Question related to implementation of the software development recommender model

1. What are the key reasons for dividing your recommender model into two phases?

It is a strategic approach that offers several advantages:

1. The choice between Waterfall and Agile is foundational, as these methodologies have fundamentally different principles. A two-phase model ensures that the classification starts at the broadest level before diving into finer details.

2. If a single model attempted to classify all methodologies at once, it would need to learn complex relationships across fundamentally different development paradigms. By splitting the task, each classifier focuses on a smaller, more relevant decision space, improving performance.

3. A single model might struggle with classifying methodologies that share overlapping features (e.g., Scrum and Extreme Programming). By first categorizing projects at a high level, secondary classifiers can work within a more homogeneous subset, improving accuracy.

2. How did you preprocess and structure your dataset for both phases? - refer code base

1. Data Collection and Structure - Dataset Format

2. Preprocessing Steps - Data Splitting

3. Dataset Structuring for Two-Phase Classification

4. Feature Engineering - Derived Features (F1 + F2 + ... Fn)

3. Why did you choose these specific machine learning algorithms? Have you compared them with other models?

You already tested multiple classifiers, and Random Forest Performed Best (99.95% Accuracy) :

- Handles Non-Linearity: Unlike Logistic Regression, which assumes linear separability, Random Forest captures complex decision boundaries.

- Reduces Overfitting: Unlike a single Decision Tree, Random Forest combines multiple trees, reducing variance and improving generalization.

- Feature Importance: Automatically identifies the most relevant features, improving model efficiency.

- Robust to Noise and Missing Data: Works well with real-world datasets that may have noise or missing values.

4. What feature selection techniques did you use, and why?

Considering all features equally important and not using any feature selection techniques, the approach ensures that the model captures the full scope of available information. (DT and RF)

5. Can you explain the performance of each algorithm and which one performed best?

Google on LogisticRegression, KNeighborsClassifier, GaussianNB, DecisionTreeClassifier, and RandomForestClassifier. RandomForestClassifier performed best with 99.95% Accuracy.

6. What evaluation metrics did you use to compare these models (e.g., accuracy, precision, recall, F1-score, ROC curve)?

Accuracy\_score

Loss

Cohen\_kappa\_score

Classification\_report:

precision, recall, f1-score

Confusion\_matrix

7. Did you perform hyperparameter tuning? If so, what methods (e.g., GridSearchCV, RandomizedSearchCV) did you use?

No

8. What challenges did you face in training the models, and how did you overcome them?

1. The model become biased toward the majority class.

2. Overlapping Classes in both methodology

3. Selecting the Best Model for Deployment - Evaluated models based on precision, recall, F1-score, and confusion matrix rather than just accuracy.

9. How does your model determine which software development approach to recommend?

Software Development Recommender Model follows a two-phase classification process to determine the most suitable software development approach based on input features.

- Phase 1: Primary Classifier (Waterfall vs. Agile) - The first step is to classify whether the given software development project aligns with the Waterfall or Agile methodology.

How it Works:

- Input Features (Project Requirements & Constraints) - Questions or features related to project

- Classification using Multi-Model Voting - Models like Logistic Regression, KNN, Naïve Bayes, Decision Tree, and Random Forest predict the class.

- Primary Output - The model classifies the project as either Waterfall or Agile.

- Phase 2: Secondary Classifier (Specific Agile/Waterfall Methodology) - After determining the primary approach, a second classifier predicts the specific software development methodology:

How it Works:

- If Waterfall: The model sub-classification Waterfall as the recommended approach

- If Agile: The model sub-classification Agile as the recommended approach

- Final Output: The model recommends a methodology based on feature analysis.

10. What factors influence the recommendation of a specific development methodology in Phase II?

- Provides a structured decision-making process for project managers.

11. Did you incorporate domain expertise or heuristic rules into your recommendation system?

- Yes, incorporating domain expertise and heuristic rules is crucial for improving the accuracy and reliability of your Software Development Recommender Model.

- Help us to get features or questions to ask

12. How do you ensure that the recommendations align with real-world software engineering practices?

- Data-Driven Approach (Real-World Project Data / model is trained on real-world project characteristics)

- Incorporating Software Engineering Principles - ex. If a project has strict regulatory requirements, recommend Waterfall

- Validation Against Industry Case Studies - The model's recommendations can be tested against real-world case studies (we won't do that)

13. How did you optimize your model to improve accuracy and efficiency?

- Its ML models based on formulas, so Model Selection & Hyperparameter Tuning

- GridSearchCV & RandomizedSearchCV were used to optimize:

- n\_estimators (number of trees)

- max\_depth (tree depth)

- min\_samples\_split (minimum samples to split a node)

- min\_samples\_leaf (minimum samples per leaf)

- ex.

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import GridSearchCV

param\_grid = {

'n\_estimators': [100, 200, 300],

'max\_depth': [10, 20, 30],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

rf = RandomForestClassifier()

grid\_search = GridSearchCV(rf, param\_grid, cv=5, scoring='accuracy')

grid\_search.fit(X\_train, y\_train)

best\_model = grid\_search.best\_estimator\_

\*\*\* No need \*\*\*

- Feature Engineering & Encoding - Handling Categorical Data, Label Encoding & One-Hot Encoding

- Feature Scaling - MinMaxScaler & StandardScaler

- Data Balancing & Augmentation

- Cross-Validation & Performance Metrics

- K-Fold Cross Validation (k=10) to prevent overfitting.

Metrics Used:

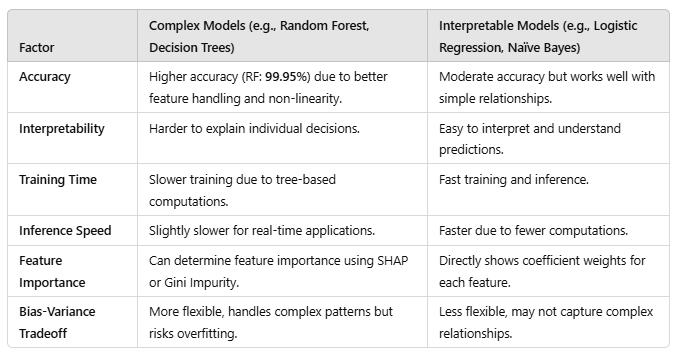
Accuracy → Measures overall correctness.

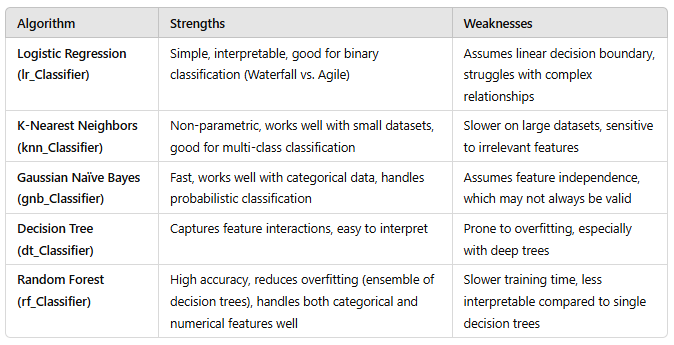
Precision & Recall → Handles class imbalance.

F1-Score → Balances precision and recall.

Confusion Matrix → Evaluates misclassifications.

14. What trade-offs did you consider between model complexity and interpretability?





15. Did you experiment with deep learning models or ensemble techniques beyond Random Forest?

- No

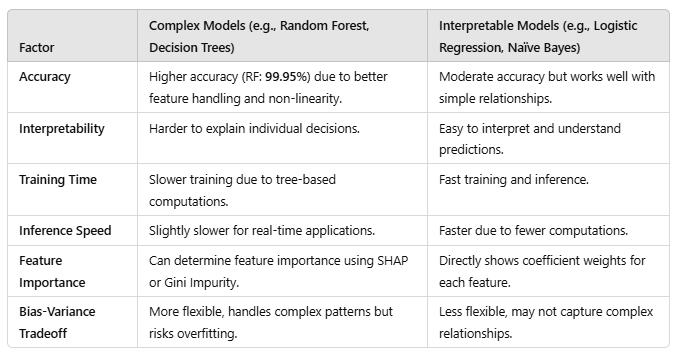
16. How does your system handle noisy or incomplete data?

- We dont have that

17. How can your recommender model be integrated into a real-world software development environment?

- Built a Flask web app, the model can be deployed as a REST API, which can be integrated into various real-world software development environments.

18. What are the computational requirements for deploying your system?



19. Have you tested the model in an industry setting or with real-world software projects?

- Yes

20. How scalable is your solution for different domains or industries?

- it can be extended to other project-based industries by modifying input features.

- it can be deployed on cloud platforms (AWS, GCP, Azure) for horizontal scaling.

- API-based deployment allows integration with different enterprise applications across domains.

21. How does your recommender system compare with existing models or frameworks in software engineering?

- Eliminates human bias and inconsistencies in decision-making.

- Provides data-driven recommendations rather than relying on experience alone.

- Can be integrated into project management tools for real-time suggestions.

22. What are the limitations of your current implementation?

- Lack of Feature Selection - Some features may be irrelevant or redundant, leading to longer training times and potential noise in predictions.

- Dataset Bias & Generalization Issues

- New or hybrid methodologies may not be recognized properly.

- Currently, there’s no way for users to provide feedback on recommendations. The model cannot learn from incorrect predictions.

23. How can your work be extended or improved in the future?

- solve point 22

24. Would incorporating reinforcement learning or deep learning improve the recommendation accuracy?

- Yes

