Player Intelligence System: Comprehensive Machine Learning Report

Course: CPE342-ML | **Project:** Player Intelligence System

Table of Contents

1. Project Overview

1.1 Introduction

The **Player Intelligence System** is a comprehensive Machine Learning framework designed to address the multifaceted challenges of modern online gaming. In an industry where user experience, fair play, and monetization are paramount, traditional rule-based systems often fail to adapt to the complexity of player behavior. This project leverages advanced data science techniques to create a holistic solution that enhances game integrity, optimizes revenue, and secures user assets.

1.2 Project Objectives

The primary objective of this project is to demonstrate the practical application of Machine Learning across five critical domains of game operations:

- **Fair Play (Anti-Cheat):** Detecting malicious actors who undermine competitive integrity.
- **Personalization (Segmentation):** Understanding player archetypes to tailor experiences.
- **Business Intelligence (Monetization):** Forecasting revenue to stabilize the game economy.
- **Content Management (Computer Vision):** Automating the classification of visual assets.
- **Cybersecurity (Anomaly Detection):** Protecting player accounts from compromise.

1.3 System Architecture

The system is built as a modular pipeline, where each task represents a specialized "Intelligence Module":

- **Module 1:** Cheater Detection Engine (Supervised Classification)
- **Module 2:** Player Segmentation Engine (Unsupervised/Supervised Hybrid)
- **Module 3:** Spending Forecast Engine (Regression/Hurdle Models)
- **Module 4:** Game Vision Engine (Deep Learning/Transformers)

- **Module 5:** Account Sentinel Engine (Unsupervised Anomaly Detection)

1.4 Project Repository

The complete source code, datasets, and documentation for this project are available on GitHub:

- **GitHub Repository:** LuXeVi1/KMUTT-CPE342-PlayerIntelligenceSystem_KaggleCompetition

---

2. Task 1: Anti-Cheat (Cheater Detection)

2.1 Methodology

Objective: Identify cheaters based on gameplay patterns and user reports.

- **EDA Findings:** The dataset exhibited significant class imbalance (~35% cheaters) and structural missingness in columns like `reports_received`, where missing values often indicated "zero activity."
- **Preprocessing:**
  - **Neutral Imputation:** Implemented to mitigate "Missing = Cheater" bias using KNN and Median strategies.
  - **Feature Engineering:** Developed interaction features such as `aim_efficiency` and `kill_effectiveness` to capture multi-dimensional anomalies.
- **Model Design:** Utilized a **5-Model Stacking Ensemble** (Random Forest, XGBoost, LightGBM, CatBoost) with a Meta-XGBoost learner. ADASYN and SMOTE were applied to address class imbalance.

2.2 Evaluation & Results

- **Primary Metric: F2 Score** (Prioritizing Recall).
- **Performance:** The Meta-Model achieved an **F2 Score of 0.8369**, with a Recall of ~95% and Precision of ~56%.
- **Threshold Optimization:** The decision threshold was tuned to **0.429** to balance detection sensitivity with false positive reduction.

2.3 Insights & Interpretation

- **Key Signals:** `reports_received` and `crosshair_placement` were top predictors, confirming that combining community reports with mechanical stats is highly effective.
- **Domain Impact:** The system acts as a force multiplier, automating the detection of 95% of cheaters and preserving game integrity.

2.4 Common Mistakes & Failed Experiments

- **Failure 1: The "Missing = Cheater" Bias**
  - *Issue:* Early models achieved artificially high scores by simply flagging anyone with missing data as a cheater.
  - *Solution:* We implemented "Neutral Imputation" (filling with median instead of 0 or mean) to force the model to look at actual gameplay stats rather than data artifacts.
- **Failure 2: Over-reliance on Accuracy**
  - *Issue:* Using raw `accuracy` scores caused false positives for skilled legitimate players (smurfs).

- *Solution:* We created "Consistency" features (`kill_consistency`) and interaction terms (`reports_x_crosshair`). A skilled player has good crosshair placement; a cheater often doesn't, despite hitting shots.

---

3. Task 2: Player Segmentation

3.1 Methodology

Objective: Classify players into distinct behavioral segments (e.g., Casual, Hardcore, Whale).

- **EDA Findings:** A distinct class imbalance was observed, with the majority being Casual players (Class 0, ~39%) and a minority of High-Value players (Class 3, ~15%).
- **Preprocessing:**
  - **Behavioral Metrics:** Engineered features like `engagement_score` and `spending_intensity`.
  - **Balancing:** Applied **SMOTE** to perfectly balance the training set across all 4 classes.
- **Model Design:** Employed a **Soft Voting Classifier** ensemble (XGBoost, LightGBM, CatBoost) to stabilize predictions across borderline cases.

3.2 Evaluation & Results

- **Metric: F1-Macro Score**.
- **Performance:** Achieved an average **F1-Macro of 0.8085**, indicating robust separation between all player segments.

3.3 Insights & Interpretation

- **Segmentation Logic:** The model successfully distinguished segments using a mix of time-investment and monetary-investment features.
- **Strategic Value:** Enables targeted marketing strategies, such as retention campaigns for Casuals and VIP services for Whales.

3.4 Common Mistakes & Failed Experiments

- **Feature Selection Oversight:** We couldn't achieve a higher prediction score because we dropped the player ID and never included it in the features. Turns out, it's the most important feature for this classification task.

---

4. Task 3: Spending Prediction

4.1 Methodology

Objective: Predict the exact spending amount for a player in the next 30 days.

- **EDA Findings:** The data was **Zero-Inflated** (~48% non-spenders) and highly right-skewed among spenders.
- **Preprocessing:** Applied `log1p` transformation to the target variable and engineered historical spending features.

- **Model Design:** Implemented a **Two-Stage Hurdle Model** using **XGBClassifier** and **LGBMClassifier** for classification, and **XGBRegressor** and **LGBMRegressor** for regression:
    1. **Classification:** Predict probability of spending > 0.
    2. **Regression:** Predict amount for identified spenders.

4.2 Evaluation & Results

- **Metric: Normalized MAE**.
- **Performance:** Achieved an OOF Normalized MAE of **0.7881**.
- **Thresholding:** Optimized the classification threshold to **0.5055**.

4.3 Insights & Interpretation

- **Predictive Power:** Historical spending is the strongest predictor of future behavior ("stickiness").
- **Business Application:** Allows finance teams to forecast revenue and marketing teams to focus on high-probability conversion targets.

4.4 Common Mistakes & Failed Experiments

- **Hyperparameter Tuning:** Initially, we used a small number of `n_estimators`, which prevented the XGBoost model from fully exploring and capturing the features to its full potential. After increasing the value of `n_estimators`, the model's prediction score improved significantly.

---

5. Task 4: Game Image Classification

5.1 Methodology

Objective: Classify low-resolution game screenshots into genre categories.

- **EDA Findings:** Images were low quality (~144p), and there was a significant distribution shift between training and test sets.
- **Preprocessing:** Used **Albumentations** for robust augmentation.
- **Feature Extraction:** Leveraged **Transfer Learning** with a pre-trained **Vision Transformer (ViT)** to extract 768-dimensional embeddings.
- **Model Design:** A **Neural Network (MLP)** head trained on ViT features outperformed XGBoost.

5.2 Evaluation & Results

- **Metric: Macro F1 Score**.
- **Performance:** Achieved a Validation F1 of **0.6515**.

5.3 Insights & Interpretation

- **ViT Superiority:** Vision Transformers proved exceptionally robust to low-resolution data compared to traditional CNNs.
- **Scalability:** The feature extraction approach allows for scalable content analysis without retraining heavy backbones.

5.4 Common Mistakes & Failed Experiments

- **Failure 1: XGBoost on Embeddings**
  - *Experiment:* We hypothesized that XGBoost would dominate as it did in Tasks 1-3.
  - *Result:* It performed poorly (F1 0.48 vs NN 0.65).
  - *Lesson:* Gradient Boosting is King of Tabular Data, but Neural Networks are Queen of Embeddings.
- **Failure 2: Ignoring Class Weights**
  - *Issue:* The model initially ignored the minority classes.
  - *Solution:* We implemented "Soft Class Weights" (Square Root of inverse frequency) to gently nudge the model towards minority classes without causing instability.

---

6. Task 5: Account Security

6.1 Methodology

Objective: Detect compromised accounts using unsupervised anomaly detection.

- **EDA Findings:** The problem was unsupervised (no labels). Anomalies were characterized by sudden spikes in volatility or behavioral drift.
- **Preprocessing:** Engineered **Vectorized Temporal Features** (Volatility, Trend, Spike Ratio) to capture changes over time.
- **Model Design:** A **Rank-Based Ensemble** of Isolation Forest, One-Class SVM, and Autoencoder.

6.2 Evaluation & Results

- **Outcome:** Detected **1,165 anomalies** (4.50% of the test set).
- **Consensus:** The ensemble approach reduced false positives by requiring agreement across diverse algorithms.

6.3 Insights & Interpretation

- **Security Profiles:** Successfully identified "Booster" accounts (high skill volatility) and "Hacked" accounts (asset liquidation patterns).
- **Operational Use:** Serves as a triage layer for Trust & Safety teams.

6.4 Common Mistakes & Failed Experiments

- **Failure 1: Raw Score Averaging**
  - *Experiment:* Averaging the raw output of Isolation Forest (-0.5 to 0.5) with Autoencoder MSE (0.0 to 10.0).
  - *Result:* The Autoencoder dominated the decision simply because its numbers were bigger.
  - *Solution:* **Rank Averaging** normalized the contributions, allowing each model to vote equally based on relative ordering.
- **Failure 2: Ignoring Time**
  - *Experiment:* Treating the 4 time steps as independent features.
  - *Result:* Missed the "change" signal. A high value might be normal for a whale, but a *sudden jump* to a high value is suspicious. Vectorized temporal features fixed this.

---

7. Team Collaboration & Roles

University: King Mongkut's University of Technology Thonburi

Member Name	Student ID	Role	Key Responsibilities
Nuttaya Ngamsard	65070501094	**Lead CV Engineer (Task 4)**	Image Classification, ViT Implementation
Ponprathan Kuearung	66070501036	**Lead Data Scientist (Task 3)**	Spending Prediction, Two-Stage Modeling
Ratchamongkol Mongkoldit	66070501046	**Lead Data Scientist (Task 1)**	Anti-Cheat Pipeline, Imbalance Handling (ADASYN/SMOTE)
Arkkhanirut Pandej	66070501062	**Lead Security Engineer (Task 5)**	Anomaly Detection, Temporal Vectorization
Khunnapat Aubontara	66070501068	**Lead Data Scientist (Task 2)**	Player Segmentation, Behavioral Feature Engineering

Key Teamwork Takeaways

- **Unified Architecture:** Enforced strict OOP standards (Config, Logger, Pipeline) across all tasks to facilitate cross-debugging.
- **Knowledge Sharing:** Successfully transferred techniques like Stacking and Interaction Features between tasks.
- **Iterative Workflow:** Maintained a versioned development process (`v1` -> `v3`) to ensure continuous improvement.

---

8. Conclusion & Future Outlook

8.1 Summary

The Player Intelligence System successfully met its objectives, delivering high-performance models across all five domains. From achieving **95% Recall** in anti-cheat to **81% F1** in segmentation, the project proves the viability of ML in gaming.

8.2 Key Lessons

- **Data Quality is King:** Feature engineering consistently yielded better ROI than hyperparameter tuning.
- **Ensembles are Essential:** Stacking diverse models is the most reliable way to improve robustness and generalization.
- **Context Matters:** Understanding the domain (e.g., "Zero-Inflation" in spending) is crucial for model selection.

8.3 Future Improvements

- **Real-Time Inference:** Migrating Anti-Cheat and Security modules to a stream processing architecture (Kafka/Flink) for instant action.

- **Graph Neural Networks:** Implementing GNNs to detect cheater rings and fraud clusters based on social connections.
- **Explainable AI (XAI):** Integrating SHAP values to provide transparent reasons for automated bans.