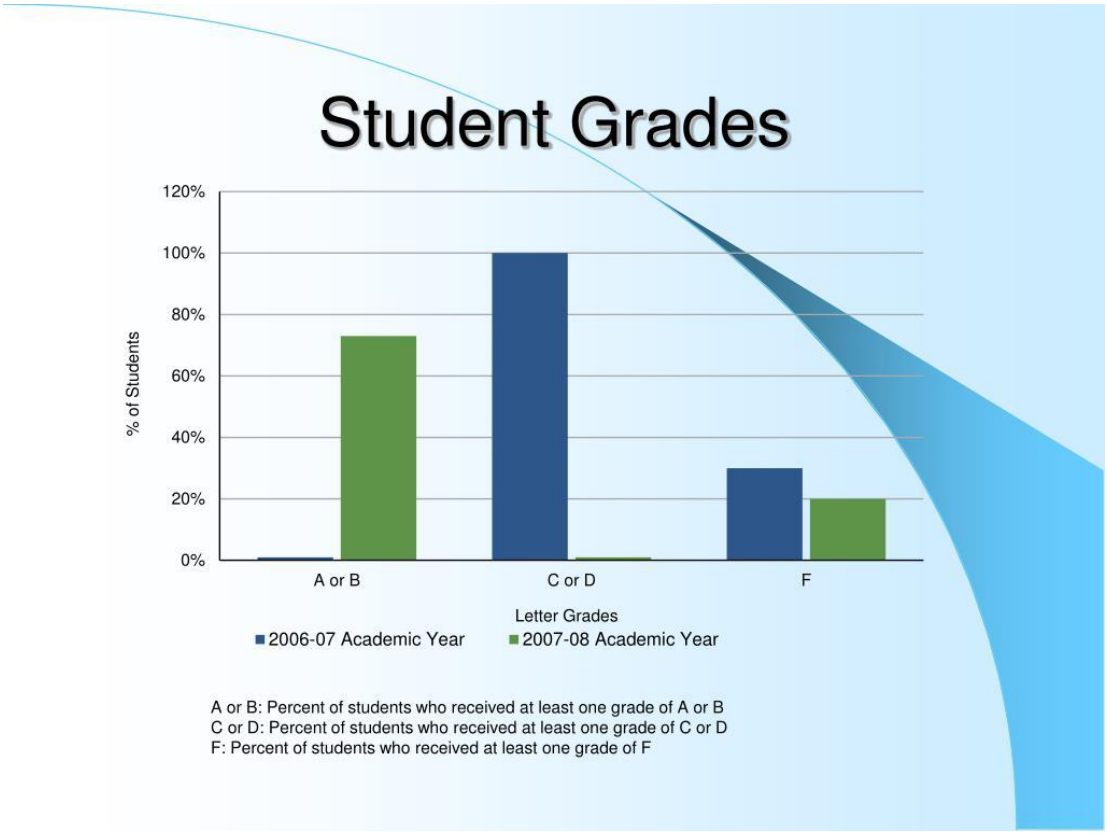
Student Grade Prediction

A Machine Learning Project



Divyanshi Gautam(20ucs066)

Ayush Jain(20ucc028)

15-05-2023

6TH SEM - ML

INTRODUCTION

In recent years, the use of educational data mining and machine learning techniques in higher education has become increasingly popular. These techniques can help educators analyze large amounts of data, gain insights into student behavior, and make informed decisions about how to improve learning outcomes.

One area where these techniques have shown great promise is in predicting student success, including the prediction of final grades. The proposed method in this research paper builds on previous work in the field and offers a new approach to predicting student final grades. By clustering students based on their experience points, our method can better tailor support to different groups of students.

Additionally, by balancing the cluster sizes, we can ensure that each cluster is given equal priority and improve the accuracy of our prediction algorithms. One of the strengths of our method is that it relies solely on performance data from the current semester. This means that we can make predictions early on in the course, allowing for timely intervention and support if needed. The use of feature selection techniques also helps to improve the efficiency and accuracy of our model, making it a valuable tool for both students and instructors. The results of our study demonstrate that our method outperforms other approaches while achieving a high level of accuracy. This suggests that our method has the potential to be widely adopted in higher education settings and can help to improve student retention and learning outcomes..

Following classifiers have been chosen for comparison in this project:

- Random Forest
- Support Vector Machine
- K-Nearest Neighbours
- Naive Bayes
- Artificial Neural Network

Performance Metrices

- **Precision**

Precision is calculated by dividing the number of correct positive results by the total number of positive results returned by the classifier.

- **Recall**

Recall is defined as the number of correct positive results divided by the total number of relevant samples (all samples that should have been identified as positive).

- **F1-score**

The F1-score quantifies a model's accuracy on a dataset. To compute the score, it takes into account both the precision p and the recall r of the test. It is the arithmetic mean of precision and recall.

- **Confusion Matrix**

The F1-score is a measure of a model's accuracy on a dataset. It computes the score by taking into account both the precision p and the recall r of the test. It is the harmonic average of precision and recall.

- **Accuracy**

The number of correctly predicted data points out of all data points is referred to as accuracy. It is the number of true positives and true negatives divided by the total number of true positives, true negatives, false positives, and false negatives.

Prediction Approach

Clustering Students

In this study, the students' XP data is utilized to cluster them into distinct groups, allowing for a tailored gamified experience based on their characteristics. The XP data is particularly informative as it is used in calculating student grades. The k-means clustering algorithm is chosen for its simplicity and efficiency. Initially, the students are clustered into four groups, as justified in previous research. In cases where a cluster contains a small number of students, three clusters (good, average, and naive students) are generated. The clustering algorithm presented follows the k-means approach.

Algorithm 1: Our Clustering Algorithm.

Input: All students S_a, XP data.
Output: Students Clusters.

$C_a \leftarrow$ Divide S_a into 4 groups using XP data via a k-means.
if Size of a Cluster in C_a $<= 2$ **then**
    $C_a \leftarrow$ Divide S_a into 3 groups using XP and a k-means.
Return C_a

```
"""
Algorithm
Ca Divide Sa into 4 groups using XP data via a k-means.
2: if Size of a Cluster in Ca <= 2 then
3: Ca Divide Sa into 3 groups using XP and a k-means.
4: Return C
"""
from sklearn.cluster import KMeans
def cluster(clust):
  global kmeans
  kmeans = KMeans(n_clusters=clust, random_state=0)
  kmeans.fit(data[numerical])
  dataframe=pd.DataFrame()
  dataframe['kmean'] = kmeans.labels_
  arr=dataframe['kmean'].value_counts()
  print(arr)
  for i in arr:
    if i <=2:
      clust=3
      cluster(clust)
  return arr
clust=4
array=cluster(clust)
```

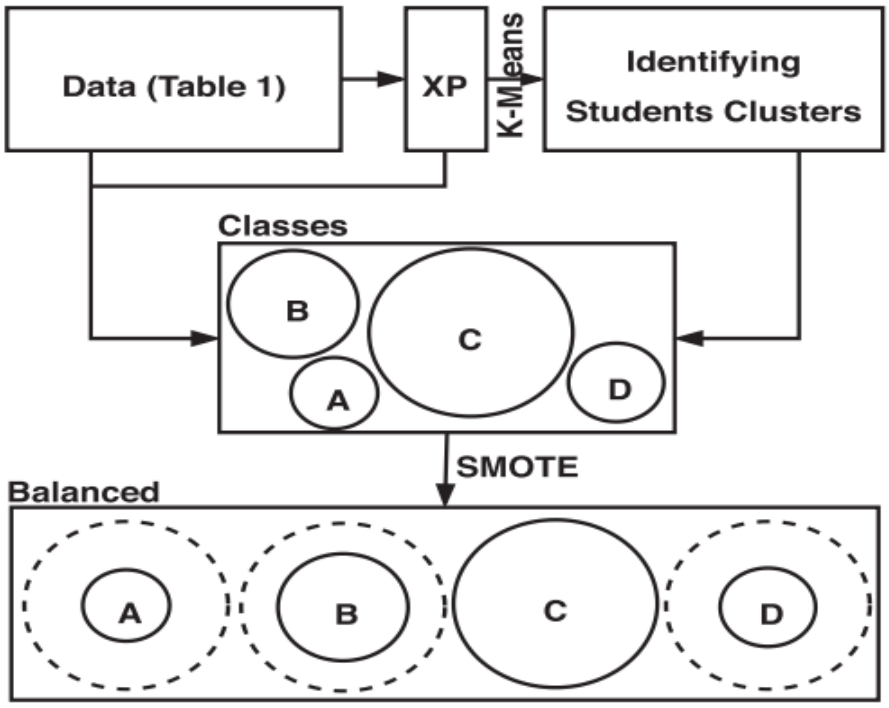
Generating Virtual Students

To address the issue of highly imbalanced clusters observed after clustering the students, a resampling approach is employed to improve the performance of prediction models. Specifically, a utility-based learning (UBL) tool, as proposed in, is utilized. This tool leverages a synthetic minority oversampling technique (SMOTE) to enhance prediction accuracy for the minority clusters. SMOTE generates synthetic instances for the minor clusters by randomly selecting k members from each cluster and oversampling accordingly, thereby balancing the clusters and mitigating the impact of class imbalance on predictive models.

Algorithm 2: Virtual Students Generation.

Input: All students S_a, All data D, C_a from Alg 1.
Output: Balanced clusters using virtual students.

1: **for** *(1 to i (number of clusters))* **do**
2: $L_i \leftarrow$ Size C_{a_i};
3: $L_b \leftarrow$ Max L_i;
4: $L_s \leftarrow$ Min L_i;
5: **for** *(1 to i (number of clusters))* **do**
6: $R_i \leftarrow round(\frac{L_i}{L_b})$; ▷R is a ratio that shows how much a cluster needs to enlarge.
7: $n \leftarrow L_s - 1$;
8: $Class_B \leftarrow$ SMOTE $(D, R, dist ='' Euclidean,'' k = n)$;
9: **Return** $Clus_B$;



Running flow of our balancing approach

```
from imblearn.over_sampling import SMOTE
from collections import Counter
dataframe=pd.DataFrame()
# dataframe['data_index'] = data["school"]
dataframe['cluster'] = kmeans.labels_
dataframe[dataframe.cluster == 3]
# print(Counter(data["target"]))


# SMOTE : Synthetic Minority Oversampling Technique

maxi=max(array)
mini=min(array)
val=[]
for i in range(clust):
  val=round(maxi/mini)
n=mini-1
sm = SMOTE(random_state=val, k_neighbors=7)
data[numerical], data["target"] = sm.fit_resample(data[numerical], data["target"])
Counter(data["target"])
```

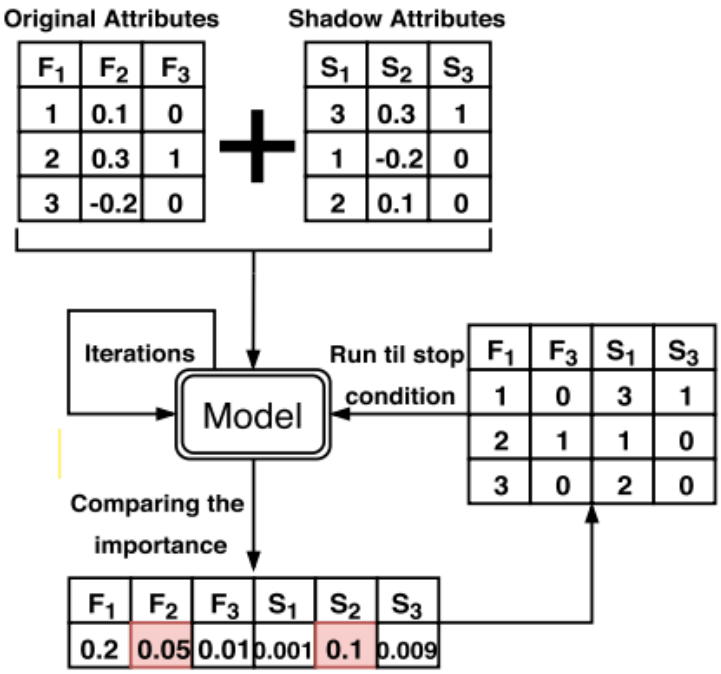
```
Counter({0: 59665, 3: 59714, 2: 19600, 1: 22424})
```

Data Attribute Selection

Attribute selection is a technique used to identify important and relevant attributes while discarding redundant and irrelevant ones. It improves learning speed, reduces storage volume, and minimizes noise in the data.

All attribute selection algorithms fall in two categories: filter and wrapper. Wrappers assess attributes by means of estimated accuracy provided by a target algorithm. Filters apply general data characteristics for attribute assessment, and it works separately from any learning algorithm.

The Boruta algorithm, a wrapper method employing Random Forest, is utilized in this study. It compares the importance of original attributes with shadow attributes generated by shuffling them. Attributes with lower importance are dropped, while higher ones are confirmed. The process iterates until no confirmed attributes remain or a stopping criterion is met. Boruta has been widely used in Kaggle competitions for attribute selection.




```python
from boruta import BorutaPy
import numpy as np
from sklearn.ensemble import RandomForestClassifier
tree = RandomForestClassifier(n_estimators = 1000,max_depth=5, random_state=42)
feat_selector = BorutaPy(
    verbose=2,
    estimator=tree,
    n_estimators='auto',
    max_iter=10
)

feat_selector.fit(np.array(X_train), np.array(y_train))
print("Testing with shadow features")
for i in range(len(feat_selector.support_)):
    if feat_selector.support_[i]:
        print("Important Features: ", X_train.columns[i],
              " - Ranking: ", feat_selector.ranking_[i])
    else:
        print("Features to be rmoved: ",
              X_train.columns[i], " - Ranking: ", feat_selector.ranking_[i])
```

```
Iteration:      1 / 10
Confirmed:      0
Tentative:      8
Rejected:       0
Iteration:      2 / 10
Confirmed:      0
Tentative:      8
Rejected:       0
Iteration:      3 / 10
Confirmed:      0
```

Grade Prediction Model

After identifying the informative attributes and dropping the irrelevant ones from the data, we have used the data to predict the final grades of the students. For that, we have employed the regular form of three predictive algorithms:

1) **Naive Bayes (NB)** is a classification algorithm that makes use of the Bayesian theorem and assumes that all data attributes are independent. NB has several advantages, including a simple design process, requiring only a small amount of data for training, being computationally fast, and providing probability-based results that are easier to interpret and use for various tasks.

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
gnb.get_params()
y_pred = gnb.predict(X_test)
print("Training Accuracy: ", gnb.score( X_train, y_train))
print("Testing Accuracy: ", gnb.score(X_test, y_test))
```

```
Training Accuracy:  0.9188189269087111
Testing Accuracy:  0.9163586047376139
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(cm)
disp.plot()
```

2) **Random Forests (RF)** are a type of ensemble learning algorithm that combines multiple decision tree predictors. The trees are built using randomly sampled values from the same distribution. RF has several advantages, including high classification accuracy, ability to model complex interactions among predictor attributes, ability to

handle large amounts of data attributes, reduction of overfitting, ability to train trees in parallel, and insensitivity to missing data due to random sampling. RF is widely used by researchers in various fields.

```
[23] from sklearn import metrics
     from sklearn.ensemble import RandomForestClassifier
     model = RandomForestClassifier(n_estimators=1000)
     model.fit(X_train, y_train)
     y_pred = model.predict(X_test)
     print(metrics.classification_report(y_pred, y_test))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	17804
1	1.00	1.00	1.00	6690
2	1.00	1.00	1.00	5892
3	1.00	1.00	1.00	18035
accuracy			1.00	48421
macro avg	1.00	1.00	1.00	48421
weighted avg	1.00	1.00	1.00	48421

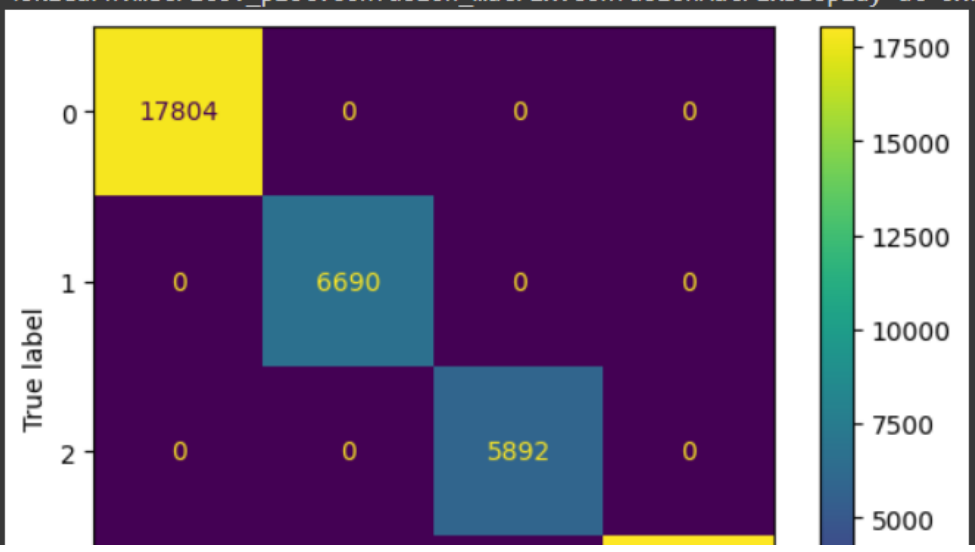
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
mat = confusion_matrix(y_test, y_pred)
# sns.heatmap(mat.T, square=True, annot=True, fmt='d',
#             cbar=False, cmap='Blues')
# plt.xlabel('true label')
# plt.ylabel('predicted label');
disp = ConfusionMatrixDisplay(mat)
disp.plot()
```

3) **K-Nearest Neighbor (KNN)** is a simple and effective classification approach that is suitable for unknown data distributions. It works by classifying an instance based on a majority vote of its K nearest neighbors, using a distance function. KNN is advantageous

due to its simplicity, ease of implementation and debugging, availability of noise reduction techniques, and ability to provide output clarification by analyzing neighbors. KNN is a valuable tool for classification tasks and should be considered as a first choice for dealing with unknown data distributions.

```
### K-Nearest Neighbor

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=4)
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

knn.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy is : ", knn.score(X_test, y_test))
knn_mat = confusion_matrix(y_test, y_pred)
disp_knn = ConfusionMatrixDisplay(knn_mat)
disp_knn.plot()
```

```
Accuracy is :  0.9801325871006381
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f303f2df280>
```



Novelty Component Added: We have made value of k dynamic in KNN

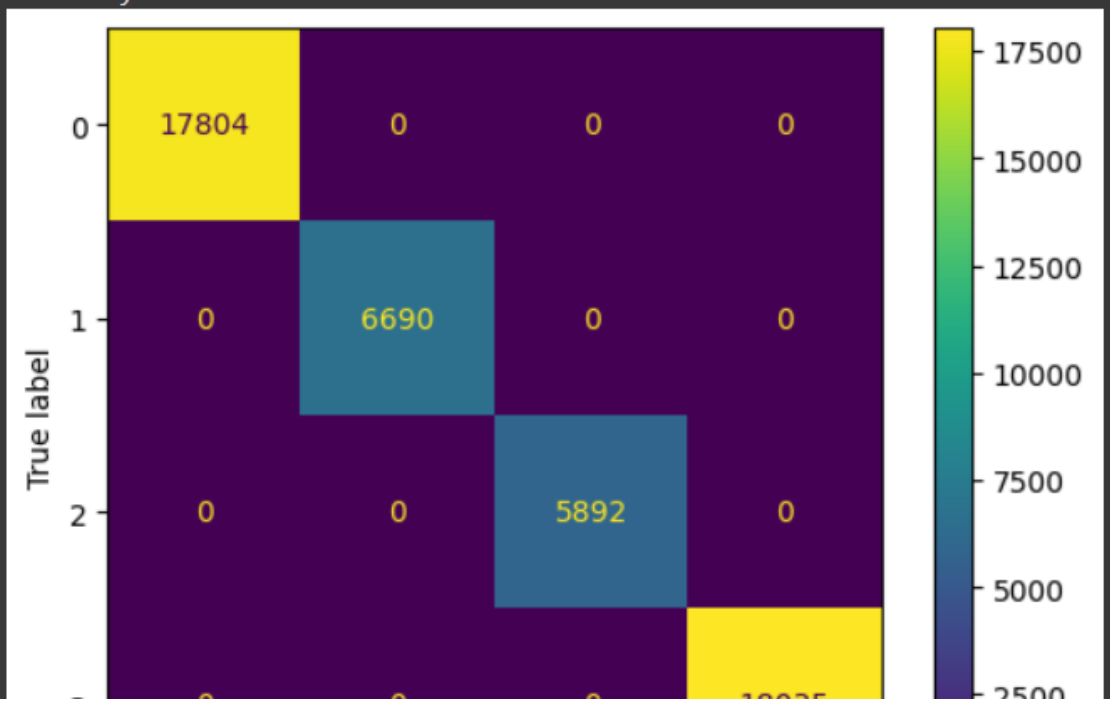
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# get the input value of k from the user
k = int(input("Enter the value of k: "))

# create a KNN classifier object
knn = KNeighborsClassifier(n_neighbors=k)

# fit the model on the training data
knn.fit(X_train, y_train)

# predict the grades for the test data
y_pred = knn.predict(X_test)

knn_mat = confusion_matrix(y_test, y_pred)
disp_knn = ConfusionMatrixDisplay(knn_mat)
disp_knn.plot()
```

```python
import numpy as np
from scipy.spatial.distance import cdist

def knn_predict(X_train, y_train, X_test, k):
    # Calculate distances between test points and training points
    distances = cdist(X_test, X_train)

    # Find indices of k nearest neighbors for each test point
    knn_indices = np.argsort(distances)[:, :k]

    # Get labels of k nearest neighbors for each test point
    knn_labels = y_train[knn_indices]

    # Make predictions based on majority vote of k nearest neighbors
    predictions = np.apply_along_axis(lambda x: np.bincount(x).argmax(), axis=1, arr=knn_labels)

    return predictions
```

4) **The support vector machine (SVM)** is a widely used machine learning algorithm for classification and regression analysis. It constructs a hyperplane or a set of hyperplanes in a high-dimensional space that can be used for classification or regression. SVM has several advantages, including good generalization ability, effectiveness in high-dimensional spaces, and the ability to handle non-linear decision boundaries through kernel functions. Additionally, it has been successfully applied in various fields such as image recognition, text classification, and bioinformatics.

```
model = SVC(kernel='linear')
model.fit(X_train, y_train)
predictions = model.predict(X_test)
percentage = model.score(X_test, y_test)

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
res = confusion_matrix(y_test, predictions)
disp_svm = ConfusionMatrixDisplay(res)
disp_svm.plot()
print(f"Test Set: {len(X_test)}")
print(f"Accuracy = {percentage*100} %")
```

```
Test Set: 48421
Accuracy = 100.0 %
```



5) **The Artificial Neural Network (ANN)** is a computational model that is inspired by the structure and function of the biological neural networks of the human brain. ANNs are capable of learning from data, recognizing patterns, and making predictions. The structure of an ANN consists of layers of interconnected artificial neurons, where each neuron receives input signals, performs computations, and passes the output to the next layer. The strength of the connections between the neurons is determined by the weights, which are adjusted during the learning process to minimize the error between the predicted output and the actual output.

```python
# Import Library
from sklearn.neural_network import MLPClassifier

# Fitting Model
mlp = MLPClassifier(hidden_layer_sizes=(5), activation = 'relu', solver = 'adam',max_iter= 10000, verbose = True)
mlp = mlp.fit(X_train,y_train)

# Prediction to Test Dataset
y_predmlp = mlp.predict(X_test)
```

```python
[29] print('Number of Layer =', mlp.n_layers_)
     print('Number of Iteration =', mlp.n_iter_)
     print('Current loss computed with the loss function =', mlp.loss_)
```

```
Number of Layer = 3
Number of Iteration = 122
Current loss computed with the loss function = 0.0005450877540868083
```

```python
# Import the metrics class
from sklearn import metrics
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay


# Confussion Matrix
cnf_matrixmlp = confusion_matrix(y_test, y_predmlp)
disp_ann = ConfusionMatrixDisplay(cnf_matrixmlp)
disp_ann.plot()
```

The final grades are calculated based on a 20-points grading system, we came up with the following categories:

$$For\ Each\ Final\ Grade\ (G) = \begin{cases} \textbf{if}\ \ 18 \leqslant G\ \leqslant 20 \rightarrow A \\ \textbf{if}\ \ 14 \leqslant G\ < 18 \rightarrow B \\ \textbf{if}\ \ 10 \leqslant G\ < 14 \rightarrow C \\ \textbf{if}\ \ \ 0 \leqslant G\ < 10 \rightarrow D\ (Failed). \end{cases}$$

```
avg=data["# Level 1"]
# avg=[i for i in data["# Level 1"]]
print(avg)
# avg = [int(i) for i in avg]
avg2=data["# Level 2"]
print(avg2)
# avg2 = [int(float(i)) for i in avg2]
avg3=data["# Level 3"]
# avg3 = [int(float(i)) for i in avg3]
avg4=data["% Level 3"]
# avg4 = [int(float(i)) for i in avg4]
zz=zip(avg,avg2,avg3,avg4)
final_avg=[round(sum(i)/4) for i in zz]
data["avg"]=final_avg
numerical.append("avg")
grade_target=[]
for i in final_avg:
  if i>=0 and i<10:
    grade_target.append(0)
  elif i>=10 and i<14:
    grade_target.append(1)
  elif i>=14 and i<18:
    grade_target.append(2)
  else:
    grade_target.append(3)
data["target"]=grade_target
data.head()
```

EVALUATION AND DISCUSSION

For each academic year, we have considered five subsets, accounting for a different time span .We compared the final grades of the students in different years. As shown in this figure, the average grades were around 15. In addition, there were several students that failed the course mostly after the year 2015–2016.

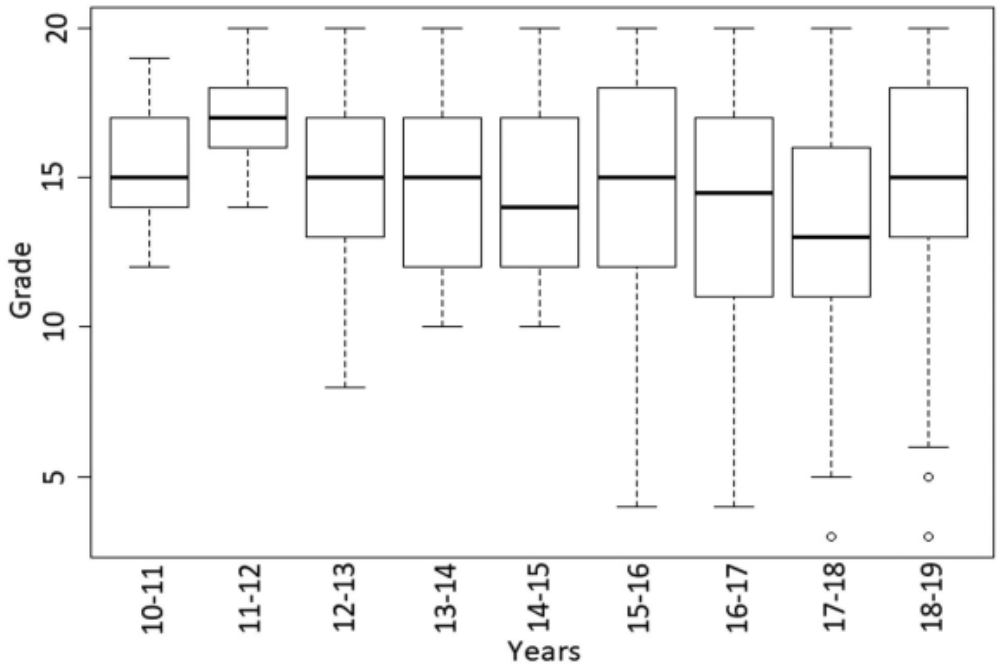


Fig. 6. Students' final grades in different years.

After the grade prediction, the results were compared with the actual grades of the students and the accuracy was calculated. The overall accuracy was defined as the total number of correct predictions divided by the total number of predictions, as presented in the following :

$$Overall\ Accuracy = \frac{\sum_{i=1}^{K} n_i}{n}.$$

Prediction Results

1) Baselines: In these methods, the NB, RF, and KNN algorithms were used on the datasets to predict the final grades of the students. Here, the data were not balanced and the feature selection technique was not used.

2) Confirmed Features: In these methods, initially the irrelevant and unimportant data features were dropped using a feature selection technique, and then the final grades

were predicted using the NB, RF, and KNN algorithms.

3) Reverse Methods: These methods took the same steps as our method while they first used the feature selection technique and then balanced the datasets but in our method we first balanced the data and then used the feature selection technique

All the mentioned prediction methods were generated using 75% of the data as training set and 25% as test, and their prediction ability was tested using a tenfold cross-validation technique.

Our justification for outperforming the reverse methods is that our method used more data to balance the datasets while the reverse methods balanced the data after dropping the attributes that were detected as the irrelevant ones. We estimated the average accuracy of the results of these three versions of Our Method. The average accuracy of the RF-based algorithm was 78% while it was descended to 72.7% and 74.7% using the NB-based and KNN-based algorithms, respectively. Therefore, our method based on the RF had a better performance than the ones using the NB and KNN algorithms.

Future Work

In the future, we can plan to extend our grade prediction approach for personalizing the course materials for the students.

CONCLUSION

The article describes a method for predicting the final grades of students in a gamification course with high accuracy at an early stage of the course. The method involves clustering students based on their accumulated XP and balancing the clusters by adding virtual students. Three algorithms were used to predict the final grades. The results showed that the final grades could be predicted at the end of the fourth week of a semester for all years. The approach can be extended in the future by personalizing course materials and considering a dynamic number of student clusters.