Noa Fishman – 319055430
Rachel Belokopytov – 212197248

Introduction:

After a semester of exploring various Machine Learning (ML) topics—including NLP, clustering, and more—we were assigned a final project with flexibility in structure and topic selection. To diversify our experience, we adopted 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.

We chose the Leetcode Problems dataset from Kaggle due to its relevance to technical interviews—a major focus for final-year computer science students. Given LeetCode's popularity in interview preparation, analyzing its problems through ML offered both practical and academic value.

In this project, we explore the characteristics of coding problems, predict their difficulty levels, and identify topic-related patterns. Beyond applying machine learning concepts, the project also offers insights that support real-world 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 certain topics appear more in questions of a certain difficulty level (related_topics) **(Fig. 5)**? Do certain words in the question description indicate a certain difficulty level **(Fig. 6)**? 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. 7)**? and so on. The answers to this type of questions were mostly inferable from the plots chosen to visualize the data (part 2).

In addition to these exploratory questions, we developed machine learning models to answer more complex tasks, such as:

- 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. Finally, 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.

Some of these preprocessing steps - especially cleaning and vectorizing the problem descriptions - were challenging, as they required careful text normalization and trial-and-error to capture meaningful features while avoiding noise. Additionally, multi-label encoding and handling shorthand notation in numeric columns demanded extra attention to maintain consistency across the dataset.

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

2. Data visualization:

To better understand the dataset and uncover patterns between features, we conducted an extensive visual exploration. These visualizations helped answer some of our initial questions and also guided the development of appropriate machine learning models. The insights shown are specific to this dataset and may not generalize beyond it.

The first visualizations reveal a key insight about LeetCode's problem distribution in this dataset **(Fig. 1):** medium-difficulty problems dominate the platform at 53%, followed by easy (26%) and hard (21%) problems, showing an imbalanced distribution of difficulty levels.

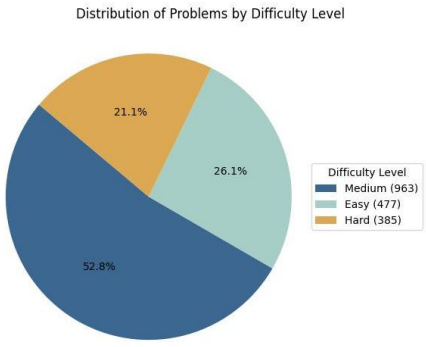


Fig. 1. Pie chart of the original difficulties distribution

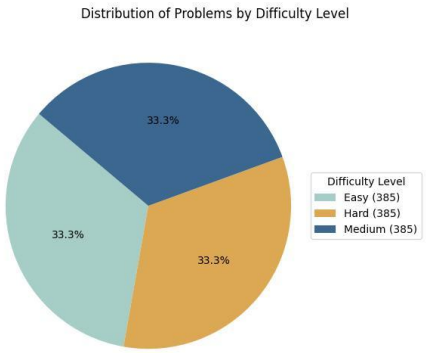


Fig.2. Pie chart of the balanced difficulties distribution

After examining the data distribution and running several initial models, we realized that the imbalance could introduce bias into our machine learning models. To mitigate this, we created a more balanced dataset (**Fig. 2**) with a total of 1,155 entries. Since the 'hard' class had the fewest questions, we sampled an equal number of questions from the other classes to match it. This balanced dataset was used throughout the rest of the project.

The line plot (**Fig. 3**) 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 supports the difficulty classifications and provides users with realistic expectations about success rates at different levels.

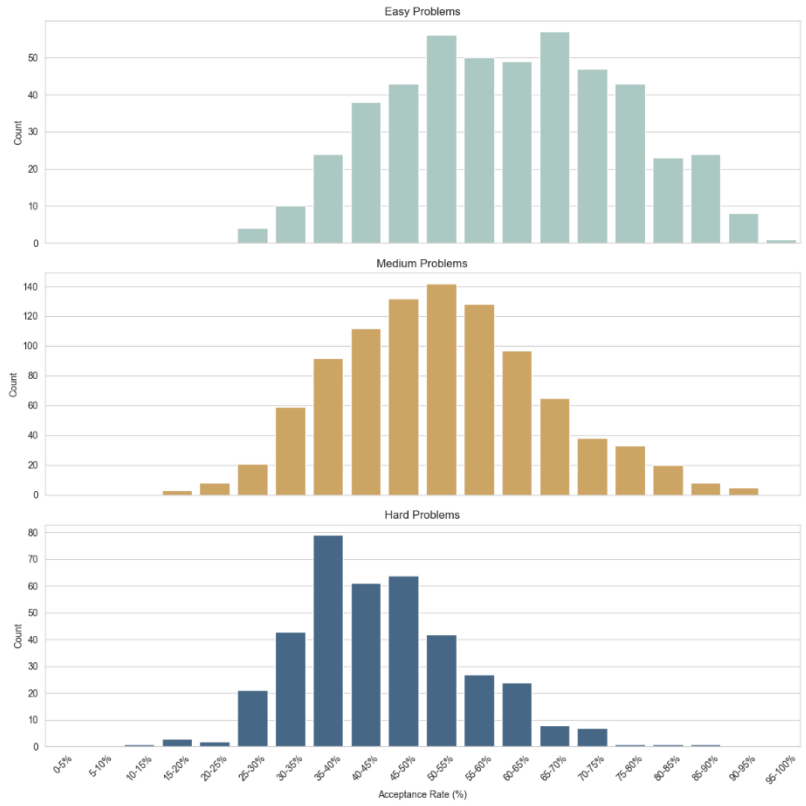


Fig. 3. Distribution of acceptance rate by difficulty level

The next visualization **(Fig. 4)** illustrates how different companies distribute their interview questions across difficulty levels. For example, Google has the largest set of problems, with a strong focus on hard and medium questions, suggesting a tougher interview process. Amazon also has many questions but emphasizes medium difficulty more than

hard. Most companies follow a general trend of medium > easy > hard questions, which can help candidates better tailor their preparation for specific companies.

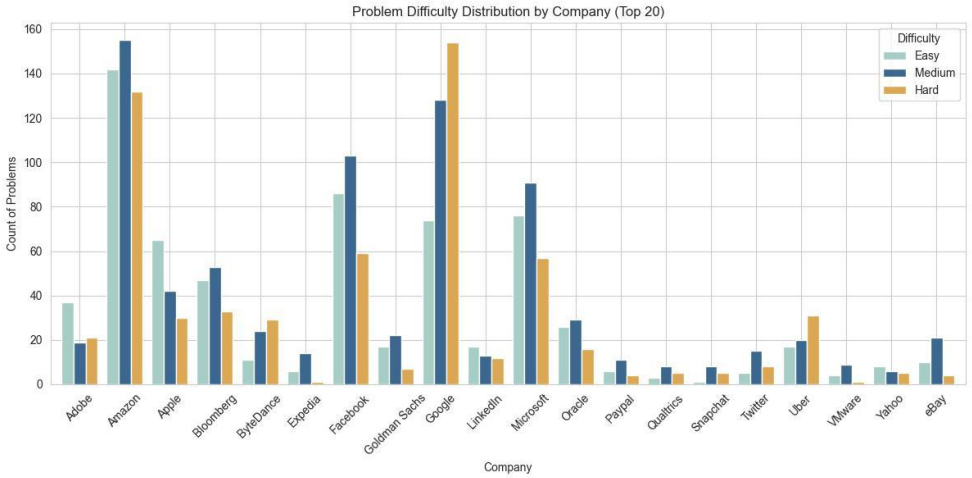


Fig. 4. Distribution of problem count by difficulty level for each company

The next bar chart **(Fig. 5)** illustrates the distribution of problem difficulties (Easy, Medium, Hard) across different topics. It shows that while some topics like 'String' and ‘Math’ 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 looking to understand which topics might require more advanced preparation.

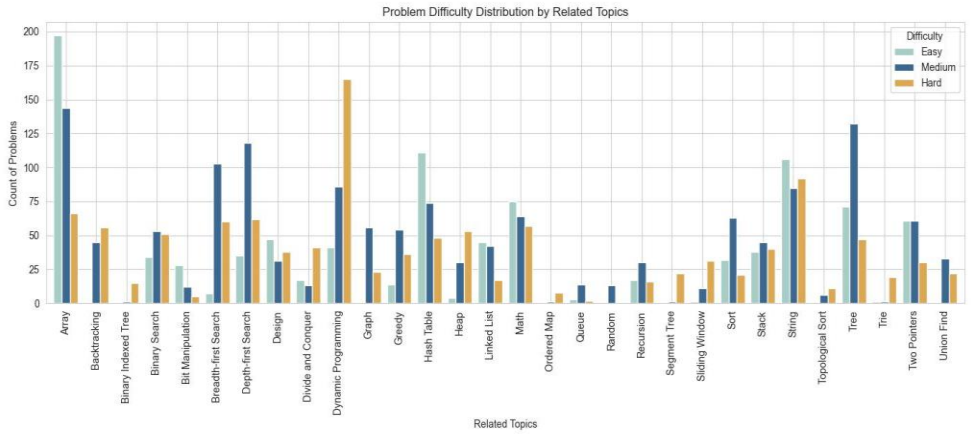


Fig. 5. Distribution of problem count by difficulty level for each related topic

The next bar chart **(Fig. 6)** shows the frequency of the most used words appearing in problem descriptions, grouped by difficulty level. Notable patterns include, among other things, certain technical terms are more dominant in harder problems (e.g., ‘graph’, ‘node’),

while medium problems show the most diverse vocabulary, and easy problems tend to use simpler, more straightforward terminology. This analysis helps clarify how problem descriptions correlate with difficulty levels and could be useful for, later, difficulty classification.

The heatmap **(Fig. 7)** visualizes the relationship between top companies and the programming topics they emphasize in their interview questions. Darker blue squares indicate a stronger focus on a specific topic within that company. The values are normalized per company, so the intensity reflects the relative importance of a topic compared to others within the same company.
For instance, companies like Amazon and Google cover a wide range of topics, while data structures such as Arrays, Strings, and Trees appear frequently across many companies. This visualization helps candidates identify which topics to prioritize when preparing for interviews at specific companies.

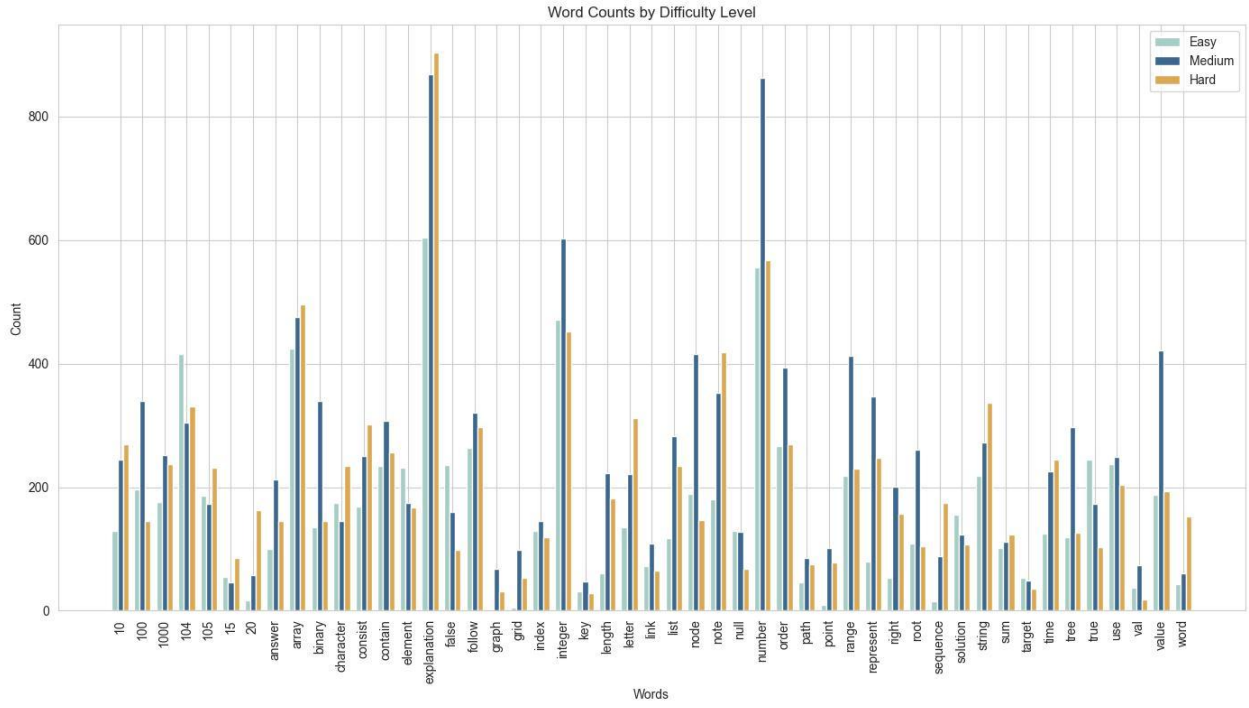


Fig. 6. Distribution of word count by difficulty level

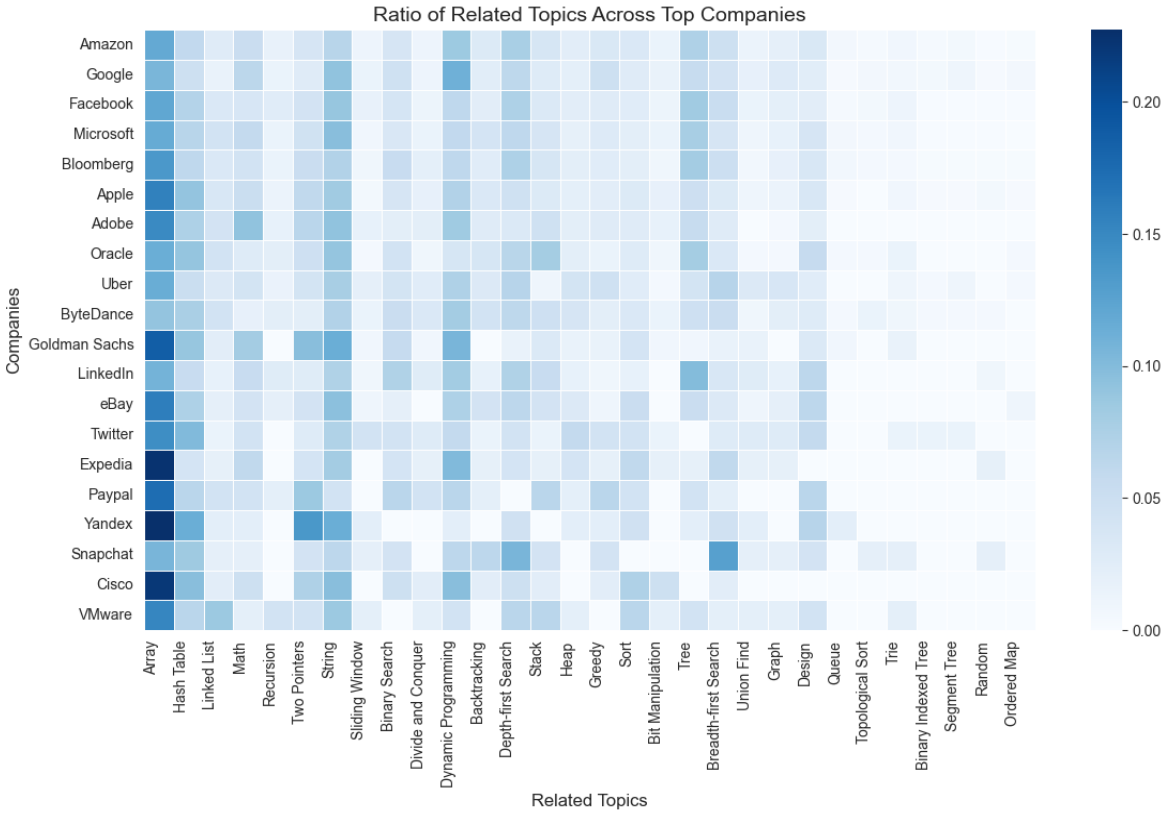


Fig. 7. Ratio of related topics across top 20 companies

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.

Since each question required a different modeling approach, we used both classification and regression techniques depending on the nature of the target variable.

Starting with the basic questions and their models:

Question 1 - Is it possible to predict the number of accepted submissions?

To predict the number of accepted submissions for a given problem, we created the accepted_submissions_regression function. This function used **linear regression**, leveraging features such as number of submissions, difficulty, discussion count, and premium status.

With the function, the data was split into training and test sets. After training, we evaluated performance using Mean Absolute Error (MAE) and Mean Squared Error (MSE). A scatter plot **(Fig. 8)** was generated to compare predicted vs. actual values, giving a visual sense of the model's accuracy.

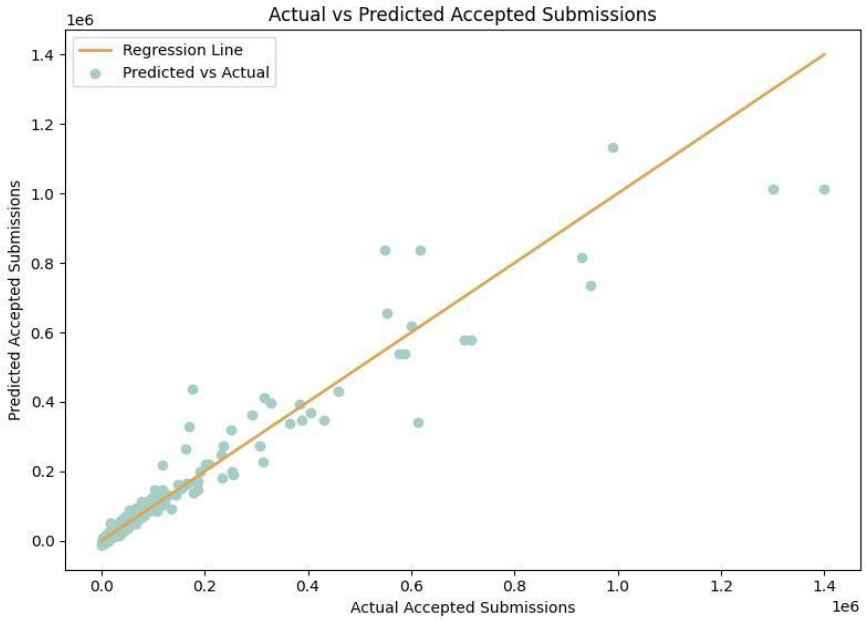


Fig. 8. Scatter plot of actual Vs. predicted accepted submissions

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** – 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** – an ensemble learning method that combines multiple decision trees to improve predictive accuracy and control overfitting. 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** – an ensemble learning method that builds models sequentially, with each new model correcting the errors made by the previous one. It fits weak learners to the residuals of the prior models.
- **XGBoost** (Extreme Gradient Boosting) – an optimized version of gradient boosting that is faster and more efficient. It includes regularization techniques to reduce overfitting.

Each model was evaluated using MAE, MSE, and RMSE. We visualized model performance using bar charts comparing these metrics (**Fig. 9**), and scatter plots showing predicted vs. actual values (**Fig. 10**). The get_best_model function helped determine the top-performing model for each task.

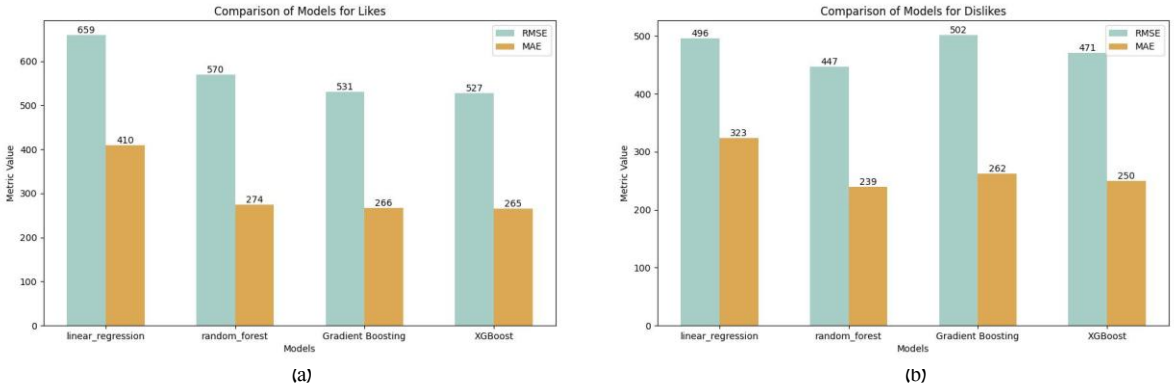


Fig. 9. Histograms showing the comparison between the models based on RMSE and MAE. (a) for the likes prediction and (b) for the dislikes

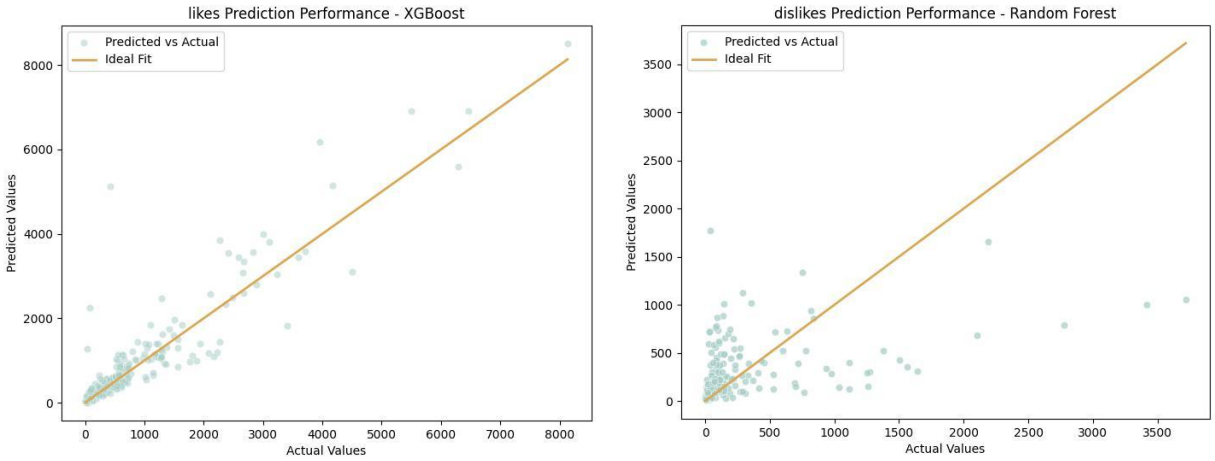


Fig. 10. Scatter plots of the actual Vs. predicted of the **best** evaluated models.
(a) for likes prediction (b) for dislikes prediction

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 problem difficulty while comparing different modeling techniques to understand their strengths and limitations. Classification was performed using both traditional machine learning models and a neural network.

The 'difficulty_classification' function applied several machine learning models using a feature set consisting of metadata such as is_premium, acceptance_rate, rating, and discuss_count, along with top words from TF-IDF and related topics. The models used were:

- **Logistic Regression** – a simple and interpretable classification model that estimates the probability of an 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 lacked the ability to capture complex interactions between features.
- **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) - 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 overfit, especially when using kernel functions to capture non-linear relationships.

- **K-Nearest Neighbors** (KNN) - a non-parametric model that classifies a sample based on the majority class of its k nearest neighbors. It does not assume any underlying distribution, but its performance depends heavily on the value of k and the distance metric. It can struggle with high-dimensional data and large datasets due to computational cost.
- **Gradient Boosting** - highly effective for structured data and capable of capturing complex patterns in the features.
- **XGBoost** - 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 our difficulty classification.

In the classification process, identifying the most influential features was important. Tree-based models like Random Forest, Gradient Boosting, and XGBoost provided built-in feature importance metrics (e.g., gain, cover, frequency). These helped highlight which features — such as is_premium, acceptance_rate, rating, discuss_count, and top TF-IDF words — were most predictive. For example, is_premium and rating were strong indicators of difficulty, reflecting the exclusivity and perceived challenge of a problem.

After training these ML models, we introduced a deep learning model to evaluate whether it could outperform them. The 'difficulty_classification_nn' function implemented a simple three-layer feedforward neural network. It included fully connected layers with Batch Normalization, ReLU activations, and Dropout to prevent overfitting. The model used the AdamW optimizer, ReduceLROnPlateau scheduler, and early stopping. Data was split into training, validation, and test sets, and the model was trained for up to 100 epochs. Loss and accuracy were tracked and visualized across epochs and per difficulty level.

Since the course focused on ML rather than DL, we kept the neural network architecture simple.

The model parameters were tuned through experimentation. We selected those that performed best based on validation metrics.

The main evaluation metric was accuracy, which measures the proportion of correctly predicted difficulty levels. Accuracy was calculated for each model, with XGBoost achieving the highest score **(Fig. 11)**.

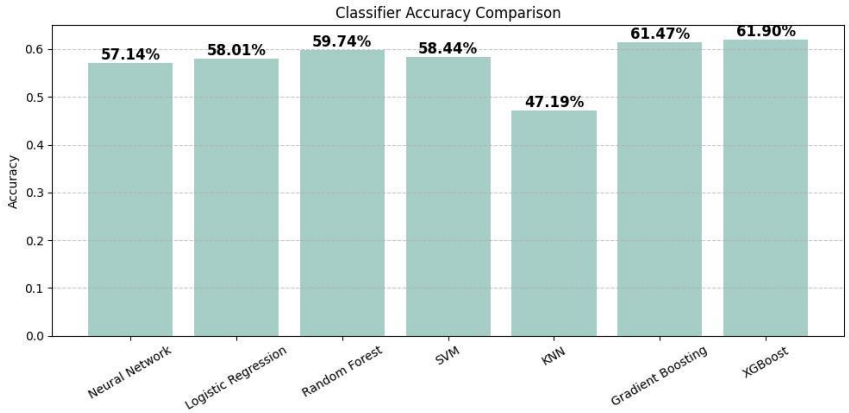


Fig. 11. Accuracy comparison of difficulty classification models

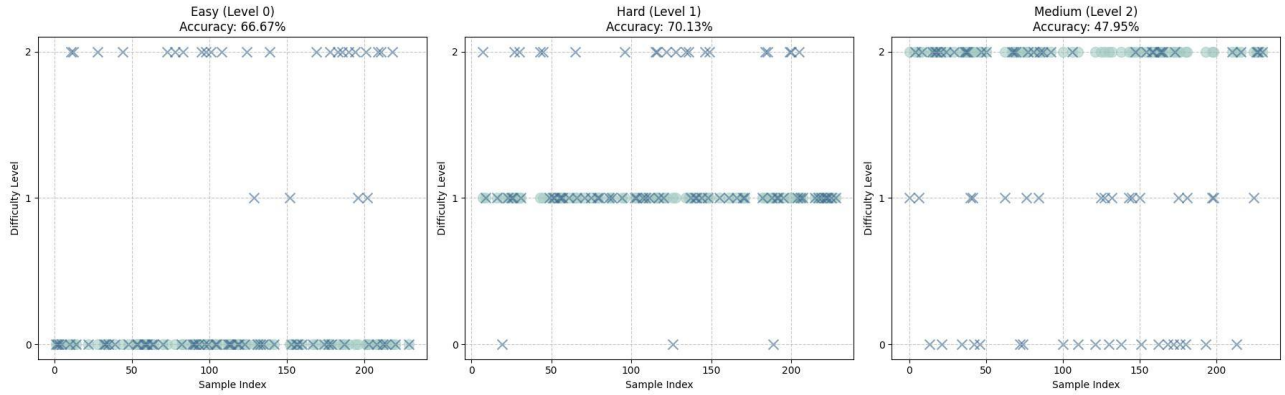


Fig. 12. Graphs that display the actual Vs. predicted labels across different sample indexes of the test set for the XGBoost model. Colored circle being the actual label and X being the predicted label. (a) for the samples with actual label 'easy' (b) for 'hard' and (c) for 'medium'

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

This task was framed as a multi-label classification problem, where each problem description could be associated with multiple topics. The related_topics_prediction function aimed to train various machine learning models, optimize thresholds, evaluate performance, and identify the best-performing model.

Models used included:

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

For SVM, Logistic Regression, and Random Forest, we used a One-vs-Rest (OvR) strategy, training a binary classifier for each 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 were determined using Stratified K-Fold Cross-Validation, where the optimal threshold for each label is averaged across folds to reduce overfitting. After optimization, predictions were 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. There are many different metrics to use for that purpose. After exploring and deep understanding the metrics, the following metrics was chosen to evaluate the model:

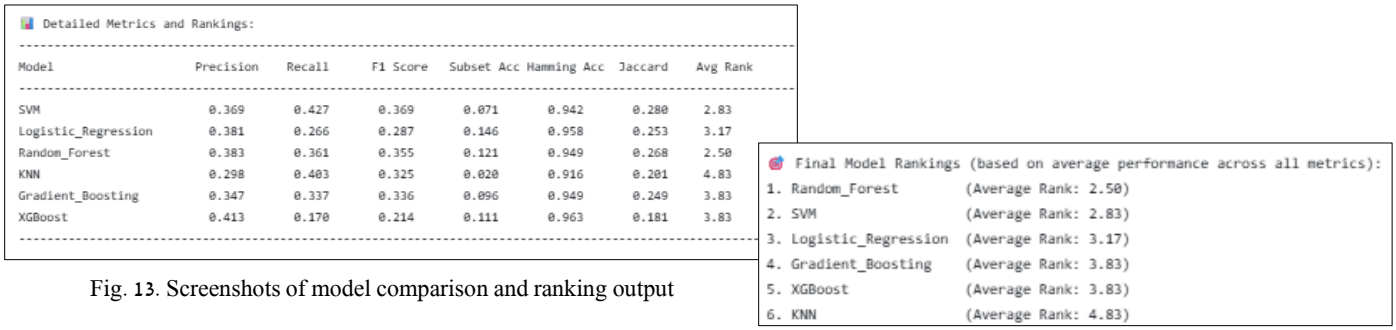
- **Weighted Precision** - averages precision across classes, weighted by the number of true instances.
- **Weighted Recall** - similarly averages recall with weighting, measuring how well the model captures true labels.
- **Weighted F1 Score** - the harmonic mean of weighted precision and recall; balances both.
- **Subset Accuracy** - percentage of exact matches between predicted and actual label sets.
- **Hamming Accuracy** - measures the fraction of correctly predicted labels across all labels.
- **Jaccard Score** - measures overlap between predicted and true labels (intersection over union).

These metrics help reduce bias toward dominant classes and provide a well-rounded view of model performance.

This question proved to be the most challenging in the project. Unlike standard classification tasks, multi-label classification required careful preprocessing, threshold

tuning, and selection of appropriate evaluation metrics. Balancing multiple labels per instance, while ensuring interpretability and generalization, was non-trivial.

Based on these metrics and a ranking method, Random Forest emerged as the most accurate model **(Fig. 13)**.



Detailed Metrics and Rankings:

Model	Precision	Recall	F1 Score	Subset Acc	Hamming Acc	Jaccard	Avg Rank
SVM	0.369	0.427	0.369	0.071	0.942	0.280	2.83
Logistic_Regression	0.381	0.266	0.287	0.146	0.958	0.253	3.17
Random_Forest	0.383	0.361	0.355	0.121	0.949	0.268	2.50
KNN	0.298	0.403	0.325	0.020	0.916	0.201	4.83
Gradient_Boosting	0.347	0.337	0.336	0.096	0.949	0.249	3.83
XGBoost	0.413	0.170	0.214	0.111	0.963	0.181	3.83

Final Model Rankings (based on average performance across all metrics):

1. Random_Forest (Average Rank: 2.50)
2. SVM (Average Rank: 2.83)
3. Logistic_Regression (Average Rank: 3.17)
4. Gradient_Boosting (Average Rank: 3.83)
5. XGBoost (Average Rank: 3.83)
6. KNN (Average Rank: 4.83)

Fig. 13. Screenshots of model comparison and ranking output

4. Discussion:

Throughout the process of building and refining our models, we encountered several key challenges and made various improvements. Developing predictive models required multiple iterations of experimentation, evaluation, and fine-tuning.

After visualizing the data, we identified a significant imbalance in the number of questions across different difficulty levels. To address this, we applied a data filtering step to create a balanced dataset, ensuring an equal number of questions for each class. However, when comparing model performance on the balanced versus imbalanced dataset, we found that the balanced dataset actually yielded worse results.

A possible explanation is that balancing the data required down-sampling overrepresented classes, which likely resulted in the loss of valuable information. This reduction in training data may have weakened the model's ability to generalize, especially if some of the removed samples contained meaningful patterns or edge cases.

We tested various machine learning models across different tasks. Regression models were used to predict numerical values such as likes, dislikes, and accepted submissions, while classification models were applied to tasks like predicting difficulty level and related topics. In general, ensemble models such as XGBoost and Random Forest consistently outperformed simpler models, reinforcing the importance of both feature selection and hyperparameter tuning.

For difficulty classification, applying TF-IDF vectorization to the problem descriptions significantly improved performance. We also explored a more experimental approach —

attempting to predict question titles from their descriptions using an LSTM model with cosine similarity loss. However, despite careful tuning, this model performed poorly due to weak text embeddings and the relatively small dataset, which limited its ability to learn patterns effectively.

While traditional machine learning models outperformed deep learning models in this project, we identified several ways to improve deep learning performance in the future. This includes:

- Designing deeper or more specialized architectures to better capture complex feature interactions
- Tuning hyperparameters more extensively
- Expanding the dataset, either through collecting more labeled examples or using text-based data augmentation
- Leveraging pre-trained transformer models (e.g., BERT) to extract richer semantic features from the problem descriptions

Overall, the project demonstrated the strength of classical ML models for structured tabular data but also highlighted the potential of deep learning methods with the right data and architecture.

5. Conclusions:

This project applied machine learning and deep learning to analyze and predict features of Leetcode problems. Ensemble models like Random Forest and XGBoost performed best, while deep learning was limited by dataset size—suggesting future improvements like data augmentation and deeper architectures.

Overall, the project provided valuable insights into applying machine learning to technical problem analysis. We started with core questions and, along the way, tackled more complex challenges that emerged.

This experience strengthened our understanding of modeling, preprocessing, and the importance of experimentation and careful evaluation in real-world ML tasks.

6. Links:

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

[2] Git – https://github.com/NoaFishman/leetcode_ml_project