# Student Dropout Prediction Using Deep FM

Guide : Dr.K.Sharada

Reviewer : Dr.N.Suresh Kumar

Dr.Veena

Submitted by

Etukuri Karthik - 122010322027

N.V.Nithin Kumar - 122010322019

Challa Snehalatha - 122010323013

Kaliki Jayakrishna - 122010322016

# TABLE OF CONTENTS


Abstract

Introduction → Literature review

Problem Identification → Objectives → System Methodology

Algorithm Explanation → Coding

Conclusion

# ABSTRACT

The research endeavors to develop a predictive model to tackle the challenge of student dropout rates within online educational platforms. Utilizing a dataset sourced from an online learning environment, the study meticulously addresses class imbalance issues during data preprocessing, ensuring a representative dataset for model training. Through comprehensive data analysis, crucial insights into feature distributions and class relationships are unveiled. The model architecture adopts a sophisticated DeepFM approach, amalgamating Factorization Machines and Deep Neural Networks to adeptly capture intricate feature interactions.

Model training involves meticulous optimization, with evaluation metrics such as precision, recall, and the confusion matrix employed to gauge efficacy. The empirical findings highlight the profound efficacy of the DeepFM model in discerning student dropout propensities with notable accuracy, thereby providing educational stakeholders with invaluable insights to fortify student retention strategies.

# INTRODUCTION

- The rise of online education offers flexibility and accessibility to learners, but it also comes with a challenge: student dropout rates. This is a concern for educators and policymakers alike.
- To address this, researchers are looking to data science and predictive modeling. This study aims to develop a model that can identify students at risk of dropping out early on. By using data analysis techniques and machine learning, the study hopes to provide valuable insights to improve student retention strategies in online education platforms.
- Against this backdrop, the present study aims to build upon the existing body of research by proposing a novel approach to addressing class imbalance in predictive modeling for student dropout. By combining oversampling and undersampling techniques with the DeepFM architecture, the research seeks to enhance the predictive accuracy of the model while ensuring equitable representation of minority classes.

# LITERATURE REVIEW

- Deep Reinforcement Factorization Machines: A Deep Reinforcement Learning Model with Random Exploration Strategy and High Deployment Efficiency.
- Machine Learning with Oversampling and Undersampling Techniques.
- Deep Factorization Machines network with Non-linear interaction for Recommender System.
- Deep Factorization Machines for Knowledge Tracing

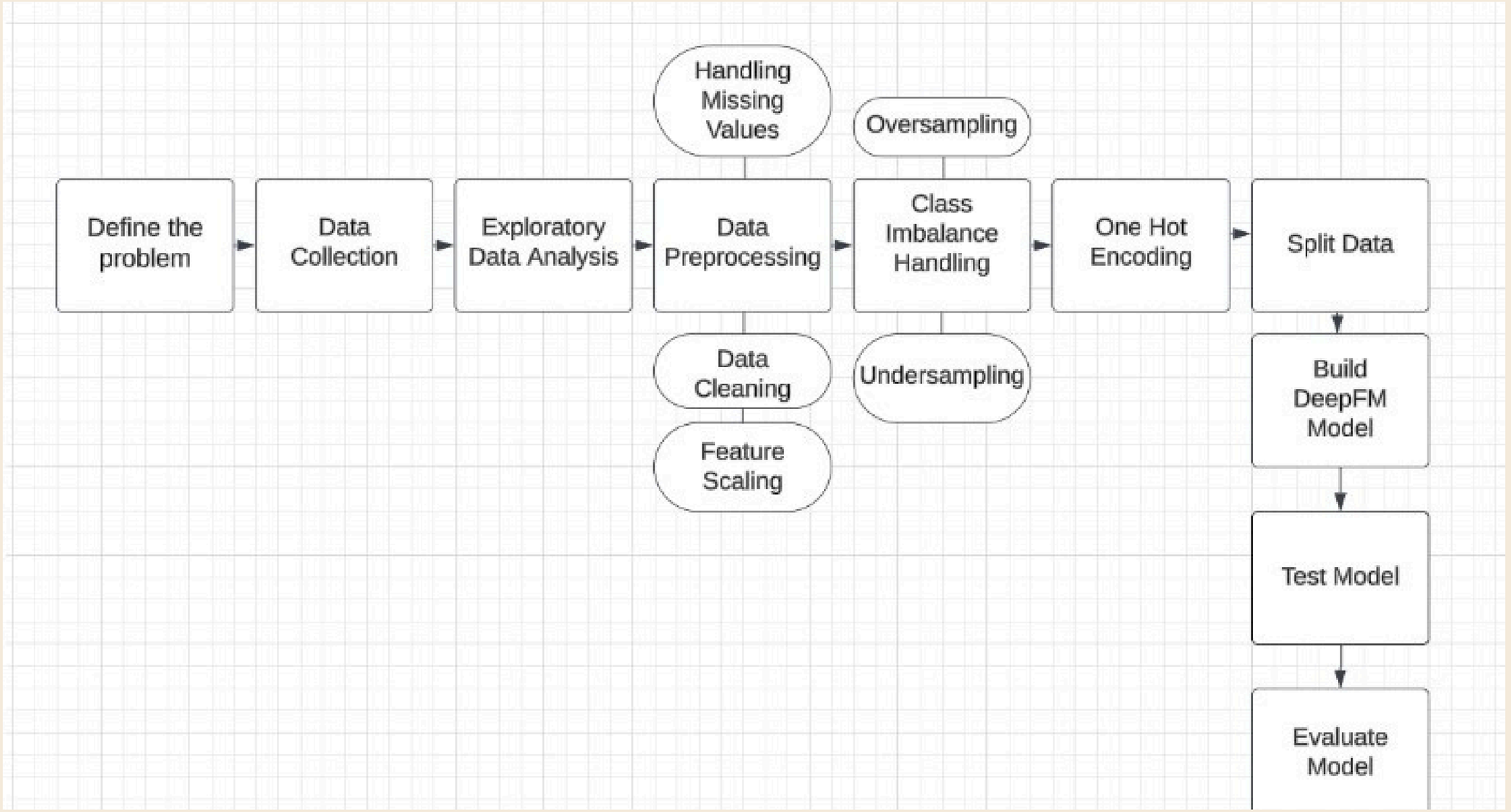
# PROBLEM IDENTIFICATION:

- Predicting student dropouts is complex due to two main reasons:
- First, there's class imbalance. Most students complete their studies, which means there's much less data available on students who drop out. This can make it difficult for algorithms to learn the dropout patterns.
- Second, the data itself can be overwhelming. There might be dozens of factors influencing student dropout, from demographics to grades. These many factors can make it hard to identify the most important ones and can slow down the analysis.

# OBJECTIVES:

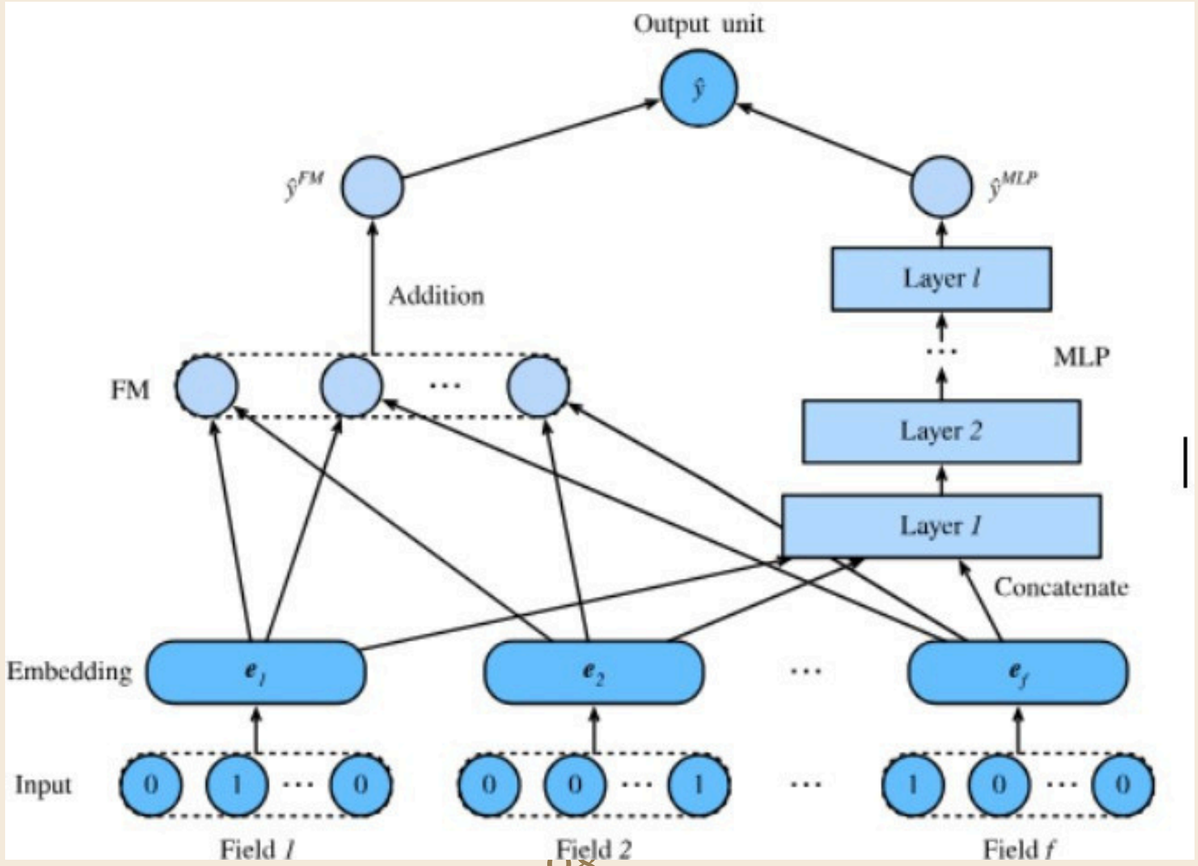
- Develop a predictive model for student dropout rates in higher education, aiming to address challenges associated with non-standardized data formats, class imbalance, and high-dimensional data through dimensionality reduction techniques.
- Implement a comprehensive approach encompassing data collection, exploratory data analysis (EDA), data preparation, and handling of class imbalance to ensure robust model development.
- Develop techniques to address class imbalance specifically, ensuring that the model is trained on a balanced representation of both dropout and graduation instances. This may involve employing resampling methods such as oversampling the minority class or undersampling the majority class, as well as exploring advanced techniques like Synthetic Minority Over-sampling Technique (SMOTE) or class-weighted loss functions to mitigate the impact of class imbalance on model performance.
- Select and utilize appropriate machine learning models essential for accurate predictions, considering the complexities inherent in the dataset and the specific requirements of dropout rate forecasting in higher education settings.
- Incorporate the development of a DeepFM model, leveraging its capabilities in handling both categorical and numerical features effectively, thus providing a powerful tool for capturing intricate patterns within the data.

# SYSTEM METHODOLOGY



# ALGORITHM EXPLANATION

DeepFM, short for Deep Factorization Machine, is a hybrid recommendation algorithm that combines the strengths of factorization machines (FM) and deep neural networks (DNN). It was proposed to address the limitations of traditional recommendation systems, such as collaborative filtering and content-based methods, by capturing both low-order and high-order feature interactions effectively.

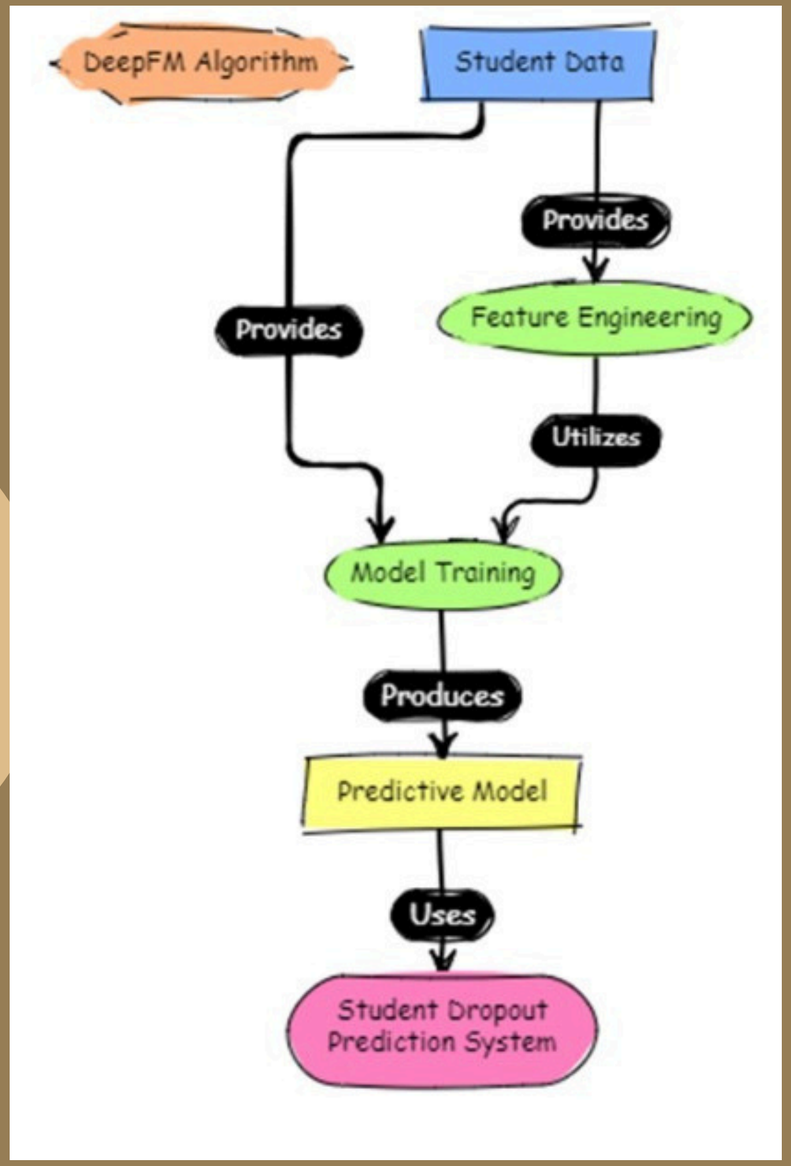


**Factorization Machines (FM):**

- Factorization Machines are a powerful class of models for handling sparse and high-dimensional data, commonly used in recommendation systems and regression tasks.
- FM models are designed to capture interactions between features by factorizing their interactions into low-rank matrices.
- They have linear complexity with respect to the number of features, making them efficient for large-scale datasets.

**Deep Neural Networks (DNN):**

- Deep Neural Networks are versatile models capable of learning complex patterns and representations from data through multiple layers of non-linear transformations.
- DNNs excel at capturing intricate feature interactions and hierarchies in the data, making them suitable for tasks with high-dimensional and non-linear relationships.




**Hybrid Architecture:**

- DeepFM combines the FM and DNN architectures into a hybrid model to leverage their complementary strengths.
- The FM component captures low-order feature interactions efficiently, while the DNN component learns higher-order feature interactions and representations through deep layers.
- By combining these components, DeepFM can effectively model both linear and non-linear relationships between features, providing enhanced predictive power.

**Architecture Overview:**

- The architecture of DeepFM typically consists of two main components: the FM component and the DNN component.
- The FM component computes the low-order interactions between features using factorization techniques, producing an embedding vector for each feature.

- The DNN component takes the concatenation of these embedding vectors as input and passes it through multiple hidden layers of neurons, learning complex feature representations.
- The final output layer of the DNN predicts the target variable (e.g., dropout prediction) based on the learned representations.

## Training and Optimization:

- DeepFM is trained using gradient-based optimization techniques, such as stochastic gradient descent (SGD) or Adam, to minimize a loss function (e.g., binary cross-entropy for binary classification tasks).
- During training, both the FM and DNN components are jointly optimized to learn the optimal parameters that minimize the prediction error.

# Coding:

## Loading libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from keras.layers import Input, Dense, Lambda, Subtract
from keras.models import Model
import keras.backend as K
```

## Converting csv file into pandas data frame

```
dataset = pd.read_csv("/content/dataset.csv")
dataset.head()
```

| | Marital status | Application mode | Application order | Course | Daytime/evening attendance | Previous qualification | Nacionality | Mother's qualification | Father's qualification | Mother's occupation | ... | Curricular units 2nd sem (credited) | Curricular units 2nd sem (enrolled) | un (e |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 8 | 5 | 2 | 1 | 1 | 1 | 13 | 10 | 6 | ... | 0 | 0 | |
| 1 | 1 | 6 | 1 | 11 | 1 | 1 | 1 | 1 | 3 | 4 | ... | 0 | 6 | |
| 2 | 1 | 1 | 5 | 5 | 1 | 1 | 1 | 22 | 27 | 10 | ... | 0 | 6 | |
| 3 | 1 | 8 | 2 | 15 | 1 | 1 | 1 | 23 | 27 | 6 | ... | 0 | 6 | |
| 4 | 2 | 12 | 1 | 3 | 0 | 1 | 1 | 22 | 28 | 10 | ... | 0 | 6 | |

5 rows × 35 columns

# Oversampling

```
# Count the number of instances in each class
class_counts = dataset['Target'].value_counts()

# Get the number of instances in the majority class
majority_class_count = class_counts['Graduate']

# Calculate the number of instances to replicate for the minority class
num_instances_to_replicate = majority_class_count - class_counts['Dropout']

# Select instances from the minority class
minority_class_instances = dataset[dataset['Target'] == 'Dropout']

# Replicate the minority class instances
replicated_instances = minority_class_instances.sample(n=num_instances_to_replicate, replace=True)

# Append the replicated instances to the original dataset
dataset = dataset.append(replicated_instances)

# Shuffle the dataset
dataset = dataset.sample(frac=1).reset_index(drop=True)
```

```
<ipython-input-5-8a37af3c4f2c>:19: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  dataset = dataset.append(replicated_instances)
```

```
dropout_count = dataset[dataset['Target'] == "Dropout"].shape[0]
graduate_count = dataset[dataset['Target'] == "Graduate"].shape[0]

print("Dropout count:", dropout_count)
print("Graduate count:", graduate_count)
```

```
Dropout count: 2209
Graduate count: 2209
```

# Creating DeepFM Model:

```
[ ]  dense_feats = [DenseFeat(feat, 1) for feat in x_train.select_dtypes('number').columns]
     sparse_feats = []

[ ]  linear_cols = sparse_cols = dense_feats + sparse_feats
     dnn_cols = dense_feats + sparse_feats

[ ]  train_data = {feat: x_train[feat].values.reshape(-1, 1) for feat in x_train.columns}
     test_data = {feat: x_test[feat].values.reshape(-1, 1) for feat in x_test.columns}

[ ]  model = DeepFM(linear_cols, dnn_cols, task='binary')

     # Compile the model
     model.compile("adam", "binary_crossentropy", metrics=['accuracy'])
```

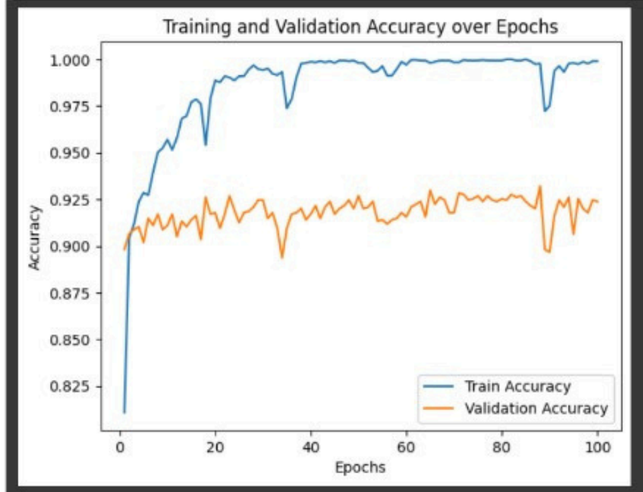
# Training the model

```
# Train the model
history = model.fit(train_data, y_train.values, batch_size=64, epochs=1000, validation_data=(test_data, y_test.values))
```

```
Epoch 72/100
49/49 [==============================] - 0s 10ms/step - loss: 0.0034 - accuracy: 0.9994 - val_loss: 0.5303 - val_accuracy: 0.9072
Epoch 73/100
49/49 [==============================] - 0s 8ms/step - loss: 0.0031 - accuracy: 0.9997 - val_loss: 0.5143 - val_accuracy: 0.9125
Epoch 74/100
49/49 [==============================] - 0s 8ms/step - loss: 0.0013 - accuracy: 0.9997 - val_loss: 0.5140 - val_accuracy: 0.9163
Epoch 75/100
49/49 [==============================] - 0s 8ms/step - loss: 8.0477e-04 - accuracy: 1.0000 - val_loss: 0.5337 - val_accuracy: 0.9103
Epoch 76/100
49/49 [==============================] - 0s 8ms/step - loss: 6.6182e-04 - accuracy: 1.0000 - val_loss: 0.5438 - val_accuracy: 0.9095
Epoch 77/100
49/49 [==============================] - 0s 8ms/step - loss: 4.4337e-04 - accuracy: 1.0000 - val_loss: 0.5563 - val_accuracy: 0.9103
Epoch 78/100
49/49 [==============================] - 0s 8ms/step - loss: 3.9399e-04 - accuracy: 1.0000 - val_loss: 0.5560 - val_accuracy: 0.9103
```

# Training and Validation Curves:

```
epochs = range(1, len(history.history['accuracy']) + 1)

# Plot training and validation accuracy
plt.plot(epochs, history.history['accuracy'], label='Train Accuracy')
plt.plot(epochs, history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy over Epochs')
plt.legend()
plt.show()
```



# Implementation and Results:

The code was implemented in 2 variations:

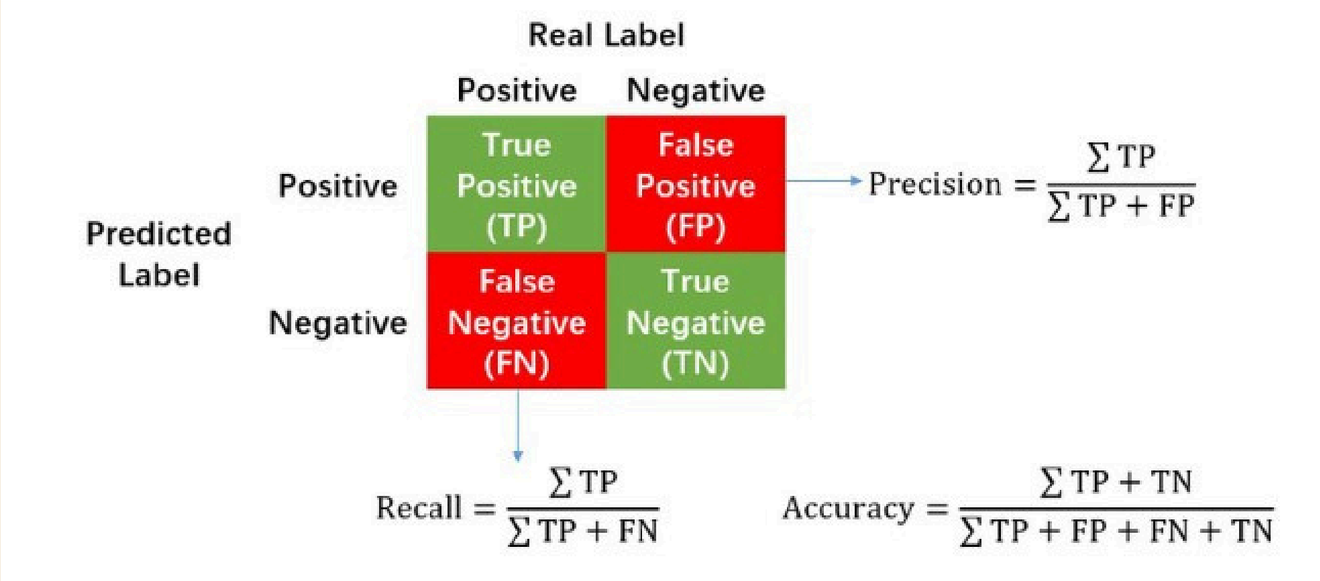
- Original dataset
- Oversampling

**Accuracy** shows how often a classification ML model is correct overall.

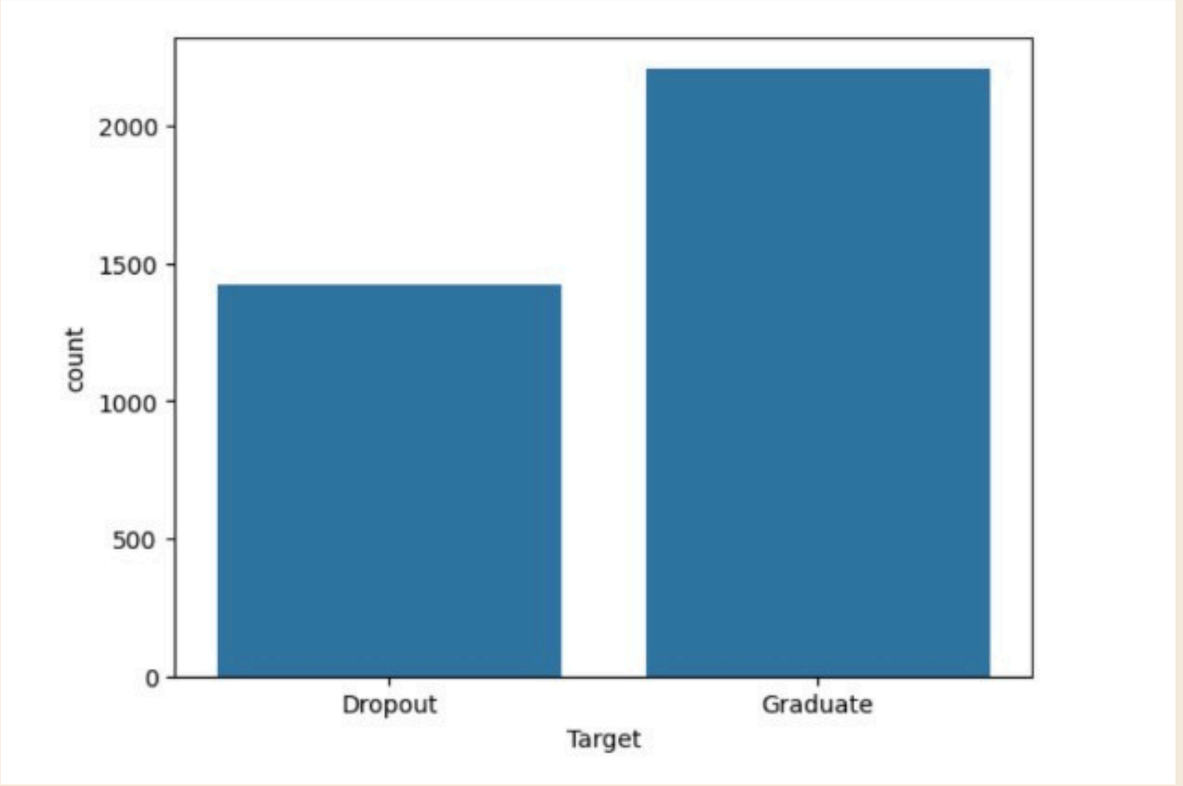
**Precision** shows how often an ML model is correct when predicting the target class.

**Recall** shows whether an ML model can find all objects of the target class.

Consider the class balance and costs of different errors when choosing the suitable metric.

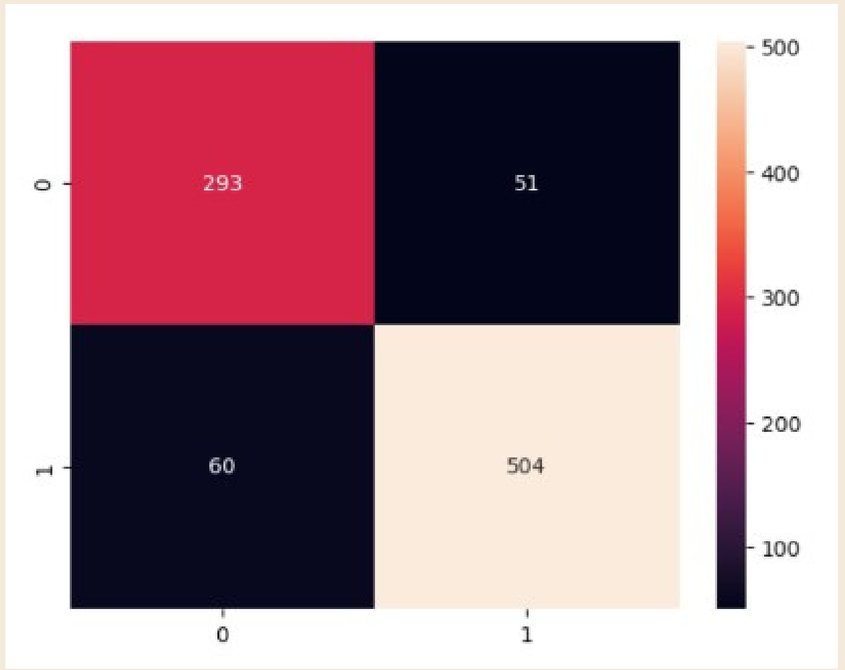


# Original Dataset




- The dataset is significantly class imbalanced as shown in the table above, in such cases, both standard and advanced classifiers tend to be overwhelmed by the large classes and ignore the small ones.
- The results are not desired as expected.

## Confusion matrix:

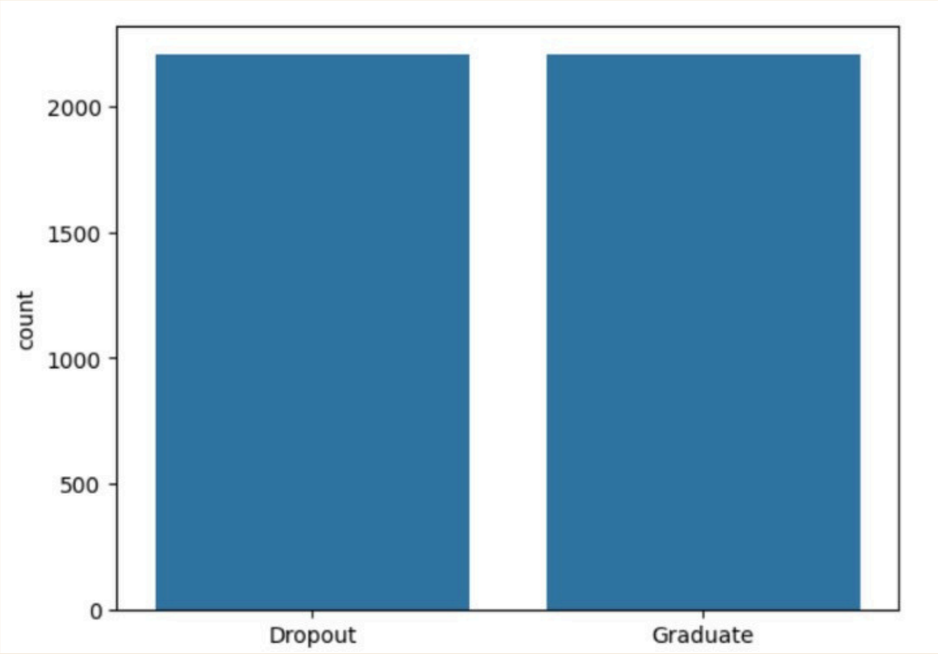



```
Precision: 0.8785
Recall: 0.8778
              precision    recall  f1-score   support

         0.0       0.83      0.85      0.84       344
         1.0       0.91      0.89      0.90       564

    accuracy                           0.88       908
   macro avg       0.87      0.87      0.87       908
weighted avg       0.88      0.88      0.88       908
```

## Accuracy scores

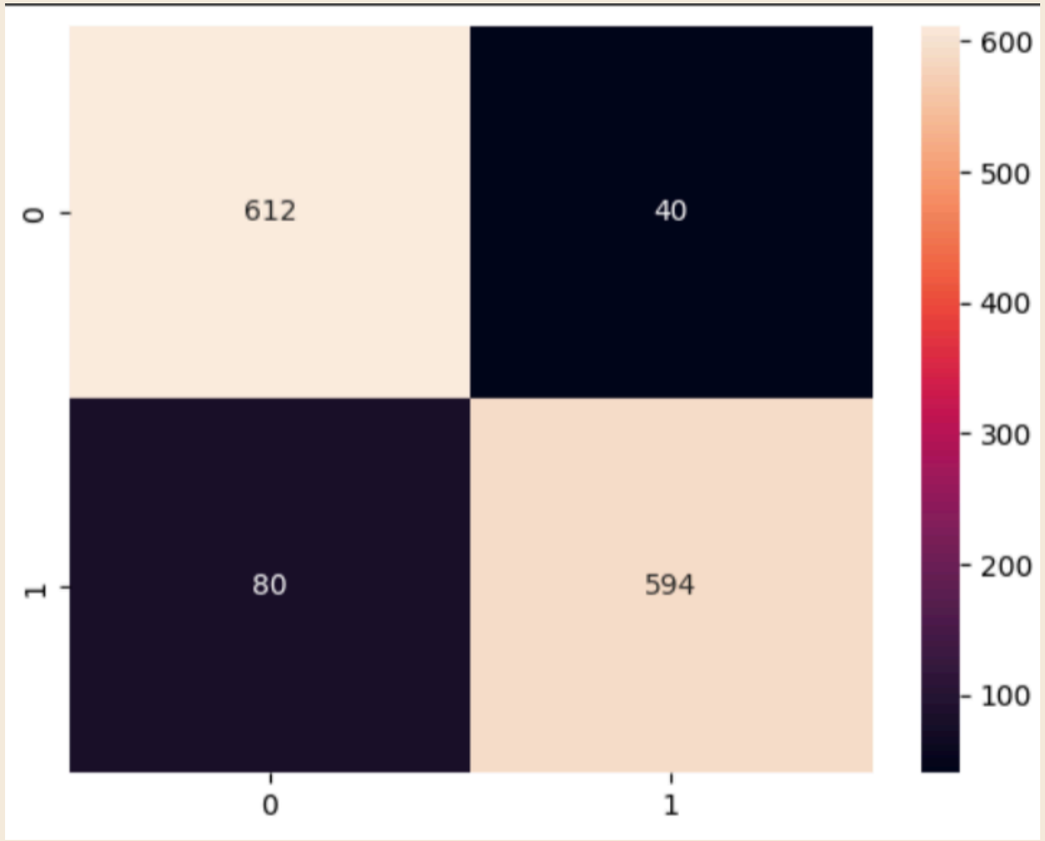
**Used Oversampling:**

- In an attempt to improve the overall accuracy, precision and recall we have implemented Oversampling where we increased the Dropout instances to match the number of Graduate instances by randomly duplicating some instances.
- As we expected, the overall accuracy, precision and recall improved after we implemented the oversampling technique.




```
Precision: 0.9239
Recall: 0.9238
              precision    recall  f1-score   support

         0.0       0.92      0.93      0.92       656
         1.0       0.93      0.92      0.92       670

    accuracy                           0.92      1326
   macro avg       0.92      0.92      0.92      1326
weighted avg       0.92      0.92      0.92      1326
```

# Confusion matrix after Oversampling



# CONCLUSION

- In conclusion, the implementation of DeepFM for student dropout prediction showcased promising results in accurately identifying potential dropout instances in higher education settings.

- By employing advanced data preprocessing techniques, including class imbalance handling through oversampling and undersampling, and leveraging the DeepFM model's ability to capture complex feature interactions, we were able to construct a robust predictive model.

- The comprehensive evaluation of the model's performance demonstrated its effectiveness in accurately predicting dropout instances, as evidenced by high precision, recall, and accuracy scores. The insights gained from the confusion matrix further highlighted the model's capability to make informed predictions across different classes.

- Through this study, we have provided valuable contributions to the field of educational analytics by offering a practical and scalable approach to student dropout prediction. By identifying at-risk students early on, educational institutions can proactively implement intervention strategies to support these students and improve overall retention rates.

# Thank You