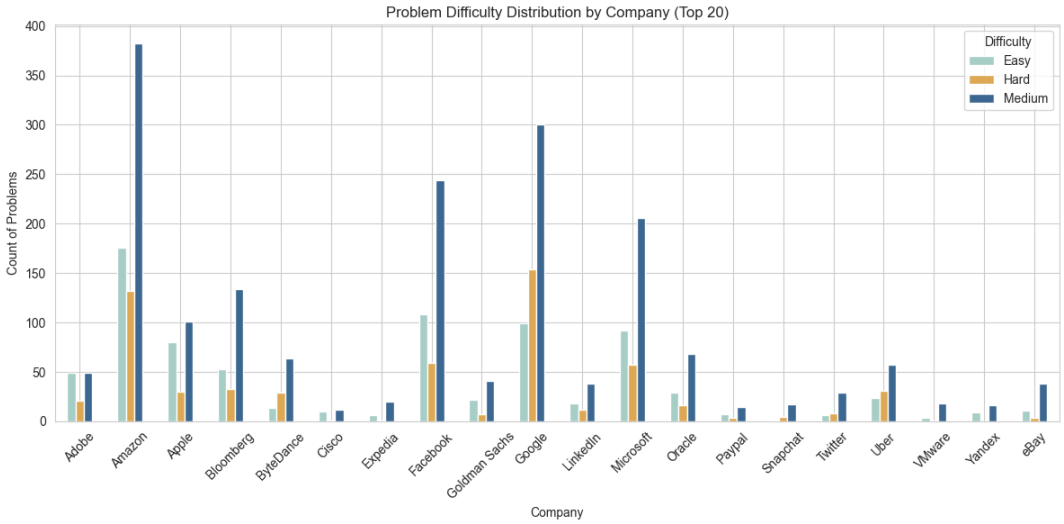


Fig. 6. Distribution of problem count by difficulty level for each company

Here the visualization (fig. 6.) illustrates how different companies distribute their interview questions across difficulty levels. For example: Amazon leads with the highest number of

problems, with a strong emphasis on medium-difficulty questions, Google and Facebook also maintain large problem sets, but with more balanced difficulty distributions, etc. One can infer that most companies maintain a rough ratio of medium > easy > hard problems in their question banks. This information is particularly valuable for job seekers targeting specific companies, as it helps them understand what level of difficulty to expect in interviews.

The next bar chart (fig. 7.) illustrates the distribution of problem difficulties (Easy, Medium, Hard) across different topics. It shows that while some topics like 'Array' and 'String' have a more balanced distribution across difficulty levels, others like 'Dynamic Programming' and 'Graph' tend to have a higher proportion of medium and hard problems. This information is valuable for users to understand which topics might require more advanced preparation.

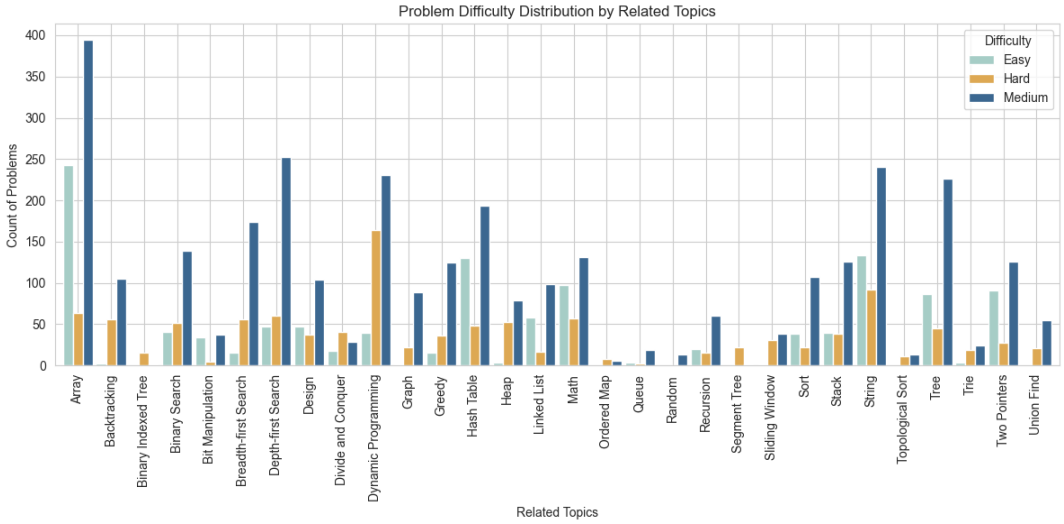


Fig. 7. Distribution of problem count by difficulty level for each related topic

This bar chart (fig. 8.) shows the frequency of top used words appearing in problem descriptions across different difficulty levels. Notable patterns include, among other things, certain technical terms are more dominant in harder problems (e.g., 'array', 'tree'), medium difficulty problems show the most diverse vocabulary and easy problems tend to use simpler, more straightforward terminology. This analysis helps understand how problem descriptions correlate with difficulty levels and could be useful for, later, difficulty classification.

The heatmap (fig. 9.) visualizes the relationship between companies and the types of programming topics they focuses on in their interview questions. The darker blue indicates a stronger preference for certain topics. This visualization is based on the ratio within each company, meaning that the intensity of a topic reflects its relative emphasis compared to other topics within the same company. For example, Amazon and Google show the most

diverse topic coverage, while data structures like Arrays and Trees appear consistently across most companies. This helps candidates understand which technical areas to focus on when preparing for interviews at specific companies.

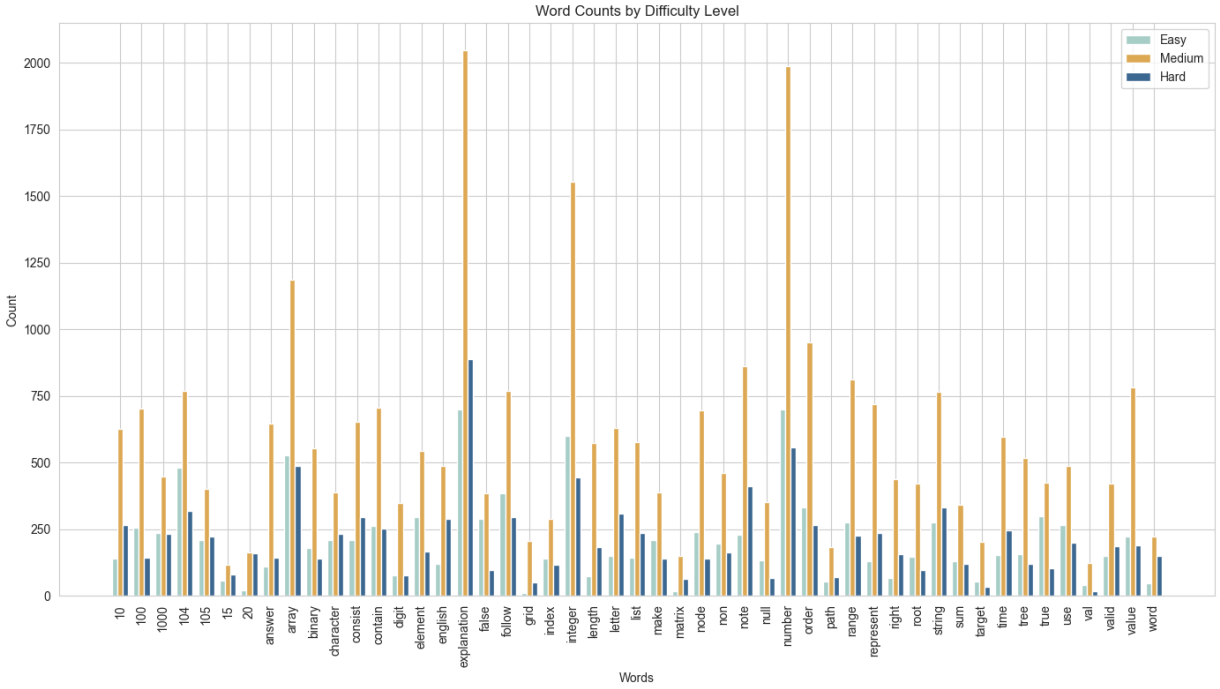


Fig. 8. Distribution of word count by difficulty level

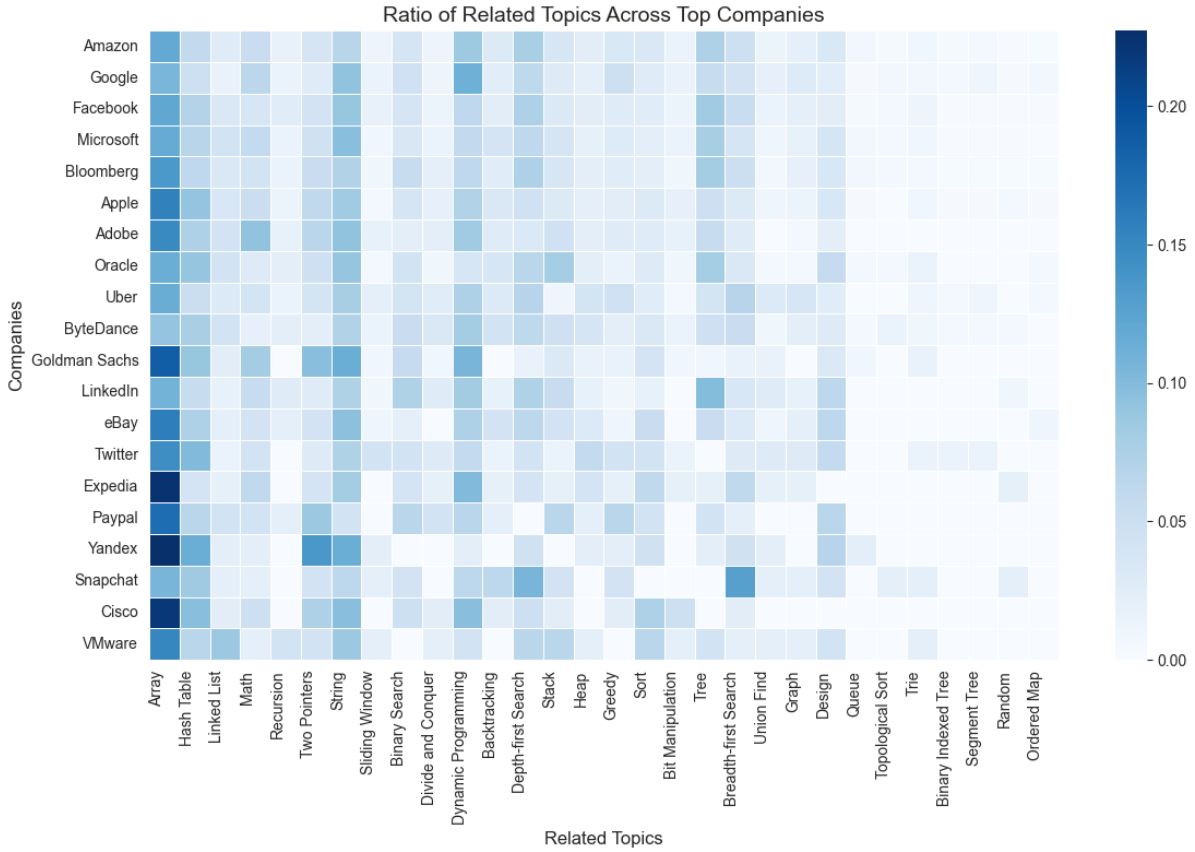


Fig. 9. Ratio of related topics across top 20 companies

3. Models:

After thoroughly reviewing the data and formulating key questions, we began assembling our models. Throughout this process, we explored a variety of machine learning models to understand their strengths and weaknesses. To gain insights from as many models as possible, we decided to apply different models to each question and compare their results.

Given that different questions require different approaches, some of our problems called for classification models to predict labels, while others required regression models to predict continuous values. To establish a good foundation, we turned to various large language models (LLMs) for an initial structure, making the necessary adjustments and adding key features we identified as important.

Starting with the basic questions and their models:

Question 1 - Is it possible to predict the number of accepted submissions?
An 'accepted_submissions_regression' function was created for this purpose. The function performs linear regression to predict the number of accepted submissions for a given question based on features such as the number of submissions, difficulty level, discussion count, and whether the question is premium. It first splits the input data into training and testing sets, then trains a linear regression model on the training data. After training, the model is used to make predictions on the test data, and its performance is evaluated using metrics like Mean Absolute Error (MAE) and Mean Squared Error (MSE). Finally, the function generates a scatter plot (fig. 10.) comparing the actual versus predicted accepted submissions, offering a visual representation of the model's accuracy.

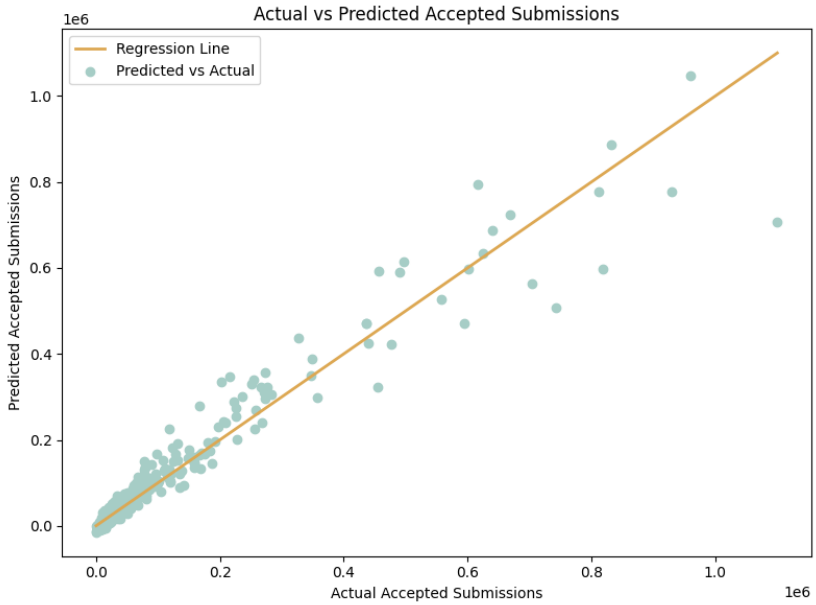


Fig. 10. Scatter plot of actual Vs. predicted accepted submissions

Question 2 - Is it possible to predict the number of likes and dislikes of a problem?

For this purpose, a few different models were used in order to check which one is better. Each model was used to predict the number of likes, and number of dislikes separately. The models are:

- **Linear Regression** is a method used to model the relationship between a dependent variable and one or more independent variables. In this case, it predicts the target variable (likes or dislikes) based on features like difficulty, frequency, discuss count, submissions, etc. It is simple, interpretable, and computationally efficient.
- **Random Forest** is an ensemble learning method that combines multiple decision trees to improve predictive accuracy and control overfitting. In this function, it is used to predict likes or dislikes based on the selected features. Each tree is trained on a subset of the data, and the final prediction is the average of all trees.
- **Gradient Boosting** is another ensemble learning method that builds models sequentially, each new model correcting the errors made by the previous one. The model fits weak learners to the residuals of the previous models.
- **XGBoost** (Extreme Gradient Boosting) is an optimized version of gradient boosting that is faster and more efficient. It implements regularization techniques to prevent overfitting and provides parallelization during training, making it faster than traditional gradient boosting methods.

After training, evaluating each model based on the evaluation metrics - MAE, MSE, RMSE – and displaying scatter plots for the prediction's performances of each model. The 'plot_comparison_histogram' function allowed for a visual comparison of the models' performance, specifically focusing on the RMSE and MAE metrics, to determine which model best predicted the target column (either likes or dislikes) (fig. 11.). The 'get_best_model' function helped identify the best-performing model (fig. 12.) based on

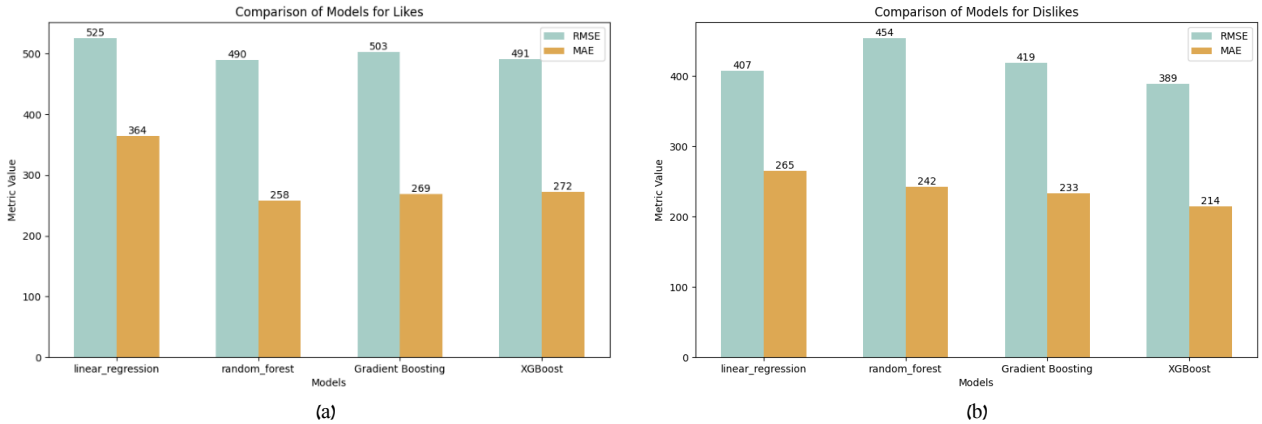


Fig. 11. Histograms showing the comparison between the models based on RMSE and MAE. (a) for the likes prediction and (b) for the dislikes

these metrics, allowing for informed decision-making on which model to use moving forward (part 4).

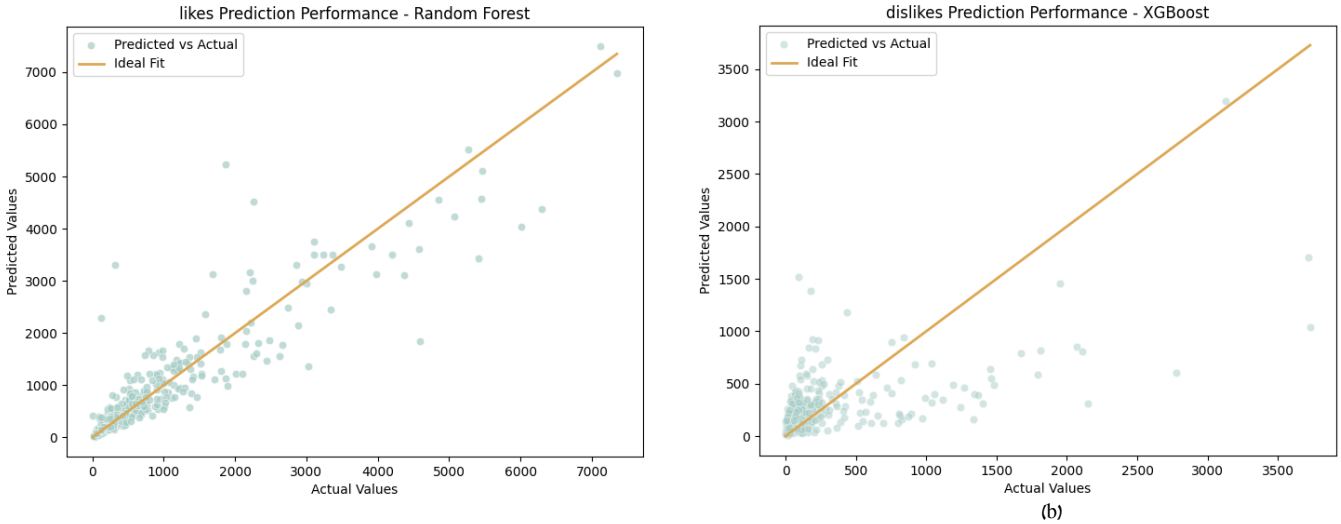


Fig. 12. Scatter plots of the actual Vs. predicted of the best evaluated models. (a) for likes prediction (b) for dislikes prediction

Moving forward to the more complex (and interesting) questions:

Question 3 - Is it possible to predict the difficulty level of a problem?

The goal was to develop an effective approach that accurately predicts difficulty while comparing different modeling techniques to understand their strengths and limitations. After some consultation with some LLMs, the classification was chosen to be performed using both traditional machine learning models and a neural network. The function 'difficulty_classification' applies multiple machine learning models using a feature set consists of metadata such as is_premium, acceptance_rate, rating, and discuss_count, along with top words from TF-IDF and related topics. To address class imbalance, SMOTE (Synthetic Minority Over-sampling Technique) is used. The machine learning models the function ran are:

- **Logistic Regression** is a simple and interpretable classification model that estimates the probability of a given input belonging to a specific class using the sigmoid function. It works well when relationships between features and target labels are mostly linear. In this task, it provided a baseline for performance, but its limited ability to capture complex interactions between features made it less effective than tree-based models.
- **Random Forest** captures non-linear relationships and manages missing data well, thus, often performs robustly on structured datasets like this one.

- **Support Vector Machine** (SVM) is a powerful classifier that works by finding the optimal hyperplane that separates different classes with the maximum margin. It can efficiently handle high-dimensional spaces and is robust to overfitting, especially when using kernel functions to capture non-linear relationships.
- **K-Nearest Neighbors** (KNN) is a non-parametric model that classifies a sample based on the majority class of its k closest neighbors in the feature space. It does not make assumptions about the data distribution, making it flexible, but its performance heavily depends on the choice of k and the distance metric. It can struggle with high-dimensional data and large datasets due to computational complexity.
- **Gradient Boosting** is highly effective for structured data and can capture complex patterns in the features.
- **XGBoost** which includes features like regularization, parallel processing, and tree pruning, making it one of the most popular models for structured classification tasks. XGBoost stands out at handling imbalanced datasets, missing values, and feature importance analysis, making it a strong candidate for difficulty classification.

In the process of difficulty classification, identifying the most influential features for predicting difficulty is crucial. Traditional machine learning models, such as Random Forest, XGBoost, and Gradient Boosting, offer built-in methods for feature importance estimation. For example, XGBoost and Gradient Boosting provide feature importance based on metrics like gain, cover, and frequency. These methods help identify which features - such as is_premium, acceptance_rate, rating, discuss_count, and top words from TF-IDF - are most predictive of problem difficulty. Features like is_premium and rating, for instance, could be strong indicators of problem difficulty, reflecting the exclusivity and perceived challenge of a problem.

After running different ML models, a deep learning model was used to see if it can outperform the ML models. The function ‘difficulty_classification_nn’ employs a deep learning approach with a simple three-layer feedforward neural network. The model consists of fully connected layers with Batch Normalization, ReLU activations, and Dropout to prevent overfitting. The training process used AdamW optimizer, learning rate scheduling (ReduceLROnPlateau), and early stopping. Data is split into training, validation, and test sets, and the model is trained for up to 100 epochs. The function tracks loss and accuracy across epochs, visualizing trends, and model performance per difficulty level. Based on the fact that this course focused on ML models and not DL models, a simple neural network was used.

The models’ parameters were determined after experimenting and running the models with different hyperparameters. Those chosen for use were those that gave the best results according to the given data.

The primary evaluation metric used to assess the performance of the machine learning and deep learning models was accuracy which is the proportion of correct predictions out of the total number of predictions made. This metric provides a clear and straightforward evaluation of how well the model is classifying the difficulty levels of the problems. The accuracy was measured for each model so that in more advanced stages (part 4) the most accurate model will be used. Here it was the Random Forest model.

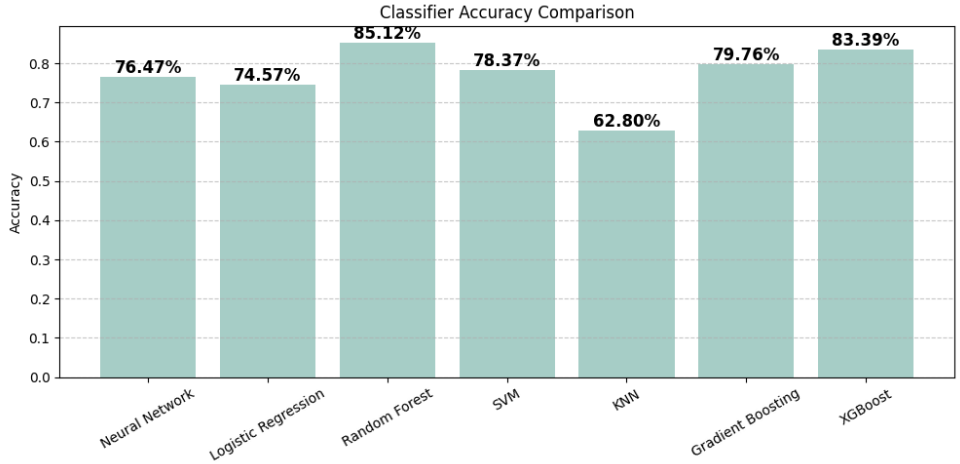


Fig. 13. Accuracy comparison of difficulty classification models

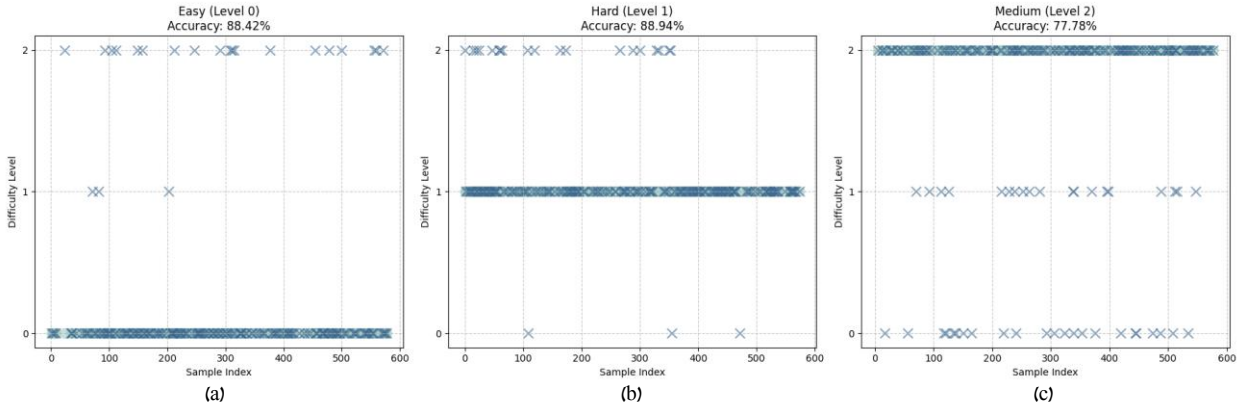


Fig. 14. Graphs that display the actual Vs. predicted labels across different sample indexes of the test set for the random forest model. Colored circle being the actual label and X being the predicted label. (a) for the samples with actual label 'easy' (b) for 'hard' and (c) for 'medium'

Question 4 - Is it possible to predict the list of the related topics of a problem?

This is a multi-label classification task where the goal is to predict the related topics for a given problem description. The dataset consists of problem descriptions and their associated topics, with each description potentially belonging to multiple topics. The main intentions of the 'related_topics_prediction' code are to train multiple machine learning models, optimize prediction thresholds for each label, evaluate model performance, and

select the best performing model based on several metrics for future use. There are several machine learning models used to perform the multi-label classification task:

- **Support Vector Machine** (SVM)
- **Logistic Regression:** A multinomial logistic regression model.
- **Random Forest**
- **K-Nearest Neighbors** (KNN)
- **Gradient Boosting**
- **XGBoost**

This time, the SVM, Logistic Regression and Random Forest models were used with a One-vs-Rest (OvR) strategy, where a separate binary classifier is trained for each class (topic).

In multi-label classification, each label (topic) is predicted independently, and the model outputs probabilities for each label. Instead of using a fixed threshold of 0.5 to classify predictions, the model optimizes thresholds for each label individually, based on the F1 score. This approach ensures that the threshold is fine-tuned to maximize the performance for each topic, leading to more accurate predictions. The thresholds are determined using Stratified K-Fold Cross-Validation, where the optimal threshold for each label is averaged across folds to reduce overfitting. After optimization, predictions are made by comparing the predicted probabilities to the respective thresholds, improving the classification precision.

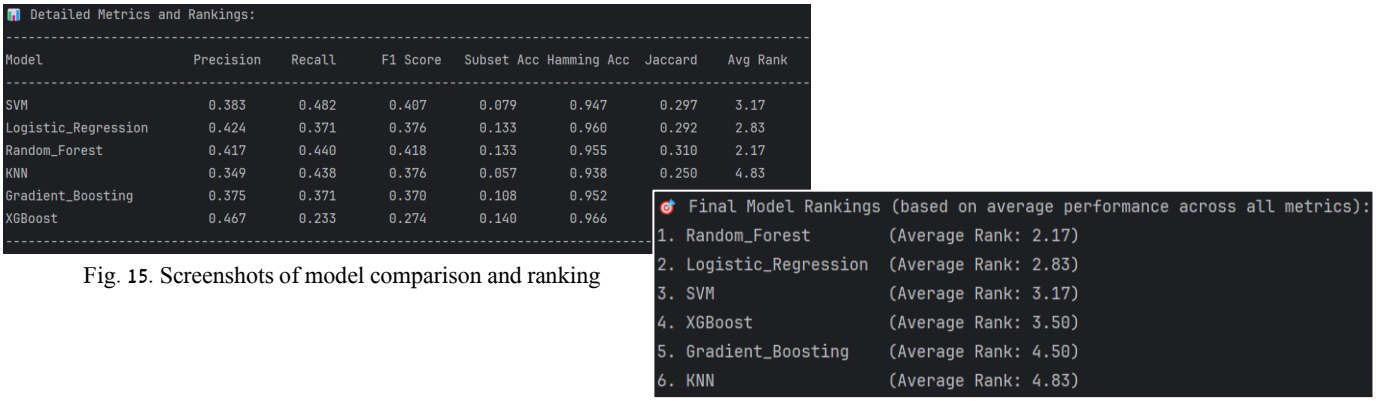
Accuracy can be a misleading metric in multi-label classification tasks because it does not capture the nuances of the problem and can count as too strict. Thus, a different way to evaluate the model was needed. The LLMs suggested many different metrics to use for that purpose. After exploring and deep understanding the metrics, the following metrics was chosen to evaluate the model:

- **Weighted Precision** averages the precision of each class, but each class's precision is weighted by the number of true instances in that class. This ensures that classes with more examples contribute more to the final score.
- **Weighted Recall** works similarly, giving more importance to classes with more true instances when calculating the overall recall which indicates how well the model identifies all the correct labels for each instance.
- **Weighted F1 Score**: The harmonic mean of the weighted precision and weighted recall, providing a single measure that balances both metrics, especially useful when dealing with imbalanced datasets.

- **Subset Accuracy** measures how often the model predicts exactly the same set of labels as the true labels.
- **Hamming Accuracy**: A metric that computes the accuracy as the fraction of labels that are correctly predicted. It is the complement of Hamming loss.
- **Jaccard Score** measures the similarity between the predicted and actual sets of labels, focusing on the intersection over union of predicted and actual labels.

The weighted metrics help avoid bias towards classes with fewer instances and provide a more balanced evaluation when dealing with imbalanced datasets.

Based on these metrics and a ranking method built with LLM that aggregates all the results, the most accurate model was Random Forest which will be helpful later (part 4).



Detailed Metrics and Rankings:

Model	Precision	Recall	F1 Score	Subset Acc	Hamming Acc	Jaccard	Avg Rank
SVM	0.383	0.482	0.407	0.079	0.947	0.297	3.17
Logistic_Regression	0.424	0.371	0.376	0.133	0.960	0.292	2.83
Random_Forest	0.417	0.440	0.418	0.133	0.955	0.310	2.17
KNN	0.349	0.438	0.376	0.057	0.938	0.250	4.83
Gradient_Boosting	0.375	0.371	0.370	0.108	0.952		
XGBoost	0.467	0.233	0.274	0.140	0.966		

Final Model Rankings (based on average performance across all metrics):
1. Random_Forest (Average Rank: 2.17)
2. Logistic_Regression (Average Rank: 2.83)
3. SVM (Average Rank: 3.17)
4. XGBoost (Average Rank: 3.50)
5. Gradient_Boosting (Average Rank: 4.50)
6. KNN (Average Rank: 4.83)

Fig. 15. Screenshots of model comparison and ranking

4. User interface:

To make our predictive models more accessible and user-friendly, we developed an interactive user interface using Gradio [2]. The interface allows users to input relevant details about a Leetcode problem and select the type of prediction they want to generate. The UI seamlessly integrates with the best-performing machine learning models identified during our evaluation, providing predictions in real time.
The interface provides input fields for users to enter key problem attributes such as:

- Title & Description – The textual details of the Leetcode question.
- Difficulty Level – Easy, Medium, or Hard.
- Related Topics – Select multiple relevant topics associated with the problem.
- Likes, Accepted Submissions, Total Submissions, and Comments - inputs reflecting user engagement metrics.
- Premium Status – Whether the problem requires a premium subscription.

Users can choose from five types of predictions:

- Related Topics Prediction – Identifies topics that best describe the problem.
- Difficulty Level Prediction – Classifies the problem as Easy, Medium, or Hard.
- Acceptance Rate Prediction – Estimates the number of accepted solutions.
- Likes Prediction – Predicts the number of likes a problem might receive.
- Dislikes Prediction – Predicts the number of dislikes.

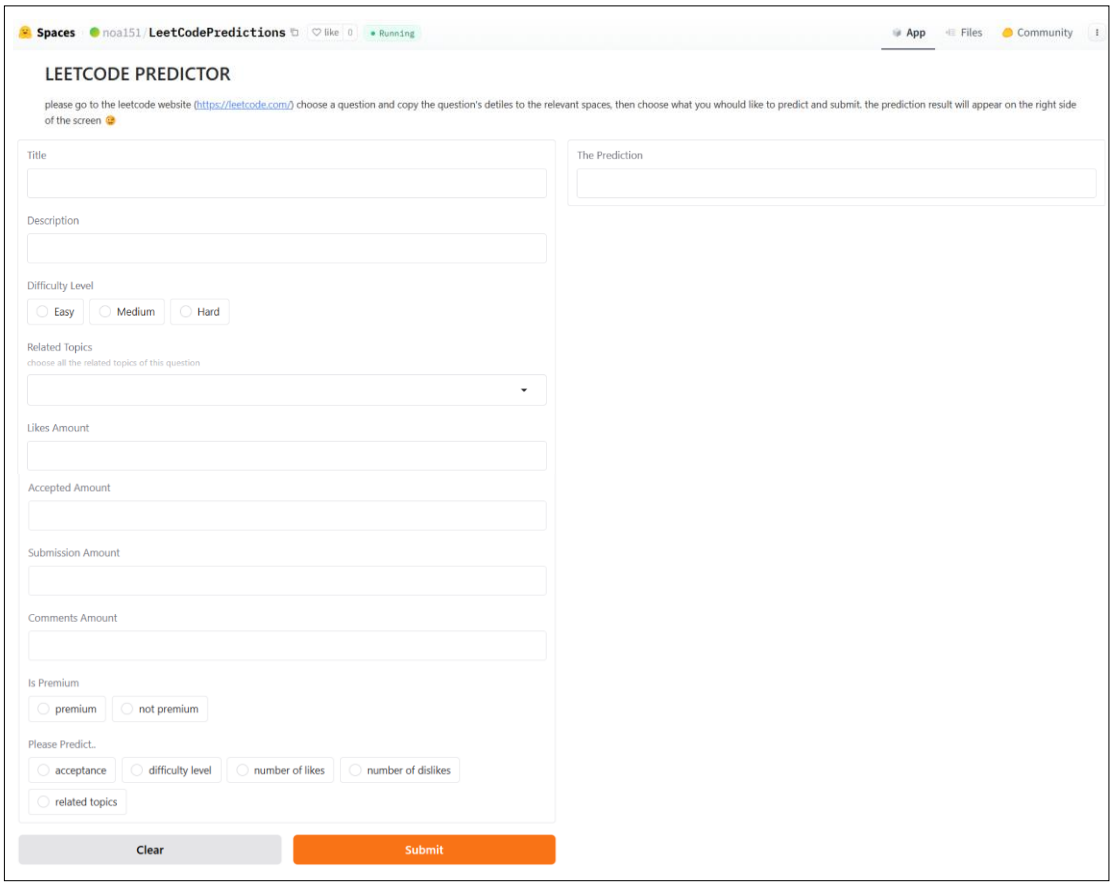


Fig. 16. Screenshot of the user interface

The UI processes user's inputs into a structured format, applying TF-IDF vectorization to convert descriptions into numerical features. Relevant features, like related topics and companies, are encoded using multi-hot encoding. The best-performing models, determined based on their evaluation metrics, were saved after training, and are now dynamically loaded when needed. This ensures that only the most accurate models are used for making predictions. Of course, each model uses its relevant features. Finally, the results are displayed instantly, providing users with valuable insights into the given problem and the asked prediction.

While the interface offers predictions in real time, it is important to note that the predictions are not always perfect. This is due to limitations in the training data, model complexity, and the inherent unpredictability of certain problem characteristics. The models provide approximations based on available data, and as such, the predictions should be used as helpful insights rather than definitive answers.

This UI enhances accessibility by allowing users to interact with complex ML models without requiring programming knowledge. By integrating the models into the simple web-based interface, we make it easier for users to analyze and predict key problem characteristics directly from the Leetcode dataset.

5. Future work:

Moving forward, we aim to expand and refine both the models, and the user interface developed in this project. Our goal is to create a platform that simulates the functionality of Leetcode, incorporating advanced machine learning and NLP techniques to enhance user experience and prediction accuracy.

One of the key directions for future work is to develop an intelligent recommendation system that suggests problems to users based on their skill level, past performance, and areas of interest. This would involve collaborative filtering techniques, content-based filtering, and recommendation models. Additionally, integrating NLP models to evaluate users' submitted answers could provide feedback and suggestions for improvement, making the platform more interactive and educational.

From a modeling perspective, while traditional machine learning models outperformed deep learning models in this project, there are several ways to improve neural network performance in future iterations. This can be done by exploring deeper architectures that can better capture complex relationships within the dataset, tuning the hyperparameters, expanding the dataset, either by collecting more labeled data or leveraging data augmentation techniques for text-based features, and by using more fine-tuning pre-trained deep learning models (e.g., transformer-based models like BERT) to enhance feature extraction from the problem descriptions.

By implementing these improvements, we can build a more robust and intelligent system that not only predicts problem characteristics but also guides users through coding challenges in a dynamic and adaptive manner.

6. Discussion:

Throughout the process of building and refining the models, several key challenges and improvements were encountered. The development of predictive models required multiple iterations of experimentation, evaluation, and fine-tuning.

Changes in model development - Various machine learning models were tested for different tasks. Regression models predicted numerical values like likes, dislikes, and accepted submissions, while classification models handled difficulty levels and related topics. Ensemble models like XGBoost and Random Forest consistently outperformed simpler models, highlighting the importance of feature selection and hyperparameter tuning. For difficulty classification, class imbalance was a key challenge. Applying SMOTE improved accuracy for underrepresented difficulty levels, while feature selection, including TF-IDF vectorization, further enhanced performance. We tried to predict question titles from descriptions using an LSTM model with cosine similarity loss. Despite fine-tuning, the model's accuracy was insufficient due to weak text embeddings and mostly limited data making pattern learning difficult. Therefore, as said before, future work could focus on collecting more data or leveraging pre-trained language models for improved results.

Usage of Large Language Models (LLMs) in model development - LLMs like GPT-4 and Claude were used in several stages of the project. These models provided valuable assistance mostly in:

- Code Generation and Optimization: Several preprocessing and modeling functions were initially drafted using LLMs, which were later improved based on experimental results. LLMs also assisted in debugging and improving efficiency by suggesting alternative approaches.
- Hyperparameter Tuning Suggestions: By leveraging LLMs, initial hyperparameter configurations were generated for machine learning models, which were later fine-tuned based on empirical results.
- Data Visualization: LLMs assisted in generating the visualizations in the dataset, making it easier to interpret relationships between features discussed in part 2.

“And, of course, LLMs helped fine-tune this very report – because even a machine learning project needs a little human-like polish! ;) ” (GPT-4)

7. Conclusions:

This project successfully applied machine learning and deep learning to analyze and predict attributes of Leetcode problems. Ensemble models like Random Forest and XGBoost excelled, while deep learning faced limitations due to dataset size, highlighting the need for future improvements like data augmentation and deeper learning.

The integration of a user-friendly interface further solidified the project's impact, making complex ML models accessible to users without technical expertise. This interface allows users to interact with models in real time, reinforcing the practical value of our work.

Overall, this project provided valuable insights into machine learning applications in technical problem analysis. Early on, we posed several key questions, and through the course of the project, we were able to answer those and even solve more difficult problems that came up along the way.

8. Links:

[1] Data – https://www.kaggle.com/datasets/gzipchrist/leetcode-problem-dataset/data

[2] UI – https://huggingface.co/spaces/noa151/LeetCodePredictions

[3] Git – https://github.com/RachelB9913/Machine-Learning-Project