Case 2: Predicting Student Success in Online Courses

Overview

This project aims to predict student success (i.e., whether a student will complete or drop out of a course) on an online learning platform. The project involves merging several datasets related to student profiles, engagement data, and historical performance. A classification model (Random Forest) is built to predict student completion status, and feature importance analysis is conducted to determine which factors contribute most to the outcome. The goal is to identify at-risk students and suggest interventions to help them succeed.

Code Breakdown

1. Loading Libraries

Several Python libraries are used for this project:

- **pandas**: For data manipulation and analysis.
- **seaborn** and **matplotlib**: For visualizations.
- **scikit-learn**: For building and evaluating machine learning models.
- **joblib**: For saving the trained model.
- **numpy**: For numerical computations.

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score

from sklearn.metrics import accuracy_score, classification_report

import joblib

import numpy as np
```

---

2. Loading Datasets

Three datasets are used:

- **Student Profile Data** (student_profile_data.csv)
- **Course Engagement Data** (course_engagement_data.csv)
- **Historical Data** (historical_data.csv)

These are merged into a single dataframe using student_id as the common key.

```
profile_df = pd.read_csv('student_profile_data.csv')
```

```
engagement_df = pd.read_csv('course_engagement_data.csv')
historical_df = pd.read_csv('historical_data.csv')

merged_df = pd.merge(profile_df, engagement_df, on='student_id')
merged_df = pd.merge(merged_df, historical_df, on='student_id')
```

---

3. Defining Completion Status

A function determine_completion_status is used to classify whether a student will likely complete the course based on predefined thresholds for:

- Number of courses completed.
- Average quiz score across all courses.
- Number of logins per week.

This function generates the target variable completion_status.

```
def determine_completion_status(row):
    if (row['courses_completed'] > 3 and
        row['avg_score_across_courses'] > 70 and
        row['logins_per_week'] > 4):
        return 1  # Likely to complete
    else:
        return 0  # Likely to drop out

merged_df['completion_status'] = merged_df.apply(determine_completion_status, axis=1)
```

---

4. Handling Missing Values

Numerical columns are filled with their mean, and categorical columns are filled with their most frequent value (mode).

```
numerical_cols = ['logins_per_week', 'time_spent_on_platform (hrs)', 'avg_quiz_score',
                  'courses_completed', 'courses_started', 'avg_score_across_courses']
categorical_cols = ['gender', 'major', 'region']

merged_df[numerical_cols] = merged_df[numerical_cols].fillna(merged_df[numerical_cols].mean())
```

```
merged_df[categorical_cols] =
merged_df[categorical_cols].fillna(merged_df[categorical_cols].mode().iloc[0])
```

---

5. Data Encoding

Categorical variables are converted into one-hot encoded columns for compatibility with machine learning algorithms.

```
merged_df = pd.get_dummies(merged_df, columns=categorical_cols, drop_first=True)
```

---

6. Splitting Data for Training

The features (X) and target variable (y) are defined. The data is split into training and test sets using an 70/30 split.

```
X = merged_df.drop(columns=['student_id', 'completion_status'])

y = merged_df['completion_status']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

---

7. Random Forest Model with Hyperparameter Tuning

A **Random Forest** classifier is trained, with hyperparameter tuning performed using GridSearchCV. The best model is selected based on F1-score.

```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}


grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5,
scoring='f1', verbose=1)

grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_
```

```
GridSearchCV
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42),
             param_grid={'max_depth': [None, 10, 20, 30],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [50, 100, 200]},
             scoring='f1', verbose=1)
    estimator: RandomForestClassifier
RandomForestClassifier(random_state=42)
        RandomForestClassifier
RandomForestClassifier(random_state=42)
```

---

8. Model Evaluation

The performance of the model is evaluated using accuracy and classification report metrics on the test data.

y_pred_best = best_model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred_best))

print("\nClassification Report:\n", classification_report(y_test, y_pred_best))

```
Best Random Forest Model Evaluation:
Accuracy: 0.9904761904761905

Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        80
           1       1.00      0.96      0.98        25

    accuracy                           0.99       105
   macro avg       0.99      0.98      0.99       105
weighted avg       0.99      0.99      0.99       105
```

---

9. Feature Importance

The importance of each feature is visualized to determine which factors contribute the most to predicting student success.

importances = best_model.feature_importances_

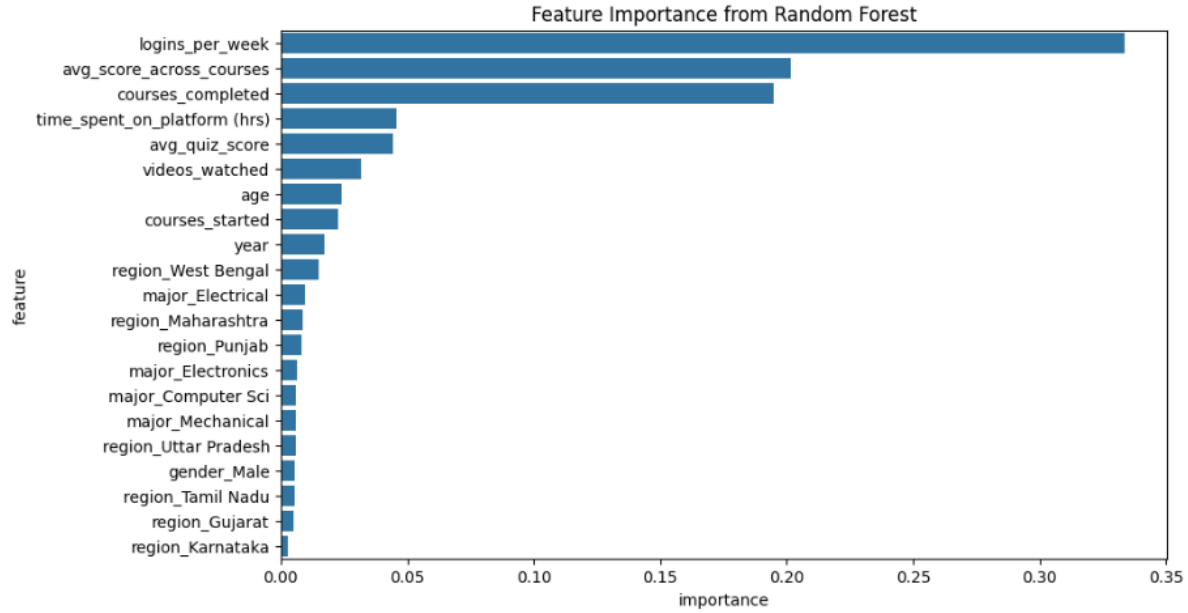
feature_importance_df = pd.DataFrame({'feature': X.columns, 'importance': importances}).sort_values(by='importance', ascending=False)

plt.figure(figsize=(10, 6))

sns.barplot(x='importance', y='feature', data=feature_importance_df)

plt.title('Feature Importance from Random Forest')

plt.show()



---

10. Cross-Validation

Cross-validation is used to evaluate the model's performance more robustly.

cv_scores = cross_val_score(best_model, X, y, cv=5, scoring='f1')

print("Cross-Validation F1 Scores:", cv_scores)

print("Mean Cross-Validation F1 Score:", cv_scores.mean())

```
Cross-Validation F1 Scores: [0.96551724 1.         1.         1.         1.        ]
Mean Cross-Validation F1 Score: 0.993103448275862
```

---

11. Saving the Model

The trained model is saved using **joblib** for future use.

```
joblib.dump(best_model, 'student_completion_model.pkl')
```

---

12. Identifying Likely Dropout Students

The model is used to predict the completion status for all students, and the students likely to drop out are saved in a separate file.

```
merged_df['predicted_completion_status'] = best_model.predict(X)

dropout_students = merged_df[merged_df['predicted_completion_status'] == 0]

dropout_students.to_csv('likely_dropout_students.csv', index=False)
```

---

13. MAP@K and NDCG@K Metrics

Functions are provided to calculate MAP@K (Mean Average Precision at K) and NDCG@K (Normalized Discounted Cumulative Gain at K) for ranking evaluation.

```
def apk(actual, predicted, k=10):
    if len(predicted) > k:
        predicted = predicted[:k]
    score = 0.0
    num_hits = 0.0
    for i, p in enumerate(predicted):
        if p in actual and p not in predicted[:i]:
            num_hits += 1.0
            score += num_hits / (i + 1.0)
    return score / min(len(actual), k) if actual else 0.0


def mapk(actual_list, predicted_list, k=10):
    return np.mean([apk(a, p, k) for a, p in zip(actual_list, predicted_list)])


def dcg_at_k(relevance_scores, k=10):
    relevance_scores = np.array(relevance_scores)[:k]
    return np.sum(relevance_scores / np.log2(np.arange(2, len(relevance_scores) + 2)))


def ndcg_at_k(actual, predicted, k=10):
    ideal_dcg = dcg_at_k(sorted([1 if i in actual else 0 for i in actual], reverse=True), k)
```

```
    if ideal_dcg == 0:
        return 0.0
    relevance_scores = [1 if p in actual else 0 for p in predicted]
    dcg = dcg_at_k(relevance_scores, k)
    return dcg / ideal_dcg
```

---

14. Example Usage of MAP@K and NDCG@K

The MAP@K and NDCG@K functions are tested with example data.

```
actual_list = [[1, 2, 3], [1], [2, 3, 4], [1, 2], [3]]
predicted_list = [[1, 2, 4], [2], [1, 2, 3], [1, 3], [1, 2]]


print(f'MAP@3: {mapk(actual_list, predicted_list, k=3)}')
ndcg_scores = [ndcg_at_k(actual, predicted, k=3) for actual, predicted in zip(actual_list, predicted_list)]
print(f'NDCG@3: {np.mean(ndcg_scores)}')
```

```
MAP@3: 0.31111111111111106
NDCG@3: 0.38184582074626466
```

Conclusion

The project successfully predicts student completion status with a Random Forest classifier, analyzes feature importance, and provides insights into student behavior.