Project-Connect Four Move Prediction

Ghaliah Alsubaie, Norah Alsuwailem, Shatha Alabbad, and Maram Alsheddi

***Abstract*—**

This project focuses on predicting the optimal move in the Connect Four game using supervised machine learning models. Three baseline models—Decision Tree, Logistic Regression, and Random Forest—were first implemented to establish reference performance. Later, advanced ensemble and deep learning models (Gradient Boosting, XGBoost, and Neural Network) were developed, both with and without feature engineering, to improve prediction accuracy. The results show a steady improvement across models, where adding feature engineering increased accuracy and reduced generalization gaps.

I. INTRODUCTION

This project focuses on building an Artificial Intelligence (AI) model capable of predicting the next optimal move in the Connect Four game. The Connect Four task was provided as a course project requirement to explore and compare different Machine Learning (ML) techniques for game-playing. Game environments are particularly valuable for AI experimentation because they provide structured, rule-based systems that clearly demonstrate how algorithms can learn to make strategic decisions.

The purpose of this project is to gain practical insight into machine learning algorithms that are well-suited for move prediction and strategic decision-making. Connect Four is a two-player strategy game on a grid with 7 columns and 6 rows, hence a total of 42 spaces. Each player is associated with a color and takes turns dropping a disc of his or her color in one column. The disc then falls down to the lowest empty cell of the column. The first player to make an alignment of four consecutive discs wins, whether vertically, horizontally or diagonally. If the board is completely filled without forming any alignment, then the game ends in a draw.

In the earlier phases of this project, baseline models such as Decision Tree (DT), Logistic Regression (LR), and Random Forest (RF) were implemented to establish foundational performance benchmarks. Among them, the Random Forest model achieved the highest accuracy. Moving forward, the project expands toward advanced models and feature engineering, exploring models such as Gradient Boosting, XGBoost, and Neural Networks to further improve predictive accuracy and generalization performance.

.

II. METHODOLOGY

In this study, we followed a structured and iterative process aimed at optimizing the prediction of the next move in the Connect Four game.

A. Dataset

The dataset represents the Connect Four game states. It contained 42 input features representing the content of the 6x7 board cells and one feature indicating the player's turn. The target variable was the column index (0–6) where the next move should be placed. The data is provided in three separate files: a training set, a validation set, and a test set.

B. Baseline models

The work began with data preprocessing and baseline model training to establish reference accuracy levels. Three classical Machine Learning models—Decision Tree, Logistic Regression, and Random Forest—were implemented using Scikit-learn.[1] Each model was trained on the same data split for consistency.

C. Advanced models

Advanced ensemble models were developed to improve upon baseline results, including Gradient Boosting (GB) and XGBoost (Extreme Gradient Boosting)[2], due to their ability to capture non-linear relationships and handle structured data effectively. XGBoost supports fine-grained hyperparameter tuning, making it particularly effective in this project. Additionally, a Neural Network (NN) model was also introduced using TensorFlow and Keras[3][4], consisting of multiple dense layers with GELU activation, dropout layers, and L2 regularization to mitigate overfitting.

D. Feature engineering

To enhance model learning, several Feature Engineering (FE) techniques were introduced. These features extracted domain-specific information from the Connect Four board state, such as column heights and winning-move indicators. We

further extended the feature set by detecting whether the opponent could win immediately in each column, identifying the availability of the center column, and calculating piece counts for both players. These engineered features were concatenated with the raw inputs to form an informative representation of the game state.

A. Model improvement

After selecting XGBoost as the best-performing model, its performance was further improved by constructing additional features and tuning key hyperparameters. Specifically, the number of estimators was increased to help the model to learn more patterns, while the learning rate was decreased to improve learning stability and accuracy. These adjustments led to an improvement in the model's overall performance.

B. Evaluation

Each model was trained and validated on separate datasets. Evaluation was based primarily on accuracy, supported by confusion matrices to visualize prediction patterns. To reduce overfitting, regularization parameters and model hyperparameter tuning were applied to improve the balance between training and validation accuracy.

III. EXPERIMENTAL RESULTS

A. Experimental Setup

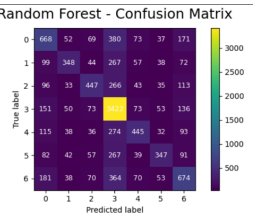
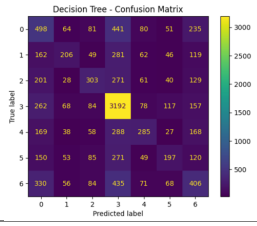
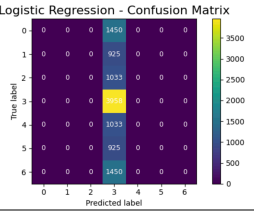
Multiple learning approaches were tested beginning with three baseline models Decision Tree, Logistic Regression, and Random Forest followed by advanced techniques including Gradient Boosting, XGBoost, and a Neural Network. All experiments were implemented in a Python environment. A validation dataset separate from training was utilized to assess performance and ensure reliable generalization across models. Feature engineering was applied to enrich the board representation and improve model accuracy.

B. *Baseline models*

Initially, three baseline models were trained Decision Tree, Logistic Regression, and Random Forest to establish a reference point for comparison. These models were evaluated using raw input data without any feature engineering. The Random Forest model achieved the highest baseline performance with an accuracy of (60.2%), followed by Decision Tree (47.2%) and Logistic Regression (36.7%).

Table 1 summarizes these results and presents the confusion matrices for each model, showing how well the models predicted each move class.

Table 1. Accuracy and Confusion Matrices for Baseline Models

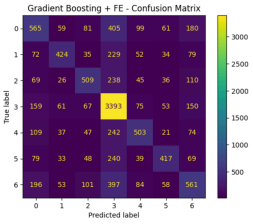
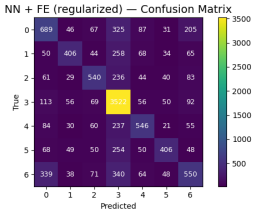
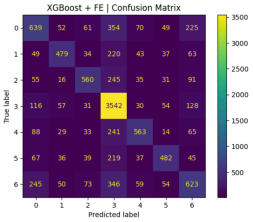
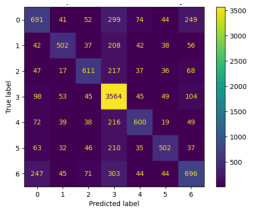
Baseline model	**Accuracy**	**Confusion Matrix**
RF	60.2%	
DT	47.2%	
LR	36.7%	

The confusion matrices also show that all models make more correct predictions for some columns than others. The Random Forest model demonstrates the most accurate and balanced results, as indicated by the stronger diagonal line in its confusion matrix. Overall, these baseline results provide a solid foundation for comparison with the feature engineering and more advanced models

C. *Advanced models*

To enhance prediction accuracy, three advanced models were tested: Gradient Boosting, XGBoost, and Neural Network. Each model was evaluated both with and without feature engineering to measure the contribution of the engineered features. The results are summarized in Table 2.

Table 2. Accuracy and Confusion Matrices for Advanced Models

Advanced model	Accuracy (without FE)	Accuracy (with FE)	Confusion Matrix
GB	58.62%	59.12%	
NN	53.09%	61.81%	
XGBoost	61.6%	63.93%	
Enhanced XGBoost	-	67.4%	

Feature engineering introduced game-specific attributes, such as column heights and potential winning positions, which helped the models better understand board dynamics. The improvement was most significant in the Neural Network, which gained nearly 9% accuracy, indicating that the new features effectively enhanced its learning capability. XGBoost also benefited moderately, improving from 61.6% to 63.9%, while Gradient Boosting remained relatively stable.

The confusion matrix for the GB, XGBoost, and NN models showed stronger diagonal dominance after applying feature engineering, confirming better class consistency.

D. *Model improvement*

We have selected XGBoost as the best performing model after comparing all baseline and advanced models. Its accuracy increased to approximately 67.4% through feature engineering and hyperparameter tuning. We improved the model by adding game-specific feature engineering, such as opponent winning threats and center column availability. Additionally, increasing the number of trees and reducing the learning rate allowed the model to capture more complex patterns while maintaining good generalization.

IV. DISCUSSION

The results across all experiments show a clear and consistent improvement when moving from baseline classical models to more advanced ensemble and deep learning approaches. The baseline group Decision Tree, Logistic Regression, and Random Forest served as the foundation for comparison. As expected, Logistic Regression recorded the weakest performance (36.7%) due to its linear nature and lack of ability to model spatial or non-linear interactions on the board. Decision Tree improved on this but remained limited by high variance and sensitivity to small changes in the data. Random Forest, on the other hand, delivered the strongest baseline accuracy (60.2%), benefiting from its ensemble structure and ability to average multiple trees, which reduced overfitting and captured more game patterns.

Transitioning to advanced models introduced more stable and powerful learning behavior. Gradient Boosting demonstrated moderate improvement over the baseline with (58.6%) accuracy and a slight increase after feature engineering. However, its simpler boosting mechanism caused it to stabilize relatively early. The Neural Network initially underperformed (53.09%) because raw board inputs alone did not provide enough structure for the model to capture meaningful spatial cues. After adding engineered features, such as column heights, immediate-win indicators, opponent threats, center availability, and piece-count differences, the NN improved significantly to (61.81%), showing that deep learning models depend heavily on well-designed, domain-specific signals.

XGBoost consistently outperformed all other models. With raw features alone, it achieved (61.6%), demonstrating its ability to handle structured data and complex interactions. After applying feature engineering, its accuracy increased to (63.93%), and with further hyperparameter tuning, adding more estimators and lowering the learning rate the enhanced XGBoost model reached (67.4%). This improvement confirms that XGBoost leverages both additional features and parameter adjustments effectively. The confusion matrices for all advanced models showed stronger diagonal patterns after feature engineering, reflecting more consistent predictions across all move classes.

Overall, the experimental results highlight three key insights. First, baseline models provide a useful starting point but lack the complexity required for strategic board-state prediction. Second, ensemble methods especially XGBoost are well-suited for Connect Four because they capture non-linear interactions and refine weak learners iteratively. Third, feature engineering plays a central role in model performance; it consistently improved accuracy across all models and was particularly impactful for the Neural Network and XGBoost. Together, these findings demonstrate that combining structured feature design with powerful ensemble methods yields the best predictive performance for this task.

V. Tuning

After evaluating all models, we applied additional tuning only to the XGBoost model, since it showed the strongest potential and the highest baseline performance. The goal was simply to push the model beyond the feature-engineered version and improve its ability to generalize to different board states.

The tuning focused on two main hyperparameters. We increased the number of estimators from 300 to 2000 so the model could learn more patterns, and we reduced the learning rate from 0.1 to 0.03 to make the updates more stable and less aggressive. These adjustments allowed the model to train more gradually while capturing deeper interactions inside the board configuration.

After applying these changes, the XGBoost accuracy improved from 63.93% (with feature engineering) to 67.4%, becoming the highest result in the entire project. This confirms that XGBoost benefits significantly from both feature engineering and careful hyperparameter tuning.

VI. Conclusion

This study explored a range of machine learning approaches for predicting optimal moves in the Connect Four game. The comparison between baseline and advanced models demonstrated that XGBoost achieved the highest performance. After applying domain-specific feature engineering and tuning key hyperparameters, the enhanced XGBoost model achieved 67.4% accuracy, outperforming both baseline and advanced models. The improvements observed across multiple models after applying feature engineering highlight the importance of integrating game-specific insights, such as column dynamics and immediate winning threats into the learning process. Overall, the results validate that ensemble learning methods, when supported by well-designed features, can outperform deeper architectures in tasks requiring spatial and strategic reasoning.

Final model XGBoost Appendix

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score,
confusion_matrix, ConfusionMatrixDisplay
from google.colab import files

Uploaded = files.upload()
uploaded = files.upload()
Uploaded = files.upload()

# load
train = pd.read_csv("train.csv")
val   = pd.read_csv("val.csv")
test  = pd.read_csv("test (1).csv")

BASE_FEATURES = [f"p{i}" for i in range(1, 43)] +
["turn"]

# helpers
def row_to_grid(row: pd.Series) -> np.ndarray:
    vals = np.array([row[f"p{i}"] for i in range(1,
43)], dtype=int)
    return vals.reshape(6, 7)
```

```
def first_empty_row(grid: np.ndarray, col: int) -> int:
    for r in range(6):
        if grid[r, col] == 0:
            return r
    return -1

def has_connect4(grid: np.ndarray, r: int, c: int, val: int) -> bool:
    dirs = [(1,0),(0,1),(1,1),(1,-1)]
    for dr, dc in dirs:
        cnt = 1
        rr, cc = r+dr, c+dc
        while 0 <= rr < 6 and 0 <= cc < 7 and grid[rr, cc] == val:
            cnt += 1; rr += dr; cc += dc
        rr, cc = r-dr, c-dc
        while 0 <= rr < 6 and 0 <= cc < 7 and grid[rr, cc] == val:
            cnt += 1; rr -= dr; cc -= dc
        if cnt >= 4:
            return True
    return False

# feature engineering
def build_features(df: pd.DataFrame, has_label: bool):
    X_raw = df[BASE_FEATURES].copy()
    H_list, W_me_list, W_opp_list, extra_list = [], [], [], []#build more features
    for _, row in df.iterrows():
        g = row_to_grid(row)
        turn_value = int(row["turn"])
        opp_value  = -turn_value
        H = np.array([(g[:, c] != 0).sum() for c in range(7)], dtype=int)
        W_me = np.zeros(7, dtype=int)
        W_opp = np.zeros(7, dtype=int)
        for c in range(7):
            r = first_empty_row(g, c)
            if r == -1:
                W_me[c] = 0
                W_opp[c] = 0
            else:
                g[r, c] = turn_value
                W_me[c] = 1 if has_connect4(g, r, c, 1) else 0
                g[r, c] = 0
                g[r, c] = opp_value
                W_opp[c] = 1 if has_connect4(g, r, c, opp_value) else 0
                g[r, c] = 0


                center_col=3
                center_available=1 if first_empty_row(g, center_col) != -1 else 0

                me_count   = int((g == turn_value).sum())
                opp_count  = int((g == opp_value).sum())
                diff_count = me_count - opp_count

        extra = np.array([center_available, me_count, opp_count, diff_count], dtype=int)

        H_list.append(H)
        W_me_list.append(W_me)
        W_opp_list.append(W_opp)
        extra_list.append(extra)

    H_df      = pd.DataFrame(np.vstack(H_list), columns=[f"h{c}" for c in range(7)])
    W_me_df   = pd.DataFrame(np.vstack(W_me_list), columns=[f"win_now_me{c}" for c in range(7)])
    W_opp_df  = pd.DataFrame(np.vstack(W_opp_list), columns=[f"win_now_opp{c}" for c in range(7)])
    extra_df  = pd.DataFrame(np.vstack(extra_list), columns=["center_available", "me_count", "opp_count", "diff_count"])

    X = pd.concat([X_raw.reset_index(drop=True), H_df, W_me_df, W_opp_df, extra_df], axis=1)
    y = df["label_move_col"].astype(int) if has_label else None
    return X, y

X_train, y_train = build_features(train, has_label=True)
X_val,   y_val   = build_features(val, has_label=True)
X_test,  _       = build_features(test, has_label=False)
```

```
# XGBoost
model = XGBClassifier(
    objective="multi:softprob",
    num_class=7,
    n_estimators=2000, #it was 300, increase the number
of trees to help model learn more
    max_depth=6,
    reg_lambda=1.0,
    reg_alpha=0.0,
    min_child_weight=1,
    gamma=0.0,
    learning_rate=0.03, #it was 0.1, decrese it to
learn accurate
    eval_metric="mlogloss",
    random_state=42
)

# fit
model.fit(X_train, y_train)

# validate
y_pred = model.predict(X_val)
acc = accuracy_score(y_val, y_pred)
print(f"XGBoost + FE Accuracy: {acc*100:.2f}%")
# confusion matrix
cm = confusion_matrix(y_val, y_pred, labels=range(7))
ConfusionMatrixDisplay(cm,
display_labels=range(7)).plot(values_format="d",
cmap="Blues")
plt.title(f"XGBoost + FE | Confusion Matrix -
Accuracy={acc*100:.2f}%")
plt.show()

# submission
test_pred = model.predict(X_test).astype(int)
pd.DataFrame({"id": range(1, len(test_pred)+1),
"label_move_col":
test_pred}).to_csv("submissionXGBoost.csv",
index=False)
print("Saved submissionXGBoost.csv")
files.download("submissionXGBoost.csv")
```

REFERENCES

[1] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[2] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.

[3] F. Chollet, *Deep Learning with Python*, 2nd ed., Manning Publications, 2021.

[4] TensorFlow, "Keras API Documentation," [Online]. Available: https://keras.io