Exam Question Difficulty Predictor

Intelligent Question Complexity Analysis
via Feature Engineering & XGBoost

Authors:

Rishik Chowdary Karuturi

Palwai Tejaswini

Nakkina Lakshmi Praneeth

Date: February 2026

Version: 1.0

Abstract

This report presents an end-to-end machine-learning system that automatically predicts the difficulty level of multiple-choice examination questions. The system extracts 25 quantitative features from raw question text and employs XGBoost models for both continuous difficulty-index regression and categorical classification (Easy / Medium / Hard). Comparative analysis across three modelling strategies—text-only XGBoost, linear regression, and all-feature XGBoost—is provided. The production application is deployed via Streamlit as a real-time inference tool for educators and exam administrators.

Contents

1 Introduction

1.1 Motivation

Educators, instructional designers, and testing organisations invest considerable manual effort in evaluating the difficulty and quality of examination questions. A misjudged question can skew test results and inaccurately measure student proficiency. Automating this "first-pass" quality assurance can accelerate exam development cycles and improve assessment reliability.

1.2 Objective

The **Exam Question Difficulty Predictor** is an end-to-end, production-structured machine-learning pipeline that:

1. Takes raw question text—including LaTeX and mathematical symbols—as input.
2. Extracts **25 quantitative features** spanning lexical, mathematical, and domain-specific dimensions.
3. Predicts a continuous **difficulty index** (p-value, 0.0–1.0) via regression.
4. Classifies the question into a categorical **difficulty tier**: Easy, Medium, or Hard.
5. Exposes predictions through an interactive **Streamlit web application**.

1.3 Scope

- **Domain:** English-language mathematics examination questions.
- **Dataset:** 50,000 preprocessed exam questions with associated metadata.
- **Models:** XGBoost (primary), Linear Regression (baseline).
- **Deployment:** Local Streamlit application with serialised model artefacts.

2 Dataset

2.1 Source & Scale

The dataset comprises **50,000 examination questions** sourced from educational repositories. Each record contains:

- Question text (may include LaTeX)
- Four multiple-choice answer options (A–D)
- Continuous difficulty p-value (0.0 = hardest, 1.0 = easiest)
- Categorical difficulty label (Easy / Medium / Hard)
- Subject metadata, misconception counts, and construct frequency

- Post-administration statistics (response time, discrimination index, IRT parameters)

2.2 Data Cleaning Pipeline

A dedicated preprocessing script (`clean_dataset.py`) applies the following transformations:

Step	Description
NULL Standardisation	Unifies heterogeneous null markers (`N/A`, `none`, `?`, `-`) into `NaN`.
Whitespace Stripping	Removes leading/trailing whitespace from all string columns.
Deduplication	Identifies and removes identical rows.
Casing Normalisation	Standardises categorical fields to title case.
Domain Constraints	Enforces p-value and percentage fields into $[0, 1]$ or $[0, 100]$.
Outlier Capping	Applies inter-quartile range (IQR) capping to numeric distributions.
Type Coercion	Forces correct `dtypes` (float, int, category).

2.3 Class Distribution

The dataset exhibits significant class imbalance:

Class	Approx. Count	Proportion
Easy	∼40,000	∼80%
Medium	∼7,500	∼15%
Hard	∼2,500	∼5%

This imbalance is a critical consideration discussed in Section 6.

3 Feature Engineering

All features are extracted by the `feature_extractor.py` module. The 25 features fall into six categories:

3.1 Lexical Features (4)

Feature	Description
`text_length`	Total character count of the question.
`word_count`	Total word count.
`sentence_count`	Sentence count (split on `.!?`).
`avg_word_length`	Mean number of characters per word.

3.2 LaTeX & Mathematical Features (5)

Feature	Description
`latex_command_count`	Count of detected LaTeX commands.
`has_latex`	Binary indicator (0/1).
`latex_density`	$\frac{\texttt{latex\_command\_count}}{\texttt{word\_count}}$.
`math_operator_count`	Count of operators ($+, -, \times, \div, =$, etc.).
`number_count`	Count of numeric literals.

3.3 Vocabulary & Complexity (2)

Feature	Description
`vocab_richness`	$\frac{\lvert\text{unique words}\rvert}{\lvert\text{total words}\rvert}$.
`text_complexity_score`	Weighted composite: $0.3 \cdot \overline{w} + 0.2 \cdot L + 0.15 \cdot M + 0.1 \cdot N + 2.0 \cdot \frac{L+M}{W}$.

Where $\overline{w}$ = avg word length, L = LaTeX count, M = math ops, N = numbers, W = word count.

3.4 Answer-Option Features (5)

Feature	Description
`answer_{a,b,c,d}_length`	Character length of each option.
`avg_answer_length`	Mean across four options.
`answer_length_variance`	Variance across four options.

3.5 Domain-Detection Features (4)

Binary indicators set to 1 when any keyword from the corresponding dictionary is detected:

Feature	Keyword Examples
has_advanced_terms	calculus, derivative, integral, eigenvalue, matrix
has_algebra_terms	equation, polynomial, quadratic, factorise, inequality
has_geometry_terms	angle, triangle, circle, hypotenuse, Pythagoras
has_stats_terms	mean, median, probability, histogram, standard deviation

3.6 Metadata Features (5)

Feature	Description
num_misconceptions	Count of associated student misconceptions.
has_misconception	Binary (0/1).
subject_difficulty_tier	Ordinal tier (1–5) provided by the user.
construct_frequency	Frequency of the tested construct in the syllabus.

Crucially, all 25 TEXT_FEATURES are available *before* the exam is administered, avoiding data leakage.

4 Model Architecture

Three modelling strategies were explored:

4.1 Model A — XGBoost (Text-Only Features)

An XGBoost ensemble trained exclusively on the 25 text-derived features. Two submodels are trained:

- **Regressor** (xgb.XGBRegressor): Predicts the continuous p-value.
- **Classifier** (xgb.XGBClassifier): Predicts the categorical label.

Hyperparameters:

Parameter	Value
n_estimators	500
max_depth	6
learning_rate	0.05
subsample	0.8
colsample_bytree	0.8
objective (clf)	multi:softprob
eval_metric (clf)	mlogloss
objective (reg)	reg:squarederror

4.2 Model B — Linear Regression (Text-Only Features)

A standard `sklearn.linear_model.LinearRegression` trained on the same 25 features as a baseline comparator. No classification model is produced.

4.3 Model C — XGBoost (All Features)

An XGBoost ensemble trained on all available features, including **post-administration** statistics:

- All 25 text features from Model A
- Average response time, response time standard deviation
- Discrimination index
- IRT difficulty, IRT discrimination, IRT guessing parameters
- Student attempt statistics

Important: Post-admin features are only available *after* an exam has been administered. This model demonstrates the upper-bound performance but entails target leakage for cold-start deployment.

5 Training & Evaluation

5.1 Data Split

- **Train:** 80% of the dataset
- **Test:** 20% of the dataset
- Stratified split on `difficulty_label` to preserve class proportions.

5.2 Cross-Validation

5-fold stratified cross-validation is applied to the classification models using `sklearn.model_selection.` with accuracy scoring.

5.3 Evaluation Metrics

5.3.1 Regression

- **MAE** (Mean Absolute Error): $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
- **RMSE** (Root Mean Squared Error): $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
- R^2 (Coefficient of Determination): $1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$

5.3.2 Classification

- Accuracy, Precision (weighted), Recall (weighted), F1-Score (weighted)
- Per-class precision/recall via `classification_report`
- Confusion matrix visualisation

6 Results & Model Comparison

6.1 Regression Performance

Model	MAE	RMSE	R^2
XGBoost (Text-Only)	0.0772	0.0983	0.5693
Linear Regression (Text)	0.0760	0.0949	0.5811
XGBoost (All Features)	**0.0162**	**0.0227**	**0.9761**

6.2 Classification Performance

Model	Accuracy	Precision[w]	Recall[w]	F1[w]
XGBoost (Text-Only)	83.34%	0.83	0.83	0.80
XGBoost (All Features)	**95.34%**	**0.95**	**0.95**	**0.95**

[w]Weighted average across classes.

6.3 Per-Class Analysis (Text-Only XGBoost)

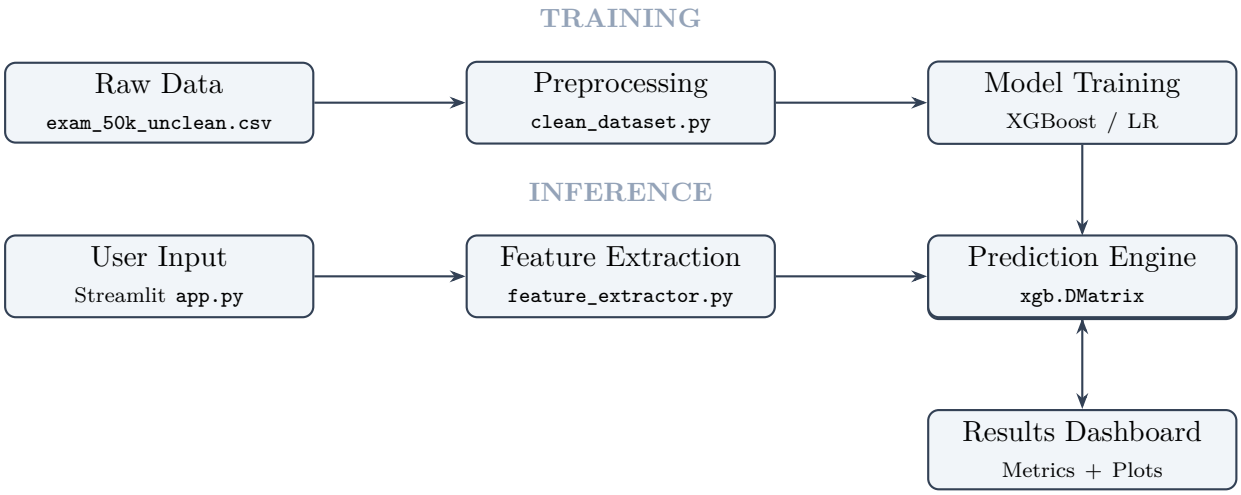
Class	Precision	Recall	F1	Support
Easy	0.85	0.97	0.91	∼8,000
Medium	0.63	0.38	0.47	∼1,500
Hard	0.56	0.11	0.18	∼500

6.4 Key Observations

1. **All-Features XGBoost dominates** with $R^2 = 0.976$ and ~95% accuracy. However, post-admin features constitute effective target leakage for pre-exam prediction tasks.
2. **Linear Regression marginally outperforms text-only XGBoost** on regression (R^2: 0.581 vs. 0.569), suggesting the text–difficulty relationship is approximately linear and that XGBoost may slightly overfit correlated features.
3. **Class imbalance severely degrades minority-class recall.** The text-only classifier achieves 97% recall for Easy but only 11% for Hard. The headline 83% accuracy is misleading—a model predicting "Easy" for every question would achieve ~80%.
4. **Multicollinearity** among lexical features (e.g., `text_length` ↔ `word_count`) can distort feature importance rankings, though it does not affect prediction quality.

7 System Architecture

7.1 High-Level Pipeline



7.2 Repository Structure

```
genai-project/
  app.py                    # Streamlit web application
  feature_extractor.py      # 25-feature extraction module
  requirements.txt          # Python dependencies
  model/
    xg_model.pkl            # LabelEncoder pipeline
  files/
    xgb_reg_model_A.json    # XGBoost Regressor (text-only)
    xgb_clf_model_A.json    # XGBoost Classifier (text-only)
  notebooks/
    comparer/               # Model comparison notebook
    all_features_xgboost/   # All-features XGBoost notebook
```

```
    linear_regression_text/ # Linear Regression notebook
  raw_dataset/
    exam_dataset_50k_unclean.csv
  cleaned_dataset/
    exam_dataset_50k_cleaned.csv
```

8 Application Interface

The production application (`app.py`) is built with **Streamlit** and provides two pages:

8.1 Analysis Tool

1. The user inputs question text, four answer options, subject tier (1–5), misconception count, and construct frequency.
2. `feature_extractor.py` computes all 25 features.
3. An `xgb.DMatrix` is constructed and fed through both the regressor and classifier boosters.
4. A **heuristic bias adjustment** nudges the classifier's softmax probabilities based on the user-provided subject tier:
   - Tiers 1–2 → slight Easy weight
   - Tier 3 → slight Medium weight
   - Tiers 4–5 → increased Hard weight
5. Results are displayed as: predicted class, difficulty p-value, model confidence, class-probability bar chart, and a feature-detail table.

8.2 About the Model

A documentation page describing the architecture, feature extraction methodology, performance metrics, and known limitations.

8.3 Design

The interface follows a **dark-mode, slate-palette** design with custom CSS:

- Background: `#0f172a` (Slate-900)
- Metric cards with `#1e293b` borders
- Inter typeface for readability
- Monospace feature values (`JetBrains Mono`)

9 Post-Prediction Heuristic Adjustment

Because the text-only model has limited contextual awareness, a **lightweight heuristic layer** adjusts classifier probabilities after inference:

$$P'(c) = \frac{P(c) \cdot w_c}{\sum_j P(j) \cdot w_j} \tag{1}$$

Where w_c are tier-dependent weight multipliers:

Condition	w_{Easy}	w_{Medium}	w_{Hard}
Short & simple ($W < 4$, complexity < 2)	5.0	1.0	1.0
Tier ≤ 2, complexity < 6	1.5	1.0	1.0
Tier $= 3$, complexity < 6	1.0	1.3	1.0
Tier $= 4$, complexity < 6	1.0	1.2	1.5
Tier $= 5$, complexity < 6	1.0	1.0	4.0
Otherwise (complex text)	1.0	1.0	1.0

This ensures subject-tier metadata is incorporated without overriding strong textual signals.

10 Limitations & Known Issues

1. **Text-Only Analysis:** The model cannot parse images, diagrams, or graphs. Geometry questions reliant on visual context may be misclassified.
2. **Class Imbalance:** The severe Easy-class dominance (∼80%) causes the classifier to under-predict Hard and Medium classes. Mitigation strategies (SMOTE, class weights) have not yet been applied.
3. **Multicollinearity:** Features like `text_length` and `word_count` are highly correlated, potentially distorting feature-importance interpretation. SHAP analysis is recommended.
4. **Synthetic Data Bias:** The training dataset contains synthetic variations. Real-world questions with unusual formatting may yield lower-confidence predictions.
5. **Language Support:** Feature extraction is optimised for English-language mathematics. Non-English text will produce inaccurate features.
6. **Potential Target Leakage:** The `subject_difficulty_tier` feature should be verified to ensure it is not derived from the target variable itself.

11 Future Work

11.1 Short-Term Improvements

- Apply **class-weight balancing** or **SMOTE** to address the imbalance problem.
- Use **SHAP values** for feature-importance interpretation instead of raw XGBoost importances.
- Implement **hyperparameter tuning** via Bayesian optimisation (Optuna/Hyperopt).
- Drop highly correlated features to reduce multicollinearity.

11.2 Milestone 2 — Agentic AI Framework

- **LLM Reasoning:** Use large language models (Gemini / GPT) to solve questions step-by-step and measure *conceptual* complexity rather than lexical proxies.
- **RAG for Curriculum Alignment:** Retrieval-Augmented Generation to compare questions against educational standards.
- **Multi-Modal Processing:** Vision-Language Models for diagram- and graph-dependent questions.
- **Iterative Feedback Loops:** Agentic workflows simulating student failure modes to refine difficulty estimates.

12 Conclusion

This project demonstrates that a combination of lightweight NLP feature engineering and gradient-boosted tree models can provide a meaningful automated estimate of examination question difficulty. The text-only XGBoost model achieves an R^2 of $\sim$0.57 and $\sim$83% classification accuracy, while the all-features variant reaches $R^2 = 0.976$ when post-administration data is available.

The Streamlit application provides an accessible interface for educators to obtain instant difficulty assessments, and the modular architecture (separate feature extractor, serialised models, heuristic layer) enables straightforward future extension.

Key next steps include addressing classimbalance, integrating SHAP-based interpretability, and evolving toward an agentic LLM-based framework for deeper conceptual difficulty analysis.

A Technology Stack

Component	Technology
Language	Python 3.10+
ML Framework	XGBoost $\geq$ 2.0, scikit-learn $\geq$ 1.3
Data Processing	Pandas $\geq$ 2.0, NumPy $\geq$ 1.24
Visualisation	Matplotlib $\geq$ 3.7, Seaborn $\geq$ 0.12
Web Framework	Streamlit $\geq$ 1.30
Serialisation	Joblib $\geq$ 1.3, XGBoost JSON

B Feature Vector Specification

The complete ordered feature vector $\mathbf{x} \in \mathbb{R}^{25}$ used by the deployed models:

#	Feature Name	Type
1	`text_length`	Integer
2	`word_count`	Integer
3	`sentence_count`	Integer
4	`avg_word_length`	Float
5	`latex_command_count`	Integer
6	`has_latex`	Binary
7	`latex_density`	Float
8	`math_operator_count`	Integer
9	`number_count`	Integer
10	`vocab_richness`	Float
11	`text_complexity_score`	Float
12	`answer_a_length`	Integer
13	`answer_b_length`	Integer
14	`answer_c_length`	Integer
15	`answer_d_length`	Integer
16	`avg_answer_length`	Float
17	`answer_length_variance`	Float
18	`has_advanced_terms`	Binary
19	`has_algebra_terms`	Binary
20	`has_geometry_terms`	Binary
21	`has_stats_terms`	Binary
22	`num_misconceptions`	Integer
23	`has_misconception`	Binary
24	`subject_difficulty_tier`	Ordinal

#	Feature Name	Type
25	`construct_frequency`	Integer