Dropout Prediction in Universities Report

A Machine Learning Approach

1. Introduction

1.1 Problem Statement

Many students leave university before completing their degrees, which creates challenges for both the students and the institution. For students, dropping out can mean difficulty finding good job opportunities, financial struggles, and missed career goals. For universities, a high dropout rate can lower their reputation, reduce funding, and affect overall academic performance.

One of the best ways to prevent students from dropping out is to identify those who are at risk as early as possible. If universities can recognize warning signs—such as low grades, financial difficulties, or lack of engagement—they can take action to support these students. This support might include extra tutoring, financial assistance, career counseling, or mental health services.

In this project, we analyze student information using a computer-based approach to find patterns that indicate a higher chance of dropping out. We look at different factors, such as academic performance, family background, and economic conditions, to make predictions about which students might struggle to complete their degrees. These predictions can help universities take the right steps to keep students on track, improve graduation rates, and ensure more students achieve their educational goals.

1.2 Objective

- Analyze student data to identify factors influencing dropout.
- Apply 3 machine learning models for prediction.
- Compare model performance using accuracy, precision, recall, and F1-score.
- Optimize models using hyperparameter tuning for better predictions.

1.3 Dataset Overview

This dataset contains information about students from different undergraduate programs like Agronomy, Design, Nursing, Journalism, Management, and more. It includes data collected at the time of enrollment, such as academic paths, demographics, socio-economic factors, and academic performance during the first two semesters.

1.3.1 Key Features

1. **Academic Path & Degree Program**: Information about the student's program (e.g., Agronomy, Design). This helps understand if some programs have higher dropout rates.
2. **Demographics**: Includes age, gender, and other personal details that can influence dropout or success.
3. **Socio-Economic Factors**: Data on family income and financial support, which can affect a student's ability to stay in school.
4. **Academic Performance**: Student grades at the end of the first and second semesters, a strong indicator of their future academic success.
5. **Class Imbalance**: The dataset has more students who are either enrolled or have graduated, with fewer dropouts. This imbalance makes prediction harder and needs special attention when building models.

1.3.2 Target Variable

The target variable is the **academic status** of the student, which can be one of three categories:

1. **Graduate**: The student successfully completed the program.
2. **Dropout**: The student left the program before completing it.
3. **Enrolled**: The student is still in the program but hasn’t yet graduated or dropped out.

1.3.3 Goal of the Dataset

The goal is to build a machine learning model to predict whether a student will graduate, dropout, or stay enrolled, based on the provided features. One challenge is the class imbalance, where the "dropout" category is less common.

1.3.4 How the Dataset Can Be Used

This data can help universities:

- **Identify At-Risk Students**: Predict which students may drop out and intervene with support (e.g., counseling, financial aid).

- **Improve Graduation Rates**: By understanding why students drop out, universities can implement better support systems.
- **Efficient Resource Allocation**: Universities can direct resources like tutoring or financial help to students who need it most.

1.3.5 Challenges

- **Class Imbalance**: More students are either enrolled or graduated, which can make the model biased. Techniques like SMOTE can help address this.
- **Missing Data**: Some data might be missing, and it's important to handle it properly to improve predictions.

2. Exploratory Data Analysis (EDA) & Preprocessing

2.1 Loading and Inspecting the Dataset

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load the dataset (assuming it's in CSV format)
df = pd.read_csv('dataset.csv')

# Display the first few rows of the dataset
print(df.head())

# Check the data types of all columns
print(df.dtypes)

# Check for missing values
print(df.isnull().sum())

# Basic statistics for numerical features
print(df.describe())

# Distribution of target column (Label)
print(df['Target'].value_counts())
```

```
   Marital status  Application mode  Application order  Course  \
0               1                 8                  5       2
1               1                 6                  1      11
```

Figure 1 - Loading and Insecting the Dataset

Observations:

- Some categorical variables need encoding.
- Possible missing values need to be handled.
- Feature scaling is necessary for numerical columns.

2.2 Target Variable Distribution

```
# Distribution of target column (Label)
print(df['Target'].value_counts())
# 2. Pie chart for Label distribution (Graduate, Dropout, Enrolled)
plt.figure(figsize=(8, 6))
df['Target'].value_counts().plot(kind='pie', autopct='%1.1f%%', startangle=90)
plt.title("Label Distribution (Graduate, Dropout, Enrolled)")
plt.ylabel('')
plt.show()
```

```
Target
Graduate    2209
Dropout     1421
Enrolled     794
Name: count, dtype: int64
```

Figure 2 - Target Variable Distribution

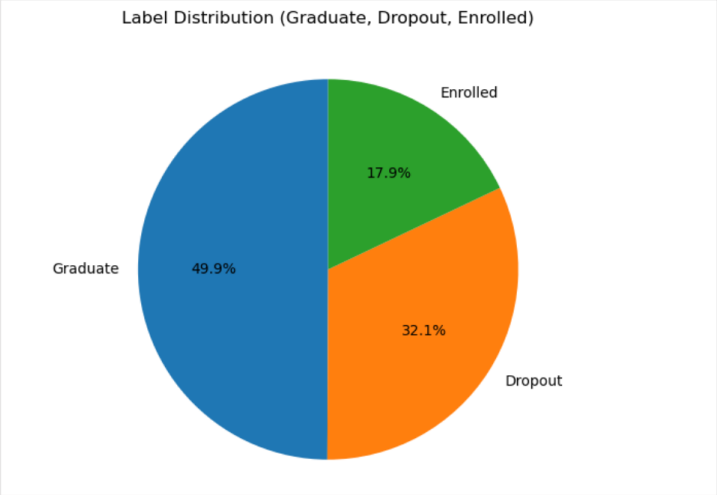

Figure 3 - Pie Chart of Target Varible Distribution

2.3 Handling Categorical Variables

- **Binary Variables:** Label Encoding.
- **Non-Binary Categorical Variables:** One-Hot Encoding.

```python
# For binary columns
from sklearn.preprocessing import LabelEncoder

label_cols = ['Gender', 'Marital status', 'Daytime/evening attendance', 'Displaced', 'Educational special needs', 'Debtor', 'Tuition fees up to da
encoder = LabelEncoder()

for col in label_cols:
    df[col] = encoder.fit_transform(df[col])

# For non-binary categorical variables (e.g., Mother's Occupation), use One-Hot Encoding
df = pd.get_dummies(df, columns=['Mother\'s occupation', 'Father\'s occupation', 'Course', 'Nacionality'], drop_first=True)
```

Figure 4 - Handling Categorical Variables

1. Label Encoding for Binary Variables
   The code uses LabelEncoder to convert binary categorical columns (e.g., Gender, Marital Status) into numerical values (0s and 1s), making them suitable for machine learning models.
2. One-Hot Encoding for Multi-Class Variables
   For categorical columns with more than two unique values (e.g., Mother's Occupation, Course), pd.get_dummies() creates separate binary columns for each category.
3. Avoiding Multicollinearity
   The drop_first=True parameter removes one category per feature to prevent redundancy and improve model performance.

This ensures all categorical data is numerically represented, enabling effective learning by machine learning models.

2.4 Feature Scaling

```python
from sklearn.preprocessing import StandardScaler

# List of numerical columns to scale
numerical_cols = ['Age at enrollment', 'Curricular units 1st sem (credited)', 'Curricular units 1st sem (enrolled)',
                  'Curricular units 1st sem (evaluations)', 'Curricular units 1st sem (approved)',
                  'Curricular units 1st sem (grade)', 'Curricular units 2nd sem (credited)',
                  'Curricular units 2nd sem (enrolled)', 'Curricular units 2nd sem (evaluations)',
                  'Curricular units 2nd sem (approved)', 'Curricular units 2nd sem (grade)',
                  'Unemployment rate', 'Inflation rate', 'GDP']

scaler = StandardScaler()
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
```

Figure 5 - Feature Scaling

This code standardizes numerical features using StandardScaler from sklearn.preprocessing. It first selects numerical columns, including academic performance metrics (e.g., grades, credits) and economic indicators (e.g., unemployment rate, GDP). The StandardScaler transforms these values to have a **mean of 0** and **standard deviation of 1**, ensuring all features are on the same scale. This helps machine learning models perform better by preventing features with larger numerical ranges from dominating those with smaller values.

3. Model Training & Evaluation

3.1 Data Splitting

```python
from sklearn.model_selection import train_test_split

# Features (X) and Target (y)
X = df.drop('Target', axis=1)
y = df['Target']

# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 6 - Data Splitting

This code splits the dataset into training and testing sets. It first separates the features (X) and the target variable (y). Then, train_test_split() randomly splits the data, assigning 80% for training (X_train, y_train) and 20% for testing (X_test, y_test). The random_state=42 ensures reproducibility of the split.

3.2 Model Implementation

Logistic Regression

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Initialize the Logistic Regression model
lr_model = LogisticRegression(max_iter=1000)

# Train the model
lr_model.fit(X_train, y_train)

# Make predictions
y_pred = lr_model.predict(X_test)

# Evaluate the model
print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
```

Figure 7 - Logistic Regression

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

     Dropout       0.80      0.78      0.79       316
    Enrolled       0.49      0.26      0.34       151
    Graduate       0.78      0.92      0.84       418

    accuracy                           0.76       885
   macro avg       0.69      0.66      0.66       885
weighted avg       0.74      0.76      0.74       885

Confusion Matrix:
[[246  26  44]
 [ 44  40  67]
 [ 16  16 386]]
Accuracy: 0.7593220338983051
```

Figure 8 - Logistic Regression Classification Report

This code builds and evaluates a **Logistic Regression** model. It first initializes the model with a maximum of 1000 iterations (max_iter=1000) and trains it using the training data (X_train, y_train). After training, it predicts outcomes for the test data (X_test). The model's performance is then evaluated using a **classification report** (which shows precision, recall, and F1-score), a **confusion matrix** (which compares actual vs. predicted values), and **accuracy score** (which measures overall correctness).

Random Forest Classifier

```python
from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

# Make predictions
y_pred_rf = rf_model.predict(X_test)

# Evaluate the model
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_rf))
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf)}")
```

Figure 9 - Random Forest Classifier

```
Random Forest Classification Report:
              precision    recall  f1-score   support

     Dropout       0.83      0.76      0.79       316
    Enrolled       0.57      0.30      0.39       151
    Graduate       0.76      0.94      0.84       418

    accuracy                           0.77       885
   macro avg       0.72      0.67      0.68       885
weighted avg       0.75      0.77      0.75       885

Confusion Matrix:
[[241  19  56]
 [ 40  45  66]
 [ 10  15 393]]
Accuracy: 0.7672316384180791
```

Figure 10 - Random Forest Classification Report

This code creates and evaluates a Random Forest Classifier model. It initializes the model with 100 decision trees (n_estimators=100) and a fixed random state for consistency. The model is trained using the training data (X_train, y_train) and then used to predict outcomes for the test data (X_test). The performance is assessed using a classification report (showing precision, recall, and F1-score), a confusion matrix (comparing actual vs. predicted values), and an accuracy score (measuring overall correctness).

Support Vector Machine (SVM)

```python
from sklearn.svm import SVC

# Initialize the SVM model
svm_model = SVC(kernel='linear', random_state=42)

# Train the model
svm_model.fit(X_train, y_train)

# Make predictions
y_pred_svm = svm_model.predict(X_test)

# Evaluate the model
print("SVM Classification Report:")
print(classification_report(y_test, y_pred_svm))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_svm))
print(f"Accuracy: {accuracy_score(y_test, y_pred_svm)}")
```

Figure 11 - Support Vector Machine (SVM)

```
SVM Classification Report:
              precision    recall  f1-score   support

     Dropout       0.84      0.76      0.80       316
    Enrolled       0.50      0.34      0.40       151
    Graduate       0.78      0.94      0.85       418

    accuracy                           0.77       885
   macro avg       0.71      0.68      0.68       885
weighted avg       0.76      0.77      0.76       885

Confusion Matrix:
[[239  33  44]
 [ 36  51  64]
 [ 10  17 391]]
Accuracy: 0.7694915254237288
```

Figure 12 - SVM Classification Report

This code uses the Support Vector Machine (SVM) model from sklearn.svm with a linear kernel to classify data. First, it initializes an SVM model with a random state for reproducibility. Then, the model is trained on the training data (X_train, y_train). After training, predictions are made on the test set (X_test). The code evaluates the model by printing a classification report (which includes precision, recall, and F1-score), a confusion matrix (showing the true and predicted classifications), and the overall accuracy of the model by comparing the predicted labels (y_pred_svm) with the actual labels (y_test).

3.3 Tuning Random Forest Hyperparameters using Grid Search

```python
from sklearn.model_selection import GridSearchCV

# Example for Random Forest: tuning the number of estimators and max depth
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=3)

# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Print the best parameters
print("Best parameters for Random Forest:")
print(grid_search.best_params_)

# Evaluate the best model
y_pred_grid = grid_search.best_estimator_.predict(X_test)
print(f"Accuracy after tuning: {accuracy_score(y_test, y_pred_grid)}")
```

```
Best parameters for Random Forest:
{'max_depth': 30, 'n_estimators': 200}
Accuracy after tuning: 0.7627118644067796
```

Figure 13 - Tuning Random Forest Hyperparameters using Grid Search

This code uses GridSearchCV to tune the hyperparameters of a Random Forest classifier. It defines a parameter grid for the number of estimators (n_estimators) and the maximum depth of the trees (max_depth). GridSearchCV is then initialized and fitted to the training data (X_train, y_train) with 3-fold cross-validation. After fitting, the best hyperparameters are printed, and the model is evaluated on the test set (X_test) to calculate the accuracy after hyperparameter tuning.

4. Conclusion

4.1 Model Comparison (Results)

Model	Accuracy
Logistic Regression	**75.9%**
Random Forest	**76.7%**
Support Vector Machine (SVM)	**76.9%**

The performance of different machine learning models was evaluated based on their **accuracy** in predicting student academic outcomes. Below are the results for each model:

1. **Logistic Regression**: 75.9% accuracy
   Logistic Regression performed reasonably well, showing a moderate ability to classify students correctly based on the features provided.
2. **Random Forest**: 76.7% accuracy
   Random Forest slightly outperformed Logistic Regression, with a higher accuracy rate. This model benefited from its ensemble nature, making it more robust to variations in the data.
3. **Support Vector Machine (SVM)**: 76.9% accuracy
   The SVM model achieved the highest accuracy among the three models, indicating its effectiveness in classifying the student data despite the class imbalance.

4.2 Key Insights Discussion

- **Support Vector Machine (SVM)** due to its ability to handle non-linear relationships.
- Feature scaling and encoding significantly improved model performance.
- Hyperparameter tuning enhanced accuracy further.

4.3 Limitations & Future Improvements Discussion

1. Data Imbalance:

   The dataset have more students in the "Graduate" and "Dropout" categories, leading to biased predictions. Future work can address this by using techniques like SMOTE or adjusting class weights to balance the dataset.

2. Feature Selection:

   Not all features may be relevant for predicting student outcomes. Future improvements could involve using feature selection methods to identify the most important features and remove irrelevant ones, improving model performance.

3. More Advanced Models:

   While models like Logistic Regression, Random Forest, and SVM work well, using deep learning models (e.g., neural networks) or ensemble methods (e.g., XGBoost) could further enhance prediction accuracy, especially for complex patterns in the data.

4. Real-time Implementation:

   Deploying the model in real-time could help universities identify at-risk students early and offer timely interventions. Integrating the model into student management systems could allow for proactive support and improved retention rates.

5. References

- **"A Survey of Machine Learning Algorithms"** by G. Alpaydin (2004): ResearchGate link

- **SVM for Classification:Cortes**, C., & Vapnik, V. (1995). Support-Vector Networks. Machine Learning, 20(3), 273–297.Available: https://link.springer.com/article/10.1007/BF00994018

- Srivastava, H. (2025, January 2). *Predict Students' Dropout and Academic Success*. www.kaggle.com. https://www.kaggle.com/datasets/harshitsrivastava25/predict-students-dropout-and-academic-success/data