Predicting Student Graduate Success with Machine Learning

Project Group 43
Orlando Closs (2720653), Hanna Tóth (2733338),
Mihaly Lorinc (2741323), Aron Ferencz (2727043),
Mark Bartos (2724195)

March 31, 2023

Abstract

Machine learning has shown great potential in improving education outcomes, and this paper aims to contribute to this growing field by applying machine learning techniques to predict student graduate success. The study utilizes a public Kaggle dataset and evaluates various machine learning algorithms to determine the most accurate model for predicting graduate success. Results indicate that machine learning algorithms such as SVM, Neural Network, and Logistic Regression can accurately predict graduate success with an accuracy of over 90 percent. These findings have significant implications for educational institutions seeking to improve the percentage of graduating students.

1 Introduction

The ability to predict student graduate success is essential for universities and educational institutions to identify students who may be at risk of dropping out and to provide them with the necessary support and resources to succeed. This research paper aims to apply machine learning techniques to predict student graduate success and determine the most accurate machine learning model for doing so.

The dataset used in this study was obtained from Kaggle [7] and will be preprocessed using packages such as pandas, sklearn, and numpy in Python. These same packages will be used for building the machine learning methods alongside the seaborn package for data visualization.

Accuracy will be used as a performance metric [10] to optimize and evaluate these models and based on prior research [8], the hypothesis is that machine learning models SVM and logistic regression will perform well in predicting graduate success due to the high dimensionality of the data that recursive feature elimination fails to reduce in preprocessing. Both SVM and logistic regression are robust to noisy data and can handle high-dimensional feature spaces, making them suitable for this task. Furthermore, ensemble methods are expected to outperform decision trees due to their ability to reduce variance and capture non-linear relationships between features.

Overall, this study aims to contribute to the literature surrounding machine learning in education and provide insights into the most accurate machine learning model for predicting student graduate success. The research question is whether we can accurately predict graduate success using machine learning, and if so, what is the best machine learning model for the task? The answer to this question will have significant implications for educational institutions seeking to improve student outcomes.

2 Data

2.1 Inspection

The raw dataset [2][7] is sourced from Kaggle and contains 34 features predicting 3 target variables: 'Enrolled', 'Dropout', and 'Graduate'. The dataset includes a total of 4424 instances, with the respective number of instances per target variable being 794 for 'Enrolled', 1421 for 'Dropout', and 2209 for 'Graduate', as shown in Figure 1. Every instance in the dataset is anonymous and it is publicly available, therefore there are no ethical concerns or limitations in its use. The dataset contains no missing values and features a mix of binary, integer, and float values and target variables are represented as strings.

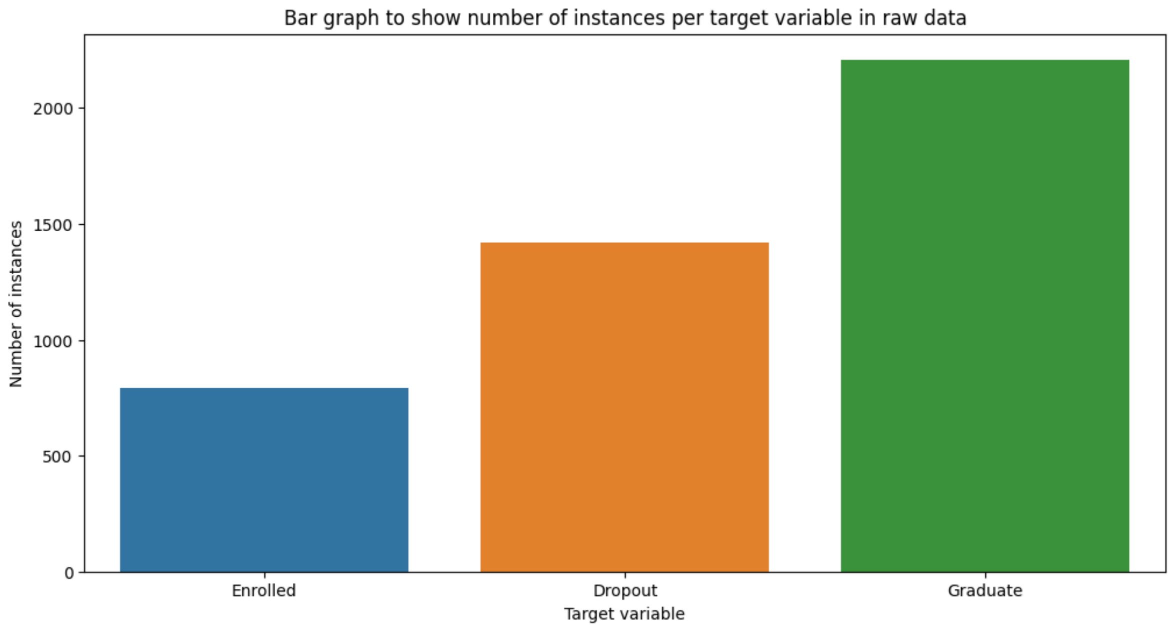

Figure 1: Bar graph to show number of instances per target variable in raw dataset.

2.2 Preprocessing

The following preprocessing steps were made using Python along with the packages Pandas, Scikit-learn (sklearn), and NumPy on a dataset stored in csv format. See Appendix 1 for scripts.

Delete negligent target variable
The research question of this paper aims to predict graduate success; therefore, the target variable ‘Enrolled’ was deemed negligent as an enrolled student has neither dropped out nor graduated. As a result, all instances in the dataset with the ’Enrolled’ target variable, totaling 794, were eliminated, reducing the total number of instances to 3630.

Feature elimination
In order to remove any irrelevant or corrupting features from the dataset, Recursive Feature Elimination [9] was used via the Scikit-learn (sklearn) package. The script ranked the features by significance and returned the bottom five ranking features for deletion, as illustrated in Appendix 1. However, no consistently negligent features were found and therefore all were deemed significant.

Undersampling
At this point in the preprocessing pipeline, the ratio of instances between ‘Dropout’ and ‘Graduate’ classes were 1421/3630 and 2209/3630, respectively. Given the class imbalance, the majority class (‘Graduate’) was undersampled in order to achieve a balanced distribution of instances, with a 50-50 split of 1421 instances for each class. This step was carried out to prevent potential errors in bias towards the majority class or overfitting the majority class.

Converting string values
As the machine learning methods in Python require numpy arrays as input, it was not feasible to use string values for the target variables. As a solution, these variables were converted into binary values, with 0 representing ’Dropout’ and 1 representing ’Graduate’.

Splitting training set and testing set

The final step was splitting the preprocessed dataset into training and test set with an 80:20 split respectively. This test set then stayed unchanged throughout the analysis and comparison of methods in order to ensure consistency in results.

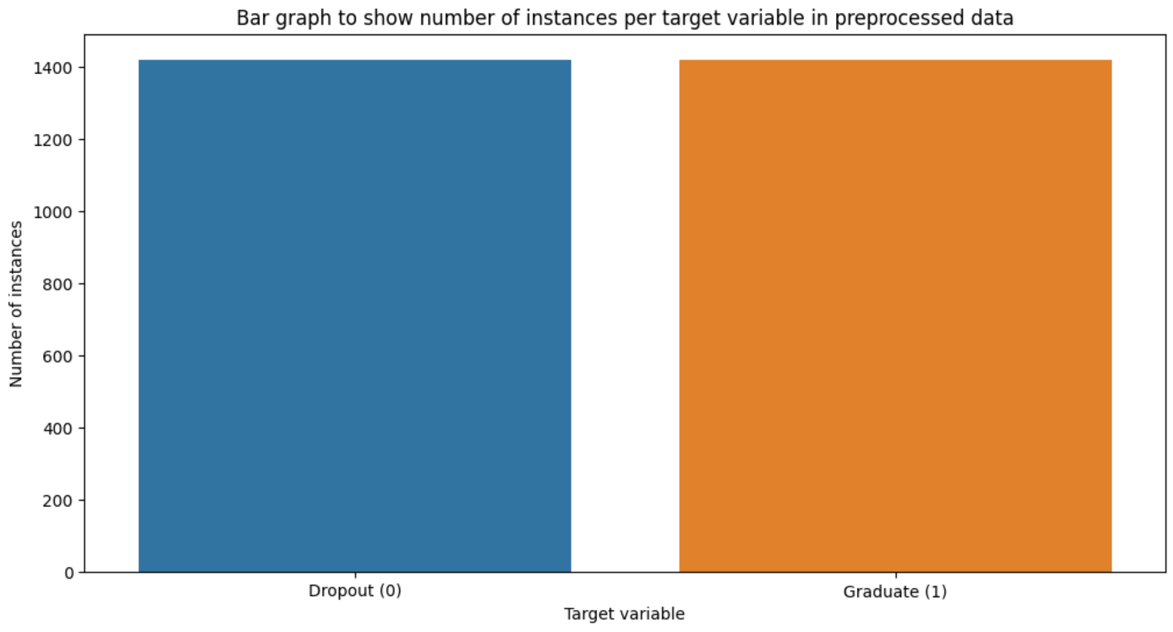

Figure 2: Bar graph to show number of instances per target variable in preprocessed dataset.

3 Methods

In this study, we explore the performance of twelve machine learning models in predicting students' graduation success rate. To ensure the best performance of each model, we optimize hyperparameters using GridSearchCV from sklearn and according to accuracy. Hyperparameters are model settings that are not learned from the data, but instead, are set prior to training the model. Examples of hyperparameters include learning rate, regularization strength, and number of hidden layers in a neural network. Accuracy is the decided performance metric because we have a balanced dataset [10] and therefore the model's ability to correctly classify both classes is equally as important. GridSearchCV is an optimization technique that can be used to find the optimal combination of hyperparameters for a given model, by systematically exploring a predefined grid of hyperparameters and evaluating the performance of the resulting models on a validation set. The majority of the models throughout these methods have been optimized via GridSearchCV.

3.1 K-Nearest Neighbour

K-Nearest Neighbour (kNN) is a supervised learning classifier [1], which uses proximity to make classification and predictions. Using the student graduate success dataset, that builds on student demographic and academic measures, kNN predicts a new instance's values after categorizing it to the k number of nearest neighbors based on their similarity. As kNN is a simple, yet effective algorithm as only the value of k can be changed and optimized for. To do so a brute force script was created, see Appendix 2, and the k value correlated with the highest accuracy was selected which was identified to be k=59.

3.2 Logistic Regression

Logistic regression is a statistical method used to model the relationship between a binary dependent variable and one or more independent variables. It is a type of generalized linear model that uses a logistic function to map the linear combination of the independent variables to a probability of the dependent variable taking a particular value. The logistic function, also known as the sigmoid function, takes any real-valued input and maps it to a value between 0 and 1, representing the probability of the dependent variable being 1. The model estimates the values of the coefficients that maximize the likelihood of observing the data given the model. This is achieved by minimizing the negative log-likelihood of the data, which is equivalent to maximizing the log-likelihood. It has several advantages over other classification methods. It is relatively robust to noise and outliers in the data, making it suitable for real-world applications. It can be efficiently trained using optimization algorithms, allowing it to handle large datasets. Additionally, The coefficients of the logistic regression model can be interpreted as the effect of the corresponding independent variables on the log-odds of the dependent variable, providing valuable insights into the relationships between the variables. This was optimized via GridSearchCV and the selected optimal values were C=0.001, maxiter=500, penalty='none', solver='sag'. Under these optimal values, the accuracy went from 0.89 to 0.90, whereas precision and recall stayed the same, so there was no significant improvement in the overall performance.

3.3 Naive Bayes

Naive Bayes algorithms are probability based algorithms, which build on Bayes' Theorem. The method assumes that the different features are independent of each other, and while this is not necessarily true in the real world, these algorithms usually present strong results. Moreover, Naive Bayes can be extremely fast compared to other methods. We have implemented our algorithm in the context of a Gaussian (normal) distribution [3], where the algorithm also assumes that the data follows a normal distribution. The algorithm without prior hyperparameter optimization produces relatively low results, with the accuracy of 0.8242. The only parameter that can be optimized is 'var-smoothing', which adds a value to the distributions variance, for which the default starting value is calculated from the training data set. This widens the distribution curve and allows values to be more spread out, even if they are more far away from the distribution mean. The optimal value for 'var-smoothing' was found to be 0.0001 with the GridSearchCV algorithm, which increased the accuracy to 0.8312.

3.4 Random Forest

Random Forest is a machine learning algorithm that belongs to the family of ensemble methods, which combine multiple individual models to make a more accurate prediction than any of the individual models alone. Random Forest is based on the concept of decision trees, which recursively partition the feature space into smaller regions to make a classification or regression prediction. However, unlike decision trees that can be prone to overfitting, Random Forest reduces the variance of the model by combining the predictions of multiple decision trees that are trained on randomly selected subsets of the training data.This randomness in the training data and feature selection helps to decorrelate the decision trees and make the model more robust to noise in the data. The performance of Random Forest depends on several hyperparameters, such as the number of decision trees in the ensemble, the maximum depth of each decision tree, and the minimum number of samples required to split an internal node. These hyperparameters can significantly affect the performance of the model, and therefore, it is essential to tune them appropriately to achieve the best performance on the given dataset. These hyperparameters are: n-estimators: the number of trees in the forest, max-features: the number of features to consider when looking for the best split, max-depth: the maximum depth of the tree, min-samples-split: the minimum number of samples required to split an internal node, min-samples-leaf: the minimum number of samples required to be at a leaf node and bootstrap: whether or not to use bootstrapping to sample the data. GridSearchCV found the optimal hyperparameters to be: n-estimators: 90, max-features: 'sqrt', max-depth: 4, min-samples-split: 2, min-samples-leaf: 1, bootstrap: True. This left the accuracy unchanged at 0.88.

3.5 Support Vector Machines

Support vector machines (SVMs) are another type of supervised learning algorithm. Through the training data, SVMs are trying to find the optimal hyperplane to separate the classes. The model uses a margin, which is the distance between the hyperplane and the closest data points from each class. SVMs can also be used for non-linear classification, where the model uses a kernel function, which turns the input data into a higher-dimensional space. In our implementation we used a basic linear kernel and a radial basis function (RBF) kernel. There were two hyperparameters that we tried to optimize, namely C and gamma. C defines the error tolerance of the margins (how many misclassifications are tolerable) and gamma, in the context of the RBF, defines how far the influence of a single training example reaches. GridSearchCV found the optimal values to be C=100, gamma=1 and kernel=linear which yielded an accuracy of 0.91.

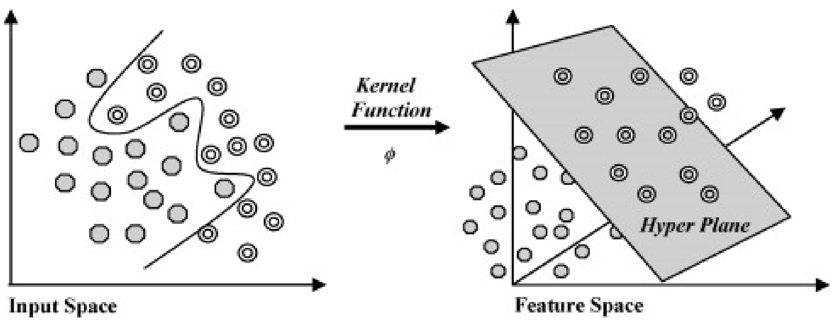

Figure 3: Visualisation of kernel function.

3.6 Ridge Regression

Ridge regression is a statistical method used in regression analysis to address the phenomenon where independent variables in a regression model are highly correlated with each other. This correlation leads to unstable and unreliable estimates of the regression coefficients, which can result in incorrect predictions and misleading interpretations. Ridge regression is a regularization technique that adds a penalty term to the sum of squared errors. The penalty term is proportional to the square of the magnitude of the regression coefficients, and it is controlled by a hyperparameter called the regularization parameter or shrinkage parameter, denoted by alpha. The intuition behind ridge regression is that the penalty term shrinks the regression coefficients towards zero, reducing their variance and making them more stable and reliable. However, the penalty also introduces bias into the estimates, which can lead to underfitting if the regularization parameter is too large. GridSearchCV found the optimal values to be alpha=1 which yielded an accuracy of 0.90.

3.7 Perceptron

A perceptron is a type of neural network, inspired by the processes of the human brain. It is a binary classifier that classifies data into two categories. It works by taking in an input vector and assigning weights to each of its elements. The input vector is then multiplied by the weight vector, and the result is passed through an activation function (ReLU, sigmoid, softMax etc.). The output of the activation function is the predicted class label of the input vector. During the training phase, the algorithm adjusts the weights of the input vector based on the error between the predicted class label and the actual class label. It keeps iterating until the weights are adjusted such that the error is minimized. By using the GridSearchCV function, the following optimal parameters were obtained : learning rate - 0.0001, tolerance - 1e-05. In this specific scenario, the accuracy of the perceptron algorithm went from 0.89 to 0.85. There are a few possible explanations to why this happened, such as overfitting during the initial training phase that can happen if the model is too complex or if the training data is not representative of the population. However, the decrease can be also a result of data variability or randomness. In order to improve the performance significantly, the perceptron can be transformed into a deep neural network (as explained in the next section). DNNs main advantage over perceptrons is their ability to learn more complex and abstract representations of the data. The multiple layers of interconnected neurons allow the DNN to capture more subtle patterns and relationships of the features. This makes DNNs particularly well-suited for handling high-dimensional datasets.

3.8 Decision Tree

Decision trees are a type of supervised learning algorithm that recursively partitions the feature space into smaller regions, with the goal of making accurate predictions for unseen data points. The algorithm works by selecting the best feature to split the data into subsets that are as homogeneous as possible with respect to the target variable. The homogeneity of the subsets is measured using a metric such as information gain, gain ratio, or Gini impurity. Information gain measures the reduction in entropy of the target variable after the split, while gain ratio normalizes the information gain by the intrinsic information of the feature. The splitting process continues until some stopping criterion is met. One of the advantages of decision trees is their interpretability. The structure of the tree can be easily visualized and understood, which makes it easier to interpret and explain the predictions made by the model. However, decision trees can be prone to overfitting, especially when the depth of the tree is too high, and the model becomes too complex. This problem can be addressed using pruning techniques such as reduced-error pruning, cost-complexity pruning, or minimum description length pruning. Several variants of decision trees have been developed to address some of their limitations. For example, Random Forest combines multiple decision trees trained on different subsets of the data and features, with the goal of reducing the variance of the model and improving its accuracy. Gradient Boosted Trees trains decision trees sequentially, with each tree trained on the residual error of the previous tree, with the goal of reducing the bias of the model and improving its accuracy. As you can see in the results, gradient boosting and random forest significantly increased accuracy. The hyperparameters of the decision tree were optimized using GridSearchCV which yielded optimal values of ccp-alpha=0.0, criterion='gini', max-depth=9, max-features='sqrt', min-samples-leaf=1, min-samples-split=2 and an accuracy of 0.82.

3.9 Gradient Boosting

Gradient Boosting is a decision tree based method, which builds on the previous tree by fitting residual errors to improve the next prediction. The algorithm starts out with a simple model, also known as a weak learner. The weak learner's residuals are calculated, and a new weak learner model is generated based on the residual and then added to the first learner. This repetition is making the model an ensemble one, and results in an improved performance of the model over time. Additionally, the possibility to fine tune Gradient Boosting with parameters is what makes it a powerful algorithm [4]. After using Grid Search Cross Validation, the optimal parameters were obtained: learning-rate of 0.1, max-depth of 4, max-features set to 'sqrt', min-samples-leaf of 2, min-samples-split of 2 and n-estimators set to 70. This yielded an accuracy score of 0.90.

3.10 Boostrap Aggregation

Bootstrap Aggregation, also known as Bagging, which trains on a dataset that has the same size as the original data set and is sampled with replacement from the original data set [6]. This means that samples are randomly pulled from the original dataset, and "put back", so that it could be chosen again. As a result, some samples are drawn multiple times, while others are not drawn at all. Bagging is also an ensemble method, meaning multiple runs of independently trained models are run to improve the performance metrics. What makes bagging a robust model, is that multiple models are trained on different subsets of the original dataset. After running Grid Search Cross Validation, the test for accuracy resulted in 0.8681 under the following learning parameters: bootstrap set as True, max-features set to 3, max-samples set to 0.5 and n-estimators set to 100.

3.11 Wide(r) and Deep(r) Neural Network

Artificial Neural Networks are a subset of machine learning methods. They are the successors of perceptrons. Their name and methodology comes from the view of a human brain as a network of interconnected neural pathways that process information. The high level overview of any neural network is that it consists of an input layer, one or more hidden layers, and lastly an output layer. A distinction can be made about NNs based on the ratio of hidden layers to nodes in a single layer. Let A,B be NNs. One says that A is wider NN than B if A has more nodes per layer than B. Similarly one says that B is a deeper NN than A if it contains a larger number of layers. Neural networks are particularly well suited to handle tasks which include complex patterns and nonlinear relationships between variables.

We implemented two types of networks. A relatively wide neural network which consists of one hidden layer, with 102 neurons as well as a comparatively deeper neural network with 4 hidden layers .

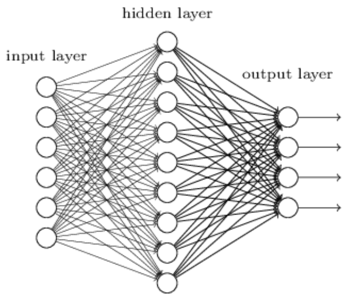

Figure 4: Wide(r) Neural Network.

Since the problem we are trying to solve is not linearly separable the literature we read suggested using at least one hidden layer. However, by Cybenko's Universal Approximation Theorem we hypothesized that the problem could be solved well by just one layer with many nodes. This hypothesis was rectified when the classification task was completed with relatively high accuracy. When determining the number of nodes, we relied on manual testing as well as the general 'rule of thumbs' concerning wide neural networks. After some testing we settled on 3x the number of inputs for the wider network. For the deeper network the main question is how many layers to include. The methodology we used to get our answer is outlined perfectly by the quote from Yoshua Bengio: "Very simple. Just keep adding layers until the test error does not improve anymore ". We ended up choosing 3 hidden layers each containing the same number of nodes as the count of inputs.

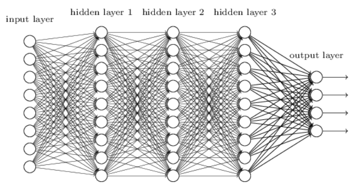

Figure 5: Deep(r) Neural Network.

For both networks we preferred the rectified linear activation function in the input and hidden layers. We used the sigmoid function for the output function for the deeper neural net. The sigmoid function maps the output value between 0 and 1. Essentially it gives a probability for which class does the current instance in question belongs to. The threshold value for the activation function 0.5, which finely separates the classes. This provides a simple, interpretable output.

4 Results

The following shows the results of the experiments and a confusion matrix for Support Vector Machines: the model which output the greatest accuracy on the test dataset.

Method	Accuracy
Bagging	0.855888
Decision Tree	0.822496
Gradient Boosting	0.892794
KNN	0.892794
Logistic Regression	0.903339
Naive Bayes	0.831283
Perceptron	0.859402
Random Forest	0.876977
Ridge	0.899824
SVM	0.910369
NN-Wide	0.905097
NN-Deep	0.89806

Table 1: Machine learning methods with corresponding accuracy on test dataset.

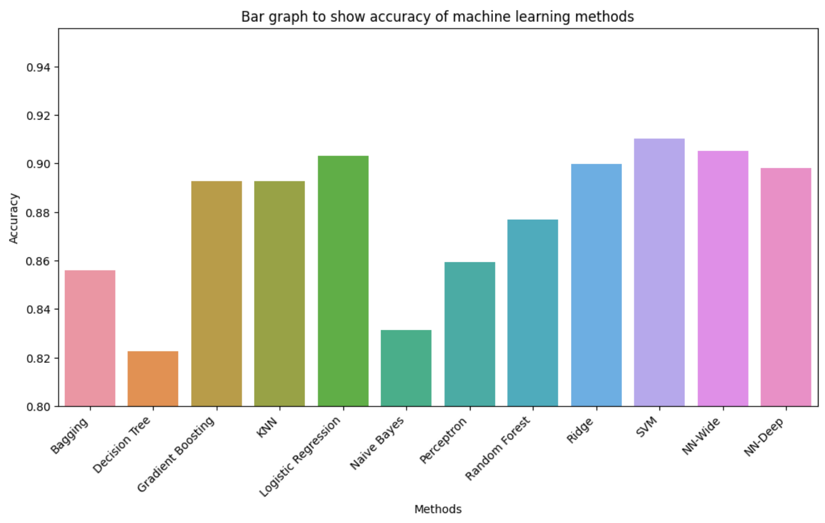

Figure 6: Bar graph to show accuracy of machine learning methods.

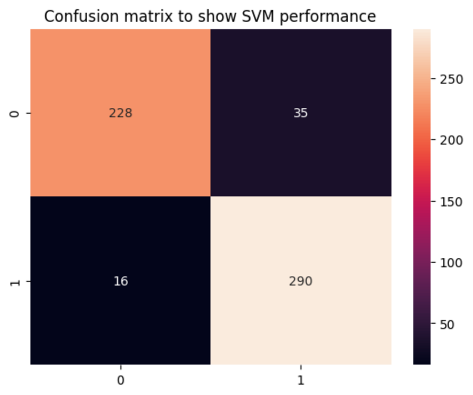

Figure 7: Confusion matrix to show SVM performance (best performing model).

5 Evaluation

Due to this study being a binary classification problem where the dataset is balanced ('Dropout': 1421 instances, 'Graduate': 1421 instances) and the costs of false positives and false negatives are relatively equal, accuracy was chosen as the suitable performance metric [10].The results indicated that the choice of method significantly influenced the prediction of student graduate success.

The SVM model performed the best with an accuracy of 0.910369, followed by Neural Network (wider) with an accuracy of 0.905097 and Logistic Regression with an accuracy of 0.903339. On the other hand Naive Bayes had the lowest accuracy of 0.831283. These results supported our hypothesis that SVM and logistic regression methods would perform well due to the high dimensionality of the data, as reported in previous studies [8].

However, the dataset's high dimensionality with 34 features could have negatively impacted other machine learning methods' performance. High dimensionality can lead to challenges for machine learning models due to the increased number of possible feature combinations, making it more difficult to identify patterns. Furthermore, there is a greater risk of overfitting the data by creating an overly complex model that fails to perform well on unseen data. Thus, better results may have been achieved by using other feature elimination techniques such as Principal Component Analysis or Regularisation in the preprocessing stage, in addition to Recursive Feature Elimination.

In addition, the ensemble methods random forest, bagging, and gradient boosting outperformed the base model of decision trees. This was due to the fact that decision trees can overfit the data, while ensemble methods can reduce variance by combining multiple models. Moreover, ensemble methods can capture non-linear relationships between features, which is beneficial when dealing with high-dimensional data. These results further supported our hypothesis.

6 Conclusion

In conclusion, this study applied machine learning techniques to predict student graduate success and determine the most accurate model for doing so. The results were in accordance with our hypothesis and showed that SVM outperformed other models with an accuracy of 0.91, followed by Neural Network (wider) and Logistic Regression. These findings demonstrate that graduate success can be predicted using machine learning techniques.

The implications of these findings are significant for universities seeking to improve student graduate success by identifying students who may be at risk of dropping out and providing them with the necessary resources to succeed. However, the limitations of this study: the dataset's lack of representativeness and the inability to remove features, should be taken into account when interpreting the results.

This study contributes to the growing body of literature surrounding machine learning in education and provides insights for predicting student graduate success. Future research could investigate other feature selection techniques for datasets of this type and explore the application of machine learning to other student outcomes, such as academic performance or career success.

7 References

(1) What is the k-nearest neighbors algorithm? — IBM. (n.d.). What Is the K-nearest Neighbors Algorithm? — IBM.
(2) Realinho, V.; Machado, J.; Baptista, L.; Martins, M.V. Predicting Student Dropout and Academic Success. Data 2022, 7, 146.
(3) Harry, Z., The Optimality of Naive Bayes. University of New Brunswick.
(4) Biau, G., Cadre, B. (2021). Optimization by Gradient Boosting. In: Daouia, A., Ruiz-Gazen, A. (eds) Advances in Contemporary Statistics and Econometrics. Springer, Cham.
(5) Chih-Wei H.; Chih-Chung C.; Chih-Jen L., A Practical Guide to Support Vector Classification, National Taiwan University 2016.
(6) Injadat, M., Moubayed, A., Nassif, A. B., Shami, A. (2020, July 22). Multi-split optimized bagging ensemble model selection for multi-class educational data mining - Applied Intelligence. SpringerLink.
(7) Kaggle, Predicting Student Dropout and Academic Success Dataset, 2022.

(8) Hongshik Ahn, Hojin Moon, Melissa J.Fazzari, Noha Lim, James J Chen, Ralph L Kodell, Classification by ensembles from random partitions of high-dimensional data, 2007.
(9) Xue-wen Chen, Jong Cheol Jeong, Enhanced recursive feature elimination, 2007.
(10) Amalia Luque, Alejandro Carrasco, Alejandro Martín, Ana de las Heras, The impact of class imbalance in classification performance metrics based on the binary confusion matrix, 2019.

8 Appendix

8.1 Data Preprocessing

```
# 1. Read original dataset.
import pandas as pd
df = pd.read_csv("data/dataset.csv")
df.head()

#2. Delete all targets 'enrolled' not relevant class for us.
df2 =  df[df.Target != "Enrolled"]
df2 = df2.reset_index(drop=True)
df2.head()

#3. Recursive Feature Elimination
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier #have to use some estimator
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Perceptron
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

#input split
x=df2.drop(columns=["Target"], axis=1)
y=df2["Target"]

#estimators
l=[DecisionTreeClassifier(), LogisticRegression(), Perceptron(),\\
RandomForestClassifier(),
GradientBoostingClassifier()]

for i in range(0,5):
    rfe=RFE(estimator=l[i], n_features_to_select=29)
    rfe.fit(x,y)
    columns_to_delete=[]
    for i, col in zip(range(x.shape[1]), x.columns):
        # print(str(rfe.support_[i]))
        if (str(rfe.support_[i])) == str(False):
            columns_to_delete.append(col)
    print(sorted(columns_to_delete))

# df3=df2.drop(columns=columns_to_delete)
# df3.head()

'''
No consistent negligent features found...

#4. Undersample to prevent errors due to class imbalance
#Graduate: 2209/3630
#Dropout: 1421/3630
```

```
#Delete (2209-1421=788) Graduate data
import random
graduate-df=df2[df2.Target != "Dropout"]
indexes=list(graduate-df.index.values)
# print(len(indexes))
indexes-to-delete=[]

for i in range(788):
    l=len(indexes)
    n=random.randint(0,l-1)
    indexes-to-delete.append(indexes.pop(n))
# print(len(indexes-to-delete))
# print(indexes-to-delete)
df3=df2.drop(indexes-to-delete, axis='index')
df3 = df3.reset-index(drop=True)

graduate-df=df3[df3.Target != "Dropout"]
dropout-df=df3[df3.Target != "Graduate"]

graduate-row=graduate-df.shape[0]
dropout-row=dropout-df.shape[0]
total-rows=df3.shape[0]

print('Balanced Classes.')

#5. Change, dropout, graduate to Boolean: ready for numpy.
df4=df3.replace({'Target':{'Dropout': 0, 'Graduate': 1}})
df4.head()
df4.to-csv('Data/preprocessed-dataset.csv')

#6. Make test and training sets.

import numpy as np
data = np.loadtxt(('Data/preprocessed-dataset.csv'), delimiter=',', skiprows=1)
x = data[:,1:35]
y = data[:,35]

from sklearn.model-selection import train-test-split
x-train, x-test, y-train, y-test = train-test-split(x, y, test-size=0.2)
```

8.2 Methods

Only kNN and decision tree methods are shown here as loading model and optimisation with GridSearchCV repetitive.

```
#KNN
from sklearn.neighbors import KNeighborsClassifier

# Create a kNN classifier
knn = KNeighborsClassifier(5) # The number of neighbors

# Use cross-validation to evaluate the performance of the model
scores = cross-val-score(knn, x-train, y-train, cv=10)
print("Cross-validation␣scores:", scores)
print("Mean␣score:", scores.mean())

# Train the model on the entire training set
```

```
knn.fit(x_train, y_train)

# Make predicitons on the test set
knn_pred = knn.predict(x_test)

# Search for the best k-value for accuracy, precision, recall and AUC-ROC

best_acc = 0
best_acc_for = 0
best_prec = 0
best_prec_for = 0
best_recall = 0
best_recall_for = 0
best_roc_auc=0
best_roc_auc_for=0

for i in range(1,150):
    knn = KNeighborsClassifier(i) # Iterating over the number of neighbors
    knn.fit(x_train, y_train)
    knn_pred = knn.predict(x_test)
    if metrics.roc_auc_score(y_test, knn_pred) > best_roc_auc:
        best_roc_auc_for = i
        best_roc_auc = metrics.roc_auc_score(y_test, knn_pred)
    if metrics.accuracy_score(y_test, knn_pred) > best_acc:
        best_acc_for = i
        best_acc = metrics.accuracy_score(y_test, knn_pred)
    if metrics.precision_score(y_test, knn_pred) > best_prec:
        best_prec_for = i
        best_prec = metrics.precision_score(y_test, knn_pred)
    if metrics.recall_score(y_test, knn_pred) > best_recall:
        best_recall_for = i
        best_recall = metrics.recall_score(y_test, knn_pred)

#Decision Tree

from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree = dtree.fit(x_train, y_train)

y_predict=dtree.predict(x_test)

from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix

def model_info(model_pred):
    print("Accuracy:", metrics.accuracy_score(y_test, model_pred))
    print("Precision:", metrics.precision_score(y_test, model_pred))
    print("Recall:", metrics.recall_score(y_test, model_pred), end="\n\n")
    print(classification_report(y_test, model_pred))

model_info(y_predict)

from sklearn.model_selection import GridSearchCV

# Applying grid search for the linear model
# This block is reall resource heavy as we have a quite big grid to search
```

```
tree_parameters = {'criterion' : ['gini', 'entropy'],
                   'max_depth' : [i for i in range(1, 15, 2)],
                     'max_features' : ['auto','sqrt'],
                     'min_samples_split' : [2, 3, 5],
                     'min_samples_leaf' : [1, 2, 3]
                     }
model_grid = GridSearchCV(dtree, tree_parameters, refit = True, verbose = 3, n_jobs=2)
model_grid.fit(x_train, y_train)
grid_pred = model_grid.predict(x_test)

model_info(grid_pred)
model_grid.best_params_

from sklearn.metrics import accuracy_score, precision_score, recall_score, \\
confusion_matrix, roc_auc_score

dtree_model = DecisionTreeClassifier(ccp_alpha=0.0, criterion='gini', \\
max_depth=9, max_features='sqrt', min_samples_leaf=1, min_samples_split=2)
dtree_model.fit(x_train, y_train)
y_predict = dtree_model.predict(x_test)

model_info(y_predict)
```

8.3 Data Visualisation

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
method = ["Bagging", "Decision Tree", "Gradient Boosting", "KNN",\\
"Logistic Regression", "Naive Bayes", "Perceptron", "Random Forest",\\
"Ridge", "SVM", "NN-Wide", "NN-Deep"]
accuracy = [b_acc, dtree_acc, gb_acc, knn_acc, logr_acc,\\
nb_acc, p_acc, rf_acc, r_acc, svm_acc, wd_nn, d_nn]
ax.bar(method,accuracy)
plt.show()

# initialise data of lists.
data = {'Method':method, 'Accuracy':accuracy}

# Create DataFrame
df = pd.DataFrame(data)
df

fig, ax = plt.subplots(figsize=(12, 6))
sns.barplot(data=data, x="Method", y="Accuracy",\\
ax=ax).set(title='Bar \\
graph to show accuracy of machine learning methods', xlabel='Methods',\\
ylabel='Accuracy')
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right')
plt.ylim(0.8)
plt.figure()

fig, ax = plt.subplots(figsize=(12, 6))
data={"Target Variable":["Enrolled", "Dropout", "Graduate"], "Instances":\\
[794, 1421,2209]}
```

```
sns.barplot(data=data, x="Target␣Variable",\\
y="Instances", ax=ax).set(title='Bar␣graph␣to␣show␣number\\
of␣instances␣per␣target␣variable␣in␣raw␣data', xlabel='Target␣variable',\\
ylabel='Number␣of␣instances')
# ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right')
plt.figure()

fig, ax = plt.subplots(figsize=(12, 6))
data={"Target␣Variable":["Dropout␣(0)", "Graduate␣(1)"], "Instances": [1421, 1421]}
sns.barplot(data=data, x="Target␣Variable",\\
y="Instances", ax=ax).set(title='Bar␣graph␣to␣show␣number␣\\
of␣instances␣per␣target␣variable␣in␣preprocessed␣data', xlabel='Target␣variable',\\
ylabel='Number␣of␣instances')
# ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right')
plt.figure()

array = [[int(228),35], [int(16),290]]

DetaFrame_cm = pd.DataFrame(array)
sns.heatmap(array, annot=True, fmt='g').set(title=\\
"Confusion␣matrix␣to␣show␣SVM␣performance")
plt.show()
```