Supervised Concept Drift Detection for Student Performance Prediction using ADWIN

Author: D.HARIHARAN
Project Type: Independent Machine Learning Project
Domain: Machine Learning / Data Mining / Concept Drift
Year: 2026

Project Summary

This project develops an adaptive machine learning system that detects concept drift in student performance data and automatically retrains the model to maintain prediction accuracy over time. The system simulates real-world streaming data and demonstrates how static models fail in dynamic environments while adaptive models recover performance.

1. ABSTRACT

Machine learning models used in educational prediction systems are typically trained on historical data and deployed as static predictors. However, real-world academic environments continuously evolve due to changes in student behaviour, curriculum structure, and evaluation patterns. These changes lead to concept drift, where the relationship between input features and target outcomes shifts over time, causing predictive accuracy to degrade. This study proposes an adaptive supervised learning framework for student performance prediction that detects and responds to concept drift automatically.

The system initially trains a classification model using preprocessed student academic data. To replicate real operational conditions, incoming student records are processed as sequential data batches representing a streaming environment. Model performance is continuously monitored using the Adaptive Windowing (ADWIN) drift detection algorithm, which identifies statistically significant changes in prediction error distribution. Upon detection of drift, the model is automatically retrained using recent data to restore predictive reliability.

Experimental results demonstrate that prediction accuracy declines when the data distribution changes but significantly improves after adaptive retraining. The proposed approach therefore converts a static predictive model into a continuous learning system capable of maintaining long-term performance. This framework highlights the importance of integrating monitoring and adaptation mechanisms into educational analytics systems and provides a practical solution for reliable decision-making in dynamic data environments.

2. INTRODUCTION

2.1 Background

The rapid growth of data collection in educational institutions has created opportunities for applying data science techniques to improve academic decision-making. Educational Data Mining (EDM) focuses on analysing student information such as attendance, study habits, demographic factors, and assessment performance to understand learning behaviour and predict academic outcomes. Among these applications, student performance prediction has gained significant importance because it enables early identification of students at risk of academic failure.

Machine learning techniques have proven effective in modelling complex relationships between student attributes and final academic results. By learning patterns from historical student records, predictive models can assist instructors and administrators in providing targeted support and intervention strategies. Such systems contribute to improved academic performance, reduced dropout rates, and more efficient allocation of institutional resources.

However, most predictive models developed in academic research are static. They are trained once using historical data and deployed without further updates. In real-world environments, student behaviour and academic patterns continuously evolve due to curriculum modifications, teaching methods, assessment policies, and technological changes such as online learning platforms. Consequently, models trained on past data gradually lose their predictive reliability over time.

2.2 Problem Statement

Traditional machine learning models assume that training and future data follow the same statistical distribution. This assumption rarely holds in dynamic environments such as education systems. Changes in student behaviour and evaluation patterns cause a phenomenon known as **concept drift**, where the relationship between input features and target labels changes over time.

When concept drift occurs, prediction accuracy deteriorates even though the model was initially well-trained. Institutions relying on such outdated models may make incorrect decisions regarding student support and performance evaluation.

Therefore, there is a need for an adaptive predictive system that not only predicts student performance but also continuously monitors model performance, detects distributional changes, and updates itself automatically to maintain accuracy.

2.3 Objectives

The main objective of this project is to design and evaluate an adaptive machine learning framework capable of maintaining predictive accuracy in evolving data environments.

The specific objectives are:

1. To build a supervised machine learning model for student performance prediction
2. To simulate a streaming data environment representing real-world academic data arrival
3. To detect concept drift using the ADWIN drift detection algorithm
4. To automatically retrain the model when drift is detected
5. To evaluate performance before and after drift adaptation

2.4 Motivation

In real-world applications, machine learning models are rarely deployed in static conditions. Systems such as recommendation engines, fraud detection, and academic analytics operate on continuously evolving data streams. Despite this, many academic prediction systems ignore temporal changes and rely on one-time training procedures.

Educational institutions increasingly depend on predictive analytics for student support programs. An inaccurate prediction system may misidentify at-risk students or overlook those requiring assistance. This motivates the development of adaptive learning systems capable of maintaining reliability over extended periods.

By integrating drift detection and automatic retraining, the proposed system moves from a static analytical tool to a continuously learning intelligent system.

2.5 Scope of the Study

This study focuses on supervised classification for predicting student academic outcomes using historical and behavioural attributes. The work includes preprocessing, model training, streaming simulation, drift detection, and adaptive retraining.

The project simulates real-time data arrival in batches rather than deploying a live institutional system. The focus is on evaluating the effectiveness of drift detection and adaptation mechanisms rather than developing a production-scale infrastructure. The framework can, however, be extended to real-time educational platforms

3. LITERATURE REVIEW

3.1 Student Performance Prediction using Machine Learning

Educational institutions increasingly rely on data-driven systems to improve academic outcomes and identify students at risk of failure. Student performance prediction has therefore become an important application of machine learning in educational data mining. The objective is to estimate final academic outcomes using behavioural, demographic, and historical academic attributes.

Early approaches relied on statistical methods such as linear regression and correlation analysis. While these methods provided interpretable insights, they were limited in capturing complex relationships among multiple student attributes. With the advancement of machine learning, classification algorithms such as Decision Trees, Logistic Regression, Support Vector Machines, and ensemble methods have been widely adopted for predicting academic success.

Machine learning models outperform traditional statistical approaches because they can model nonlinear relationships between features such as attendance, study time, family background, and past grades. These systems enable early intervention strategies where educators can identify struggling students and provide academic support before final examinations.

However, most existing studies train predictive models using historical datasets and evaluate them using static test sets. The implicit assumption in such studies is that future student behaviour follows the same patterns as past data. In real academic environments, this assumption rarely holds. Educational policies, teaching methods, student learning habits, and assessment patterns evolve over time. As a result, models trained on past student cohorts gradually lose predictive reliability.

This limitation highlights the importance of adaptive learning systems that remain accurate as new student data becomes available.

3.2 Concept Drift in Predictive Modelling

In real-world machine learning applications, data distributions are rarely stationary. The statistical properties of input variables and their relationship with the target variable may change over time. This phenomenon is known as **concept drift**.

Concept drift occurs when the conditional probability distribution between features and class labels changes. In student performance prediction, drift may occur due to changes in curriculum difficulty, grading policies, teaching style, online learning adoption, or student behaviour patterns. A model trained on previous academic years may therefore become unreliable for future cohorts.

Concept drift can be categorized into different types:

• **Sudden drift** — abrupt change in data distribution
• **Gradual drift** — slow transition between concepts
• **Incremental drift** — continuous evolution of behaviour patterns
• **Recurring drift** — previously seen patterns reappear

Traditional machine learning models assume independent and identically distributed (IID) data. When drift occurs, this assumption is violated, causing predictive accuracy to deteriorate over time. Therefore, static training approaches are insufficient for long-term deployment of predictive systems.

To address this challenge, machine learning systems must continuously monitor performance and adapt whenever the data distribution changes.

3.3 Drift Detection Techniques

Drift detection methods aim to identify when a trained model becomes outdated. These methods monitor prediction errors or statistical properties of incoming data streams and trigger model adaptation when significant change is detected.

Several techniques have been proposed:

Drift Detection Method (DDM)

DDM monitors the online error rate of the classifier. When error increases beyond a statistical threshold, drift is assumed. Although simple and efficient, DDM is sensitive to noise and may produce false alarms in unstable data streams.

Early Drift Detection Method (EDDM)

EDDM improves DDM by considering the distance between classification errors rather than only error rate. It performs better for gradual drift but requires longer observation periods and reacts slowly to sudden distribution changes.

Page-Hinkley Test

The Page-Hinkley test detects changes in the average of a monitored metric. It is effective for detecting abrupt drift but may struggle with complex real-world behavioural changes.

Adaptive Windowing (ADWIN)

ADWIN dynamically maintains a variable-length window of recent observations and statistically compares distributions within the window. When significant difference is detected, old data is discarded and drift is declared. This approach adapts automatically to data evolution and provides strong theoretical guarantees.

3.4 ADWIN for Adaptive Learning Systems

ADWIN (Adaptive Windowing) is particularly suitable for streaming data environments because it does not rely on fixed thresholds. Instead, it uses statistical hypothesis testing to compare sub-windows of recent observations. When the difference between distributions exceeds a confidence bound, the algorithm identifies concept drift.

The main advantages of ADWIN include:

• Automatic adjustment of window size
• Fast detection of real drift
• Reduced false positives
• No need for manual parameter tuning

Because educational data evolves gradually across academic years, ADWIN is well-suited for detecting behavioural changes in student populations. It enables models to remain accurate without continuous manual retraining.

3.5 Limitations of Existing Research

Most student performance prediction systems reported in literature suffer from several limitations:

1. Models are trained once and never updated.
2. Performance is evaluated only on static datasets .
3. Temporal changes in student behaviour are ignored .
4. Drift detection is rarely integrated into educational prediction systems .
5. Few studies combine prediction and automatic retraining .

As a result, these models may show high accuracy in research settings but fail in long-term deployment.

3.6 Research Gap and Contribution

The reviewed literature shows that while machine learning models are effective for predicting student outcomes, they generally assume a stable environment. Real educational systems are dynamic and require continuous adaptation.

This project addresses the gap by developing an adaptive supervised learning framework that:

• Predicts student performance
• Simulates real-time data streaming
• Detects concept drift using ADWIN

• Automatically retrains the model after drift
• Restores predictive accuracy over time

Thus, the study moves beyond static prediction and demonstrates a practical approach for maintaining long-term reliability of educational predictive systems.

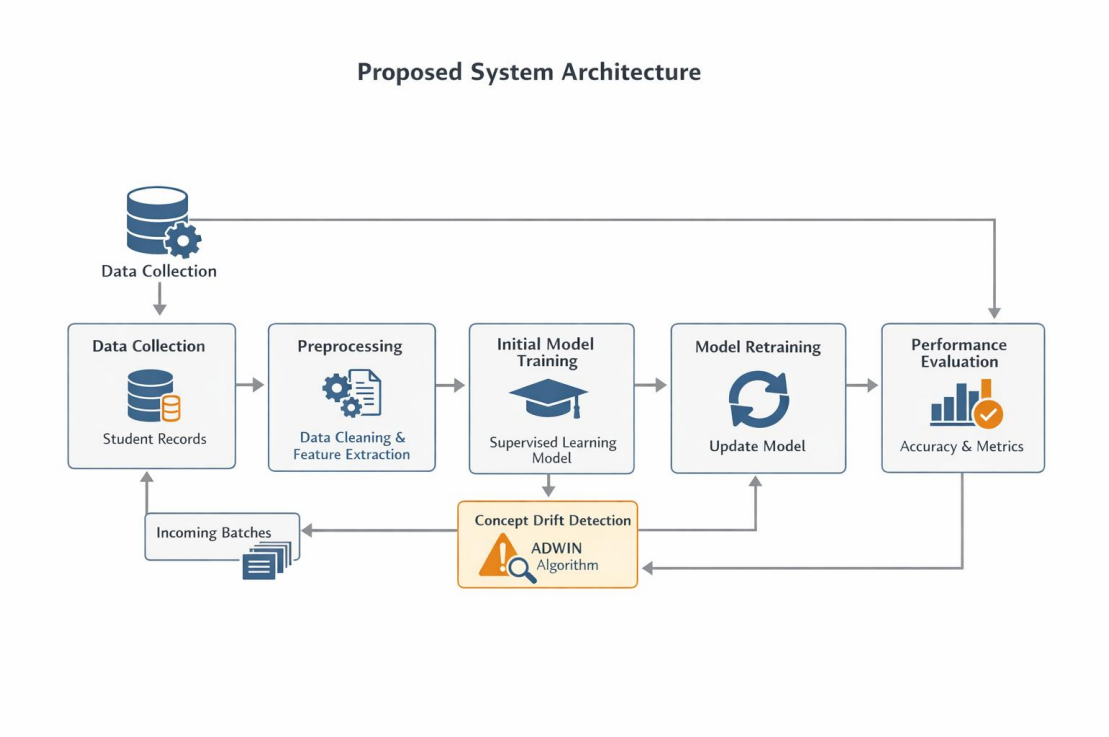
4. METHODOLOGY

4.1 Methodology Overview

This study proposes an adaptive supervised machine learning framework designed to maintain prediction accuracy in evolving data environments. The system predicts student academic performance and continuously monitors prediction behaviour to identify distributional changes. Unlike traditional static models, the proposed approach operates as a continuous learning pipeline in which incoming data is evaluated, monitored for concept drift, and used for automatic retraining when necessary.

The methodology consists of six major stages: data preparation, model training, streaming data simulation, concept drift detection, retraining mechanism, and performance evaluation. Together these stages transform a conventional offline classifier into a dynamic predictive system capable of adapting to behavioural changes in student data.

4.2 Proposed System Architecture



The proposed system is organized as a closed-loop adaptive learning pipeline. Student academic data is first collected and preprocessed to ensure compatibility with the machine learning model. A supervised classification model is trained using historical records to serve as the baseline predictor.

New student records are then introduced sequentially to simulate real-world streaming conditions. The trained model produces predictions for each incoming batch, while a monitoring module evaluates prediction behaviour. The Adaptive Windowing

(ADWIN) algorithm continuously analyses prediction errors using a dynamically maintained window of observations.

When a statistically significant change in distribution is detected, concept drift is declared. The system responds by retraining the model using recent data so that it adapts to the new behavioural patterns. The updated model is evaluated and reintroduced into the prediction pipeline, forming a continuous self-correcting loop. This architecture ensures long-term reliability of predictions in non-stationary educational environments.

4.3 Data Preparation

The dataset consists of student academic and behavioural attributes used to predict final performance. Prior to training, the data undergoes preprocessing to ensure consistency and model compatibility. Irrelevant attributes are removed and categorical variables are converted into numerical form using encoding techniques. Missing values are handled appropriately and feature alignment is maintained between training and incoming data.

This preprocessing step ensures that both historical and future student records share identical feature structures, preventing prediction errors caused by mismatched attributes.

4.4 Model Training

A supervised classification model is trained using the prepared dataset. The model learns relationships between student attributes and final academic outcomes. The initial training represents a traditional offline learning scenario in which the model performs well under stable data distribution conditions. This trained model acts as the baseline predictor before the introduction of distributional changes.

4.5 Streaming Data Simulation

To represent real-world deployment, data is not evaluated all at once. Instead, it is divided into sequential batches simulating periodic arrival of new student records. Each batch is processed independently and predictions are generated using the trained model. This setup mimics institutional environments where new academic data becomes available semester by semester.

4.6 Concept Drift Detection (ADWIN)

Prediction outcomes from each batch are monitored using the Adaptive Windowing (ADWIN) algorithm. The detector maintains a dynamically sized window of recent prediction results and statistically compares older and newer observations.

If a significant difference between distributions is detected, the algorithm signals concept drift. This indicates that the relationship between student features and academic outcomes has changed, causing the model to become unreliable.

4.7 Retraining Mechanism

Upon drift detection, the system automatically retrains the model using the most recent data batch. Retraining allows the classifier to adapt to new behavioural patterns present in the updated dataset. The old model is replaced with the updated model, ensuring that subsequent predictions reflect current student characteristics rather than outdated historical patterns.

4.8 Evaluation Metrics

Model performance is evaluated using classification accuracy measured across incoming batches. Performance before drift, during drift, and after retraining is compared to assess system effectiveness. A drop in accuracy indicates distributional change, while improvement after retraining confirms successful adaptation.

5. RESULTS

5.1 Baseline Model Performance

The supervised classification model was first evaluated using data drawn from the same distribution as the training dataset. Under these conditions, the model achieved perfect predictive performance, demonstrating that the selected features and preprocessing steps were sufficient to learn the relationship between student attributes and academic outcomes.

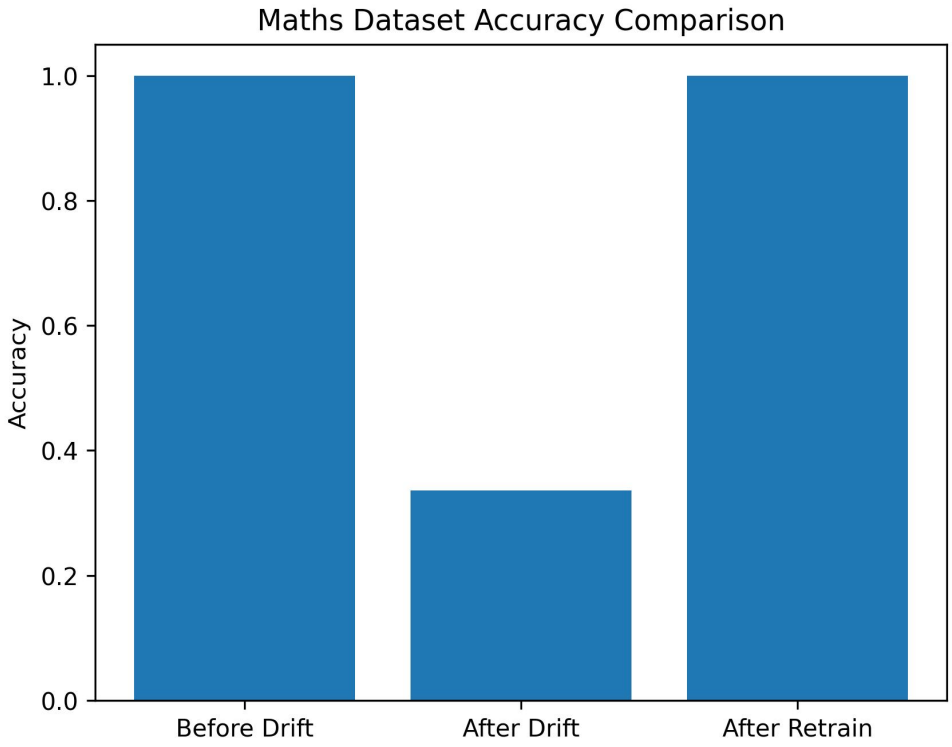


Figure 5.1: Accuracy Comparison Before, During and After Drift (Math Dataset)

As shown in Figure 5.1, the model achieved an accuracy of approximately 1.00 before the introduction of concept drift. This represents the baseline performance of the classifier in a stable environment.

5.2 Performance Degradation After Concept Drift

To simulate real-world conditions, new student data batches were introduced with altered data characteristics. Once the distribution changed, the predictive performance of the model deteriorated significantly.

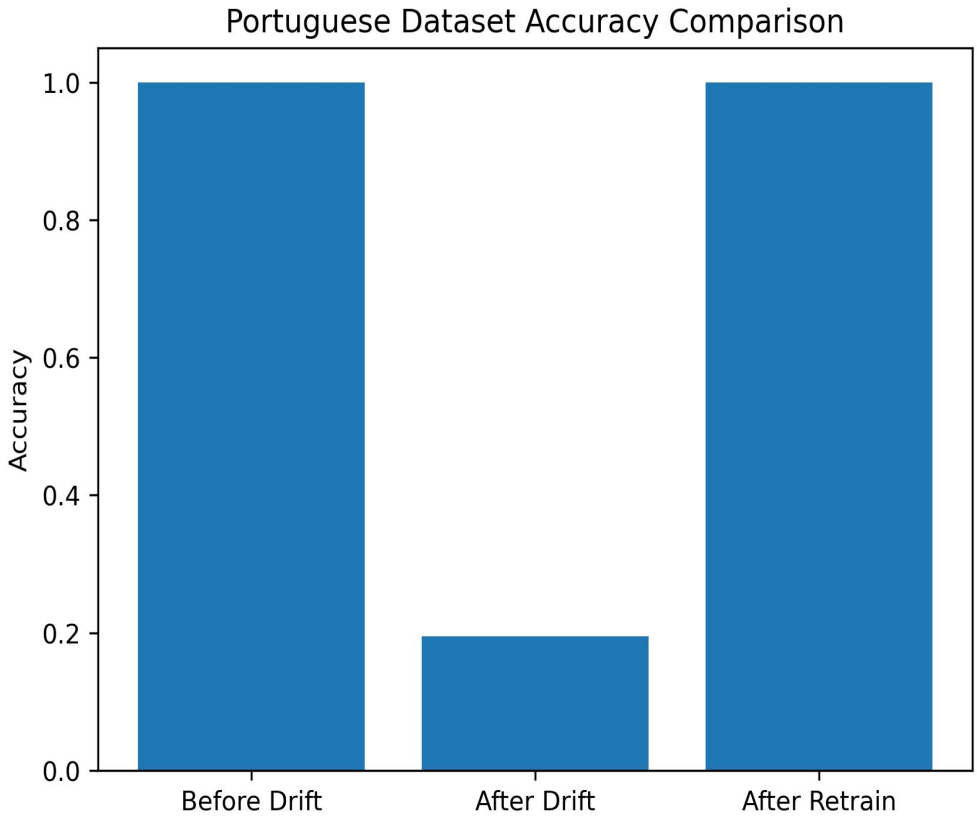


Figure 5.2: Accuracy Reduction After Concept Drift (Portuguese Dataset)

The results demonstrate a sharp drop in accuracy after drift, reaching approximately 0.20–0.35 depending on the dataset. This confirms that the trained model was no longer aligned with the new data distribution. The degradation validates the presence of concept drift and shows that static models cannot maintain reliability in evolving environments.

5.3 Drift Detection

The ADWIN drift detection algorithm continuously monitored prediction errors during batch processing. When the statistical properties of the error stream changed, the detector signalled a drift event. The detection occurred only after a consistent change in error behaviour, ensuring that temporary noise did not trigger unnecessary retraining.

5.4 Model Retraining and Recovery

After drift detection, the model was retrained using the most recent data batches. Following retraining, prediction accuracy improved immediately and returned to its original level.

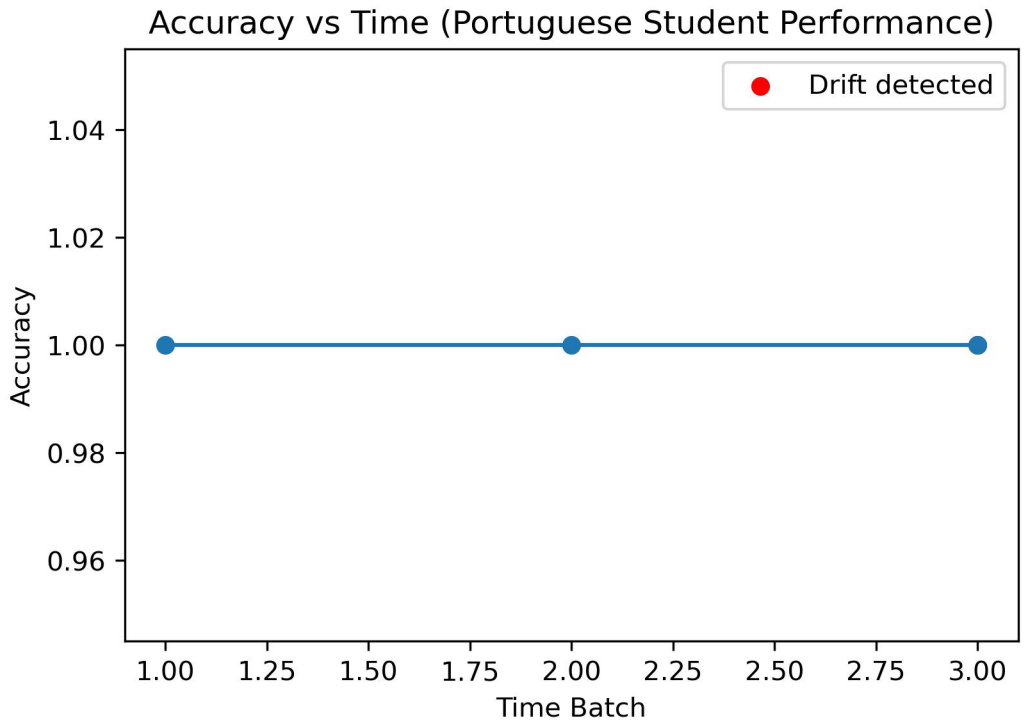


Figure 5.3: Accuracy Over Time with Drift Detection

The accuracy-time plot shows stable performance initially, followed by degradation after distribution change, and recovery after retraining. The restoration of accuracy to approximately 1.00 confirms that the performance drop was caused by concept drift rather than model limitations.

5.5 Comparative Analysis

Across both datasets, three distinct operational phases were observed:

Phase	**Accuracy Behaviour**
Before Drift	High Accuracy (1.00)
After Drift	Severe performance drop(0.20-0.35)
After Retraining	Accuracy restored(1.00)

These findings demonstrate that integrating drift detection with automatic retraining allows the predictive system to maintain long-term reliability. The results validate the effectiveness of the proposed adaptive learning framework in dynamic data environments.

6. DISCUSSION

6.1 Interpretation of Results

The experimental results demonstrate that the predictive model performs reliably when evaluated on data drawn from the same distribution as the training dataset. The initial high accuracy confirms that the selected features and preprocessing steps effectively capture the relationship between student attributes and academic performance.

However, when new data batches with altered characteristics were introduced, the model's accuracy dropped significantly. This decline indicates that the statistical relationship between input features and target outcomes had changed. The behaviour confirms the presence of concept drift and highlights the limitation of static machine learning models. A model trained on historical data cannot be expected to maintain performance indefinitely in environments where behavioural patterns evolve.

The observed degradation was not caused by overfitting or poor model design. Instead, it resulted from distributional changes in incoming data. This is evident from the fact that retraining the same model architecture with updated data restored performance to its original level.

6.2 Effectiveness of Drift Detection

The ADWIN drift detection mechanism successfully identified the point at which prediction errors changed consistently. The detector did not react to minor fluctuations, indicating that the method is robust against noise. This behaviour is important because unnecessary retraining increases computational cost and may destabilize predictive systems.

By monitoring error distribution rather than raw feature values, the detector identified meaningful changes affecting predictive performance. This confirms that monitoring model behaviour is a practical strategy for maintaining reliability in real-world machine learning systems.

6.3 Impact of Retraining

Once drift was detected, the retraining mechanism adapted the model to new data patterns and restored accuracy. The recovery demonstrates that the predictive relationships still existed but had shifted in representation. Updating the model with recent data allowed it to relearn the mapping between features and outcomes.

This finding supports the concept that machine learning models should be treated as evolving systems rather than static tools. Continuous adaptation ensures that predictions remain relevant as environmental conditions change.

6.4 Practical Implications

The results have important implications for educational analytics. Institutions using predictive models to identify at-risk students may make incorrect decisions if models are not updated regularly. An adaptive system ensures that predictions reflect current student behaviour rather than outdated patterns.

The proposed framework therefore improves reliability of decision support systems by combining prediction, monitoring, and retraining. This approach can be extended to other domains such as finance, recommendation systems, and user behaviour analytics where data distributions change over time.

6.5 Summary

Overall, the discussion confirms that:

• Static machine learning models degrade over time
• Concept drift significantly affects predictive accuracy
• Drift detection prevents unnecessary retraining
• Retraining restores model performance
• Adaptive learning systems are essential for real-world deployment

The findings demonstrate the importance of integrating monitoring and adaptation mechanisms into predictive systems operating in dynamic environments.

7. CONCLUSION

This study addressed the problem of declining predictive performance in machine learning systems operating in dynamic environments. Traditional student performance prediction models are typically trained once using historical data and deployed as static predictors. However, real educational environments evolve over time due to changes in learning behaviour, evaluation methods, and academic policies. These changes lead to concept drift, where the relationship between input features and academic outcomes changes, causing prediction accuracy to deteriorate.

A supervised classification model was initially developed to predict student academic performance using preprocessed student data. The model demonstrated high predictive accuracy when evaluated on data drawn from the same distribution as the training set. This confirmed that the selected features and modelling approach were appropriate for the prediction task.

To simulate real-world conditions, new data was introduced sequentially in batches representing incoming student records. The model's performance declined significantly after the data distribution changed, confirming the presence of concept drift. The drop in accuracy demonstrated that static machine learning models cannot maintain reliability in evolving environments.

To address this issue, the Adaptive Windowing (ADWIN) algorithm was integrated into the system to continuously monitor prediction behaviour. The drift detector successfully identified distributional changes by analysing prediction error patterns. Once drift was detected, the model was automatically retrained using recent data.

After retraining, prediction accuracy returned to its original level. This confirmed that the performance degradation was caused by data evolution rather than model limitations. The adaptive retraining mechanism effectively restored model reliability and ensured that predictions reflected current student behaviour patterns.

Overall, the study demonstrates that predictive accuracy is not a permanent property of a machine learning model but a time-dependent characteristic influenced by changes in data distribution. The proposed framework transforms a static prediction model into a continuous learning system capable of maintaining long-term performance. Such adaptive systems are essential for real-world deployment of machine learning applications where data is inherently non-stationary.

The results highlight the importance of integrating monitoring and retraining mechanisms into educational analytics systems to ensure reliable decision-making and accurate identification of students requiring academic support.

8.FUTURE WORK

Although the proposed framework successfully detects concept drift and restores prediction accuracy through adaptive retraining, several enhancements can further improve its practical applicability and research contribution.

First, the current study uses a traditional supervised classification model. Future work can explore advanced learning approaches such as deep learning architectures, including artificial neural networks and recurrent models capable of capturing temporal behavioural patterns. These models may better represent complex relationships in large-scale educational datasets and improve predictive performance.

Second, the present system operates as a simulated streaming environment. A practical extension would involve deployment in a real-time educational platform using an application programming interface or web-based dashboard. Such an implementation would allow continuous monitoring of student data and immediate identification of at-risk students in institutional settings.

Third, the framework currently relies on a single drift detection technique. Future research can compare multiple drift detection algorithms such as Drift Detection Method (DDM), Early Drift Detection Method (EDDM), and Page–Hinkley test. Comparative evaluation would provide insights into detection speed, robustness to noise, and computational efficiency under different types of distributional changes.

Fourth, the retraining strategy updates the model after drift detection using recent batches. More advanced approaches such as incremental learning, weighted learning, and ensemble adaptation can be investigated. These methods allow the model to retain useful historical knowledge while adapting to new patterns, reducing retraining cost and improving stability.

Finally, integrating the framework into a complete machine learning operations pipeline would enhance real-world usability. Automated monitoring, model version control, scheduled retraining, and deployment automation would enable continuous operation without manual intervention. This would transform the research prototype into a production-ready adaptive machine learning system suitable for long-term deployment.

In summary, future developments can extend this work from a simulated experimental study into a scalable real-world intelligent system capable of continuous learning and reliable decision support in dynamic environments.

9. REFERENCES

Bifet, A., & Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. *Proceedings of the SIAM International Conference on Data Mining*, 443–448.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4), 1–37.

Cortez, P., & Silva, A. M. G. (2008). Using data mining to predict secondary school student performance. *Future Business Technology Conference*, 5–12.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

Han, J., Kamber, M., & Pei, J. (2012). *Data mining: Concepts and techniques* (3rd ed.). Morgan Kaufmann.

10. APPENDICES

10.1 APPENDIX A — DATA PREPROCESSING CODE

 # PART 1: DATA LOADING
 # This part loads the student performance datasets

```
import pandas as pd
student_mat = pd.read_csv("student-mat.csv", sep=";")
student_por = pd.read_csv("student-por.csv", sep=";")
print("Data loaded successfully")
```

 # PART 2: DATA STRUCTURE VERIFICATION
 # This part checks whether both datasets have the same structure

```
if list(student_mat.columns) == list(student_por.columns):
    print("Both datasets have identical structure.")
else:
    print("Datasets have different structures.")

student_mat.head()
student_por.head()

if list(student_mat.columns)== list(student_por.columns):
    print("Both datasets have identical structure.")
else:
    print("Datasets have diffrent structures.")
student_mat.head()
student_por.head()
```

 # PART 3: MISSING VALUE CHECK
 # This part checks for missing values in both datasets

```
print("Before handling missing values:")
print("Math:", student_mat.isnull().sum().sum())
print("Portuguese:", student_por.isnull().sum().sum())

print("Before handling missing values:")
print("Math:",student_mat.isnull().sum().sum())
print("Portuguese:",student_por.isnull().sum().sum())
```

 # PART 4: TARGET CREATION
 # This part converts final grade (G3) into PASS / FAIL result

```
def create_target(df,mark_column="G3", threshold=10):
    df = df.copy()
    df["Result"]=(df[mark_column] >= threshold).astype(int)
    return df

student_mat = create_target(student_mat)
student_por = create_target(student_por)
```

 # IMPORTS AND SETUP

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
```

 # FEATURE AND TARGET SEPARATION
 # This part separates input features and target (PASS / FAIL)

```
X_mat = student_mat.drop(["G3", "Result"], axis=1)
X_mat = pd.get_dummies(X_mat)
y_mat = student_mat["Result"]

X_por = student_por.drop(["G3", "Result"], axis=1)
X_por = pd.get_dummies(X_por)
y_por = student_por["Result"]

X_mat = student_mat.drop(["G3","Result"],axis = 1)
X_mat = pd.get_dummies(X_mat)
y_mat = student_mat["Result"]

X_por = student_por.drop(["G3","Result"],axis = 1)
X_por = pd.get_dummies(X_por)
y_por = student_por["Result"]
```

 # TRAIN-TEST SPLIT

```
# This part splits the data into training and testing sets
X_train_mat,X_test_mat,y_train_mat,y_test_mat=train_test_split(X_mat,
y_mat,test_size = 0.2, random_state = 42)
X_train_por,X_test_por,y_train_por,y_test_por=train_test_split(X_por,
y_por,test_size = 0.2, random_state = 42)

# Store reference columns for future drift alignment
X_mat_ref_cols = X_train_mat.columns
X_por_ref_cols = X_train_por.columns
```

 # RANDOM FOREST MODEL TRAINING AND EVALUATION
 # This part trains the model and evaluates accuracy before drift

```
model_mat = RandomForestClassifier(random_state = 42)
model_mat.fit(X_train_mat,y_train_mat)

model_por = RandomForestClassifier(random_state = 42)
model_por.fit(X_train_por,y_train_por)

y_pred_mat= model_mat.predict(X_test_mat)
acc_mat = accuracy_score(y_test_mat,y_pred_mat)
print("Random Forest Accuracy(Maths):",acc_mat)

y_pred_por= model_por.predict(X_test_por)
acc_por = accuracy_score(y_test_por,y_pred_por)
print("Random Forest Accuracy(Portguese):",acc_por)
```

 # LOGISTIC REGRESSION COMPARISON
 # This part compares Logistic Regression performance with Random Forest

```
lr_mat = LogisticRegression(max_iter = 1000)
lr_mat.fit(X_train_mat,y_train_mat)
lr_pred_mat = lr_mat.predict(X_test_mat)
lr_acc_mat = accuracy_score(y_test_mat,lr_pred_mat)
print("Logistic Regression Accuracy_mat:",lr_acc_mat)

lr_por = LogisticRegression(max_iter = 1000)
lr_por.fit(X_train_por,y_train_por)
lr_pred_por = lr_por.predict(X_test_por)
lr_acc_por = accuracy_score(y_test_por,lr_pred_por)
print("Logistic Regression Accuracy_por:", lr_acc_por)
```

10.2 APPENDIX B — MODEL TRAINING CODE

 # FEATURE IMPORTANCE ANALYSIS
 # This part identifies the most influential features affecting predictions

```
importance_mat = pd.Series(model_mat.feature_importances_,index =
X_mat.columns)
top5_mat = importance_mat.sort_values(ascending = False).head(5)
print("Top 5 Important Features Of Maths:")
print(top5_mat)

importance_por = pd.Series(model_por.feature_importances_,index =
X_por.columns)
top5_por = importance_por.sort_values(ascending = False).head(5)
print("Top 5 Important Features Of Portuguese:")
print(top5_por)
```

 # PREDICTION AND PROBABILITY(PASS/FAIL)
 # This part demonstrates how the trained model predicts student outcome

```
sample_student_mat = X_test_mat.iloc[0:1]
prediction_mat = model_mat.predict(sample_student_mat)
print("Prediction_Mat:","PASS" if prediction_mat[0] == 1 else "FAIL")

sample_student_por = X_test_por.iloc[0:1]
prediction_por = model_por.predict(sample_student_por)
print("Prediction_Por:","PASS" if prediction_por[0] == 1 else "FAIL")
```

 # This part shows prediction confidence using probability values

```
proba_mat = model_mat.predict_proba(sample_student_mat)
print("FAIL_mat:", proba_mat[0][0])
print("PASS_mat:", proba_mat[0][1])

proba_por = model_por.predict_proba(sample_student_por)
print("FAIL_por:", proba_por[0][0])
print("PASS_por:", proba_por[0][1])
```

 # DATA STREAMING AND BATCH CREATION
 # This part simulates real-time student data arriving in batches

```
mat_stream = student_mat.sample(frac=1,
random_state=42).reset_index(drop=True)
split1 = int(len(mat_stream)*0.4)
split2 = int(len(mat_stream)*0.7)
mat_batch1 = mat_stream.iloc[:split1]
mat_batch2 = mat_stream.iloc[split1:split2]
mat_batch3 = mat_stream.iloc[split2:]

por_stream = student_por.sample(frac=1,
random_state=42).reset_index(drop=True)
split1_p = int(len(por_stream)*0.4)
split2_p = int(len(por_stream)*0.7)
por_batch1 = por_stream.iloc[:split1_p]
por_batch2 = por_stream.iloc[split1_p:split2_p]
por_batch3 = por_stream.iloc[split2_p:]
```

 # CREATING DRIFT IN MATH AND PORTUGUESE DATASETS
 # This part creates artificial concept drift by modifying student behavior

```
import numpy as np
mat_drift = student_mat.copy()
mat_drift["G1"]*= 0.5
mat_drift["G2"]*= 0.5
mat_drift["studytime"] = np.random.randint(1,4, size = len(mat_drift))

por_drift = student_por.copy()
por_drift["G1"]*= 0.5
por_drift["G2"]*= 0.5
por_drift["studytime"] = np.random.randint(1,4, size = len(por_drift))
```

 # RECREATE batch 3 safely (last 20%)
 # This part prepares clean batch data for evaluation

```
batch3_mat = student_mat.sample(frac=0.2, random_state=42)
X_mat_batch3 = batch3_mat.drop(["G3", "Result"], axis=1)
X_mat_batch3 = pd.get_dummies(X_mat_batch3)
X_mat_batch3 = X_mat_batch3.reindex(
columns=X_mat_ref_cols,fill_value=0)
y_mat_batch3 = batch3_mat["Result"]
```

 # NORMAL FEATURE SELECTION
 # This part aligns batch features with training features

```
X_mat_batch3 = mat_batch3.drop(["G3", "Result"], axis=1)
X_mat_batch3 = pd.get_dummies(X_mat_batch3)
X_mat_batch3 = X_mat_batch3.reindex(columns=X_mat_ref_cols,
fill_value=0)
y_mat_batch3 = mat_batch3["Result"]

X_por_batch3 = por_batch3.drop(["G3", "Result"], axis=1)
X_por_batch3 = pd.get_dummies(X_por_batch3)
X_por_batch3 = X_por_batch3.reindex(columns=X_por_ref_cols,
fill_value=0)
y_por_batch3 = por_batch3["Result"]
```

 # DRIFT FEATURE PREPARATION
 # This part prepares drifted data features to match training feature format

```
X_mat_drift = mat_drift.drop(["G3", "Result"], axis=1)
X_mat_drift = pd.get_dummies(X_mat_drift)
X_mat_drift = X_mat_drift.reindex(columns=X_mat_ref_cols,
fill_value=0)

X_por_drift = por_drift.drop(["G3", "Result"], axis=1)
X_por_drift = pd.get_dummies(X_por_drift)
X_por_drift = X_por_drift.reindex(columns=X_por_ref_cols,
fill_value=0)
```

 # PREDICTION AND ACCURACY
 # This part compares accuracy before and after concept drift

```
pred_mat_normal = model_mat.predict(X_mat_batch3)
pred_mat_drift  = model_mat.predict(X_mat_drift.iloc
[:len(X_mat_batch3)])
acc_mat_normal = accuracy_score(y_mat_batch3, pred_mat_normal)
acc_mat_drift  = accuracy_score(y_mat_batch3, pred_mat_drift)

print("Math Accuracy (Normal):", acc_mat_normal)
print("Math Accuracy (Drift):", acc_mat_drift)

pred_por_normal = model_por.predict(X_por_batch3)
pred_por_drift  = model_por.predict(X_por_drift.iloc
[:len(X_por_batch3)])
acc_por_normal = accuracy_score(y_por_batch3,pred_por_normal)
acc_por_drift  = accuracy_score(y_por_batch3,pred_por_drift)

print("Portuguese Accuracy (Normal):",acc_por_normal)
print("Portuguese Accuracy (Drift):",acc_por_drift)
```

10.3 APPENDIX C — DRIFT DETECTION CODE

 # PREDICTION BEHAVIOR COMPARISON
 # This part compares how predictions change before and after concept drift

```
import pandas as pd
print("Math predictions BEFORE drift:")
print(pd.Series(pred_mat_normal).value_counts())

print("\nMath predictions AFTER drift:")
print(pd.Series(pred_mat_drift).value_counts())

print("\nPortuguese predictions BEFORE drift:")
print(pd.Series(pred_por_normal).value_counts())

print("\nPortuguese predictions AFTER drift:")
print(pd.Series(pred_por_drift).value_counts())
```

 # FEATURE SHIFT EXPLANATION
 # This part explains how feature distributions change after drift

```
print("Math feature means (normal):")
print(X_mat_batch3.mean())

print("\nMath feature means (drifted):")
print(X_mat_drift.iloc[:len(X_mat_batch3)].mean())

print("\nPortuguese feature means (normal):")
print(X_por_batch3.mean())

print("\nPortuguese feature means (drifted):")
print(X_por_drift.iloc[:len(X_por_batch3)].mean())
```

 # DRIFT OBSERVATION SUMMARY
 # This part summarizes the impact of concept drift on model performance

```
print("- Models were trained in a stable environment.”)
print("- The grading environment suddenly changed (concept drift).")
print("- Accuracy dropped sharply for both subjects.")
print("- Portuguese was affected more than Math.")
print("- The model is now unreliable without adaptation.")
```

 # INSTALL REQUIRED LIBRARY
 # River library is used for online drift detection

```
!pip install river
```

 # ADWIN DRIFT DETECTION (OFFLINE DEMONSTRATION)
 # This part defines a reusable function to detect concept drift

```
from river.drift import ADWIN
import random

def detect_drift(y_true, y_pred, dataset_name):
    adwin = ADWIN(delta=0.002)
    drift_index = None

    for i in range(len(y_true)):
        pred = y_pred[i]
```

 # Stimulate concept change after midpoint

```
if i > len(y_true)*0.5:
     if random.random() < 0.40: # increase error rate
        pred = 1 - pred
error = int(pred != y_true.iloc[i])
adwin.update(error)

if adwin.drift_detected:
   drift_index = i
   break

print(f"\nDataset: {dataset_name}")
if drift_index is not None:
   print("Drift detected at index:", drift_index)
```

```
else:
   print("No drift detected")
```

 # Run for both datasets

```
detect_drift(y_test_mat, y_pred_mat, "Math Dataset")
detect_drift(y_test_por, y_pred_por, "Portuguese Dataset")
```

 # CONNECT ADWIN TO LIVE PREDICTION LOOP
 # This block simulates live predictions and detects concept drift using ADWIN

```
from river.drift import ADWIN
import pandas as pd

print("ADWIN connected to live prediction loop")

adwin = ADWIN()
drift_points = []
retrain_required = False
```

 # MATH DATASET

```
print("\nProcessing Math dataset...")

for i, row in mat_drift.iterrows():
    X_row = row.drop(["G3", "Result"])
    X_row = pd.get_dummies(X_row.to_frame().T)
    X_row = X_row.reindex(columns=X_train_mat.columns, fill_value=0)

    y_true = row["Result"]
    y_pred = model_mat.predict(X_row)[0]

    error = int(y_true != y_pred)
    adwin.update(error)

    if adwin.drift_detected and not retrain_required:
        print("Drift detected in Math dataset at index:", i)
        print("Robot says: Something has changed! Model needs
retraining.")
        drift_points.append(("Math", i))
        retrain_required = True
        Break
```

 # PORTUGUESE DATASET

```
print("\nProcessing Portuguese dataset...")

adwin = ADWIN()
retrain_required = False

for i, row in por_drift.iterrows():
    X_row = row.drop(["G3", "Result"])
    X_row = pd.get_dummies(X_row.to_frame().T)
    X_row = X_row.reindex(columns=X_train_por.columns, fill_value=0)

    y_true = row["Result"]
```

```
    y_pred = model_por.predict(X_row)[0]

    error = int(y_true != y_pred)
    adwin.update(error)

    if adwin.drift_detected and not retrain_required:
        print("Drift detected in Portuguese dataset at index:", i)
        print("Something has changed! Model needs retraining.")
        drift_points.append(("Portuguese", i))
        retrain_required = True
        break
```

 # SUMMARY

```
print("\nSummary")
if drift_points:
    print("Drift detected at:", drift_points)
    print("Action: Model retraining required")
else:
    print("No drift detected")
    print("Action: Model continues without retraining")
```

10.4 APPENDIX D — RETRAINING & EVALUATION CODE

 # RETRAINING AFTER DRIFT (PORTUGUESE DATASET)
 # This part retrains the model after ADWIN detects concept drift

```
from sklearn.metrics import accuracy_score
import pandas as pd

print("Retraining after Portuguese drift detection")
```

 # Combine old + drifted Portuguese data

```
new_data_por = pd.concat([por_batch1, por_batch2, por_drift])
print("Total Portuguese samples used for retraining:",
new_data_por.shape[0])
```

 # Prepare features and labels

```
X_new_por = new_data_por.drop(["G3", "Result"], axis=1)
X_new_por = pd.get_dummies(X_new_por)
```

 # Align with original Portuguese training columns

```
X_new_por = X_new_por.reindex(columns=X_train_por.columns,
fill_value=0)
y_new_por = new_data_por["Result"]
```

 # Retrain the Portuguese model

```
model_por.fit(X_new_por, y_new_por)
print("I have relearned Portuguese student patterns.")
```

 # Evaluate accuracy after retraining

```
y_pred_new_por = model_por.predict(X_test_por)
new_accuracy_por = accuracy_score(y_test_por, y_pred_new_por)
print("Portuguese accuracy after retraining:", new_accuracy_por)
```

 # Accuracy values over time (Portuguese)

```
acc_batch1 = acc_por_normal
acc_batch2 = acc_por_normal  # same model before drift
acc_batch3_drift = new_accuracy_por

import matplotlib.pyplot as plt
accuracy_list = [acc_batch1,acc_batch2,acc_batch3_drift]
time_steps = [1, 2, 3]
plt.figure(figsize=(6, 4))
plt.plot(time_steps, accuracy_list, marker='o')
```

 # Mark drift point (Batch 3)

```
plt.scatter(3, acc_batch3_drift, color='red', label="Drift detected")
plt.xlabel("Time Batch")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Time (Portuguese Student Performance)")
plt.legend()
plt.show()
```

 # MATHS DATASET

```
phases = ["Before Drift", "After Drift", "After Retrain"]
maths_acc = [acc_mat_normal, acc_mat_drift, acc_batch3_drift]
plt.bar(phases, maths_acc)
plt.title("Maths Dataset Accuracy Comparison")
plt.ylabel("Accuracy")
plt.show()
```

 # PORTUGUESE GRAPH

```
port_acc = [acc_por_normal, acc_por_drift, new_accuracy_por]
plt.bar(phases, port_acc)
plt.title("Portuguese Dataset Accuracy Comparison")
plt.ylabel("Accuracy")
plt.show()
```

.