Table of Contents

Machine Learning Project Documentation

Model Refinement

1. Overview

The **model refinement phase** focuses on improving model performance by addressing key issues identified in the initial evaluation. This involves fine-tuning hyperparameters, selecting appropriate algorithms, and implementing techniques like **grid search and ensemble learning** to enhance predictive accuracy. Additionally, measures were taken to **identify and mitigate leakage**, ensuring the model generalizes effectively to unseen data

2. Model Evaluation

Initial Model Performance

- The **Decision Tree model** was straightforward but showed signs of **overfitting**.
- The **Random Forest model** exhibited more stability but required tuning to optimize its depth and sample splits.
- **Key metrics** used:
- **Accuracy**: Evaluated classification performance.
- **Cross-validation scores**: Ensured the model performed consistently across different data splits.
- **Confusion Matrix & AUC-ROC**: Highlighted class imbalances and areas for improvement.

Areas for Improvement

- Feature leakage checks were performed to prevent data leaks between train-test splits.
- Hyperparameter tuning was necessary to optimize **tree depth, number of estimators**, and **minimum sample splits**.
- Supervised learning techniques, including **GridSearchCV**, were introduced for fine-tuning model behavior

3. Refinement Techniques

Key Adjustments

1. **Hyperparameter Tuning**
    - **GridSearchCV** was used to systematically search for the best parameters across multiple configurations.
    - Introduced **Stratified K-Fold cross-validation** to ensure robust performance metrics.
2. **Ensemble Learning**

    - The use of **Random Forest**, an ensemble method, improved generalization.
    - Multiple decision trees were trained in parallel, reducing variance.
3. **Feature Engineering**
    - Transformation techniques like **encoding, scaling**, and **log transformations** were applied.
    - Automated preprocessing was embedded into the pipeline to streamline feature processing

4. **Leakage Prevention**

- Ensured that feature processing was **only applied to the training set** before test splits.
- Verified that **target variables** weren't leaked into feature selections

4. Hyperparameter Tuning

GridSearchCV for Random Forest

- **Parameters tuned**:

'classifier__n_estimators': [100, 200]

'classifier__max_depth': [None, 10, 20]

'classifier__min_samples_split': [2, 5]

Impact:

- Increased **generalization** and stability across different test samples.
- Prevented overfitting by restricting depth and minimum sample splits.
- Balanced computational efficiency with model accuracy.

Decision Tree Adjustments

- **Standardized preprocessing within the pipeline** to optimize feature transformation.
- Applied regularization by setting constraints on tree depth and minimum samples per split.

5. Cross-Validation

Changes in Cross-Validation Strategy

During model refinement, **Stratified K-Fold cross-validation** was implemented to ensure balanced class representation across multiple splits. This approach was chosen to address

potential **class imbalance** within the dataset and to reduce the variance in model performance estimates.

Reasoning Behind Changes

- **Standard K-Fold Cross-Validation** does not guarantee equal distribution of class labels across folds, potentially leading to skewed training sets.
- **Stratified K-Fold** ensures that each fold maintains the same class proportions as the original dataset, leading to more **robust generalization metrics**.
- By using **5 folds**, the model gets trained and validated on different subsets, reducing overfitting and improving predictive stability.

Additionally, **GridSearchCV** was used with **Stratified K-Fold**, ensuring hyperparameter tuning was performed on well-balanced data splits.

6. Feature Selection

Leakage Detection & Feature Cleaning

- A dedicated process for **feature leakage identification** was implemented to prevent target-related information from unintentionally influencing model training.
- Features containing direct indicators of the target variable—such as **scores, grades, or recommendations**—were systematically **removed**.
- The logic for detecting potential leakage features was embedded into the preprocessing pipeline using `str.contains()` to dynamically identify features that might lead to **data contamination**.

Feature Selection Methods

1. **Manual Feature Inspection**
    - Leakage-prone columns such as **Exam Questions, YouTube recommendations, and personalized materials** were removed.
    - Additional checks for indirect leakage were performed by reviewing column names and correlations.

Categorical & Numeric Transformation

- **One-Hot Encoding** was applied to categorical features to standardize representation.
- **Median imputation** was used for numeric features to handle missing data while preserving statistical integrity.
- The **ColumnTransformer** pipeline streamlined preprocessing by ensuring feature transformations were consistent across training and validation sets.

Impact on Model Performance

- **Eliminating leakage improved generalization** and prevented falsely inflated accuracy scores.
- **Balanced feature selection** ensured meaningful predictors were retained, reducing overfitting.
- **Standardized preprocessing** improved training stability, allowing hyperparameter tuning to focus on optimizing true predictive potential rather than compensating for noise.

Test Submission

1. Overview

The **test submission phase** is critical for assessing how well the trained model generalizes to unseen data before deployment. During this phase, the trained **Decision Tree** and **Random Forest** models were applied to a **held-out test dataset**, allowing for a thorough evaluation of performance metrics such as **accuracy, balanced accuracy, F1-score**, and **overfitting analysis**. Additionally, feature importance was analyzed to understand key predictors contributing to classification decisions.

2. Data Preparation for Testing

To ensure **fair and unbiased evaluation**, the following steps were taken:

- **Stratified Train-Test Split**: The dataset was split using `train_test_split()` with **stratification** to preserve class distributions

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

- **Feature Preprocessing**: The preprocessing pipeline (`ColumnTransformer`) applied transformations such as:
  - **Median imputation** for missing numerical values.
  - **One-Hot Encoding** for categorical features.
- **Leakage Prevention**: Checks for **leaked features** (e.g., exam scores, recommendations) ensured training and testing remained independent.

Great! Based on the content of your uploaded notebook, here are well-structured responses for your report:

3. Model Application

After preparing the dataset with cleaned, encoded, and engineered features, we applied machine learning models to predict student ratings of different learning materials. The main input features used were:

- `Age_Encoded`
- `Internet access_Encoded`
- `Avg hrs on Digtal Entmnt_Encoded`

For each educational material (e.g., *Royal Math*, *YouTube: Blackpen Redpen*), we split the dataset into training and testing sets (80-20 split) and trained a **Decision Tree Regressor** model. Later, we optimized this model using **GridSearchCV** and also experimented with a **Random Forest Regressor** for better performance.

Code snippet:

```
X = df[['Age_Encoded', 'Internet access_Encoded','Avg hrs on Digtal Entmnt_Encoded']]

y = df[material]


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


model = DecisionTreeRegressor(max_depth=5, random_state=42)

model.fit(X_train, y_train)


y_pred = model.predict(X_test)
```

4. Test Metrics

To evaluate model performance, we used several regression metrics:

- **Mean Absolute Error (MAE)**

- **Mean Squared Error (MSE)**
- **Root Mean Squared Error (RMSE)**
- **R² Score**

These metrics were calculated for each learning material model. The R² scores helped identify how well our features explained the variance in students' ratings.

Example output:

Mean Absolute Error (MAE): 0.63

Mean Squared Error (MSE): 0.73

Root Mean Squared Error (RMSE): 0.85

R² Score: 0.65

Code snippet:

```
def evaluate_regression_model(y_test, y_pred):
    print("Regression Model Evaluation:")
    print(f"MAE: {mean_absolute_error(y_test, y_pred):.2f}")
    print(f"MSE: {mean_squared_error(y_test, y_pred):.2f}")
    print(f"RMSE: {mean_squared_error(y_test, y_pred, squared=False):.2f}")
    print(f"R² Score: {r2_score(y_test, y_pred):.2f}")
```

The Random Forest Regressor showed slightly better generalization and lower error compared to the Decision Tree model.

5. Model Deployment

While this project did not involve full production deployment, the following deployment-adjacent steps were taken:

- **Standardization of Features** using `StandardScaler` for better model performance.

- **Feature Normalization** for model interpretability and visualization.
- **Model Tuning** using `GridSearchCV` for optimal hyperparameters.
- **Preparation for integration**: The final models are trained and can be exported (e.g., via `joblib` or `pickle`) for integration into an educational recommendation system in the future.

6. Code Implementation

Below is a summarized snippet showing model refinement and prediction:

```
# GridSearchCV for hyperparameter tuning
param_grid = {
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

grid_search = GridSearchCV(DecisionTreeRegressor(), param_grid, cv=5, scoring='r2')
grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
evaluate_regression_model(y_test, y_pred)

# Random Forest alternative
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
```

evaluate_regression_model(y_test, y_pred_rf)

Each block is followed by evaluation of the prediction accuracy and error rates.

Conclusion

This project applied regression models to predict student preferences for various digital learning resources using demographic and behavioral data. Key insights include:

- Models performed reasonably well with basic features like age, internet access, and entertainment hours.
- Random Forest outperformed Decision Tree in generalization.
- Feature engineering (e.g., `Entertainment_Age_Interaction`) improved model insight.
- Data preprocessing and visualization enabled better understanding and validation of patterns.

Challenges Encountered:

- Some missing or inconsistent data required careful cleaning.
- Encoding and normalization needed tuning to avoid model bias.
- Limited feature diversity constrained prediction accuracy.

Final Outcome: A trained, tunable model framework capable of predicting learning material ratings with acceptable accuracy for potential integration in a personalized recommendation system.

References

- Scikit-learn documentation: https://scikit-learn.org/

- Seaborn and Matplotlib documentation for visualizations
- Pandas documentation: https://pandas.pydata.org/
- Z-score and IQR outlier detection techniques (Wikipedia, Statology)
- GridSearchCV for hyperparameter tuning (Scikit-learn user guide)