📌 Machine Learning Class Fall 2025 Assignment 2 Q5 Specification:

📅 Deadline: 2025-10-31 23:59

📌 Assignment Type

Individual Assignment - Each student must submit their own work

📜 Submission Policy

⏳ You have a total of two extra days available for late submissions across all assignments.

- If you want to use late days for an assignment, please send an email to ml-elsalab@googlegroups.com specifying the number of late days you intend to use (in units of days).

🚫 No copying or cheating; otherwise, you will receive zero points for this assignment.

⚙️ Rules

- You may use any standard packages or libraries (e.g., numpy, pandas, scikit-learn, tensorflow, pytorch, lightgbm, etc.) to implement this question.
- The work must be your own implementation of the method, even if the algorithm or approach itself is described in a paper or other resource.
    - You are not allowed to use any open-source implementations of this task that are publicly available, such as:
        - Copying code from GitHub repositories, blogs, or other students.
        - Directly using implementations released with research papers.
    - Otherwise, you will receive zero points for this assignment.

Who play this game?

🎯 Objective

The goal of this assignment is to implement a Machine Learning model that can identify which player played a given game of Go, based on the game record.
The dataset contains Go games annotated with player identities.
Your task is to build a model that learns individual play styles from extracted game features and accurately predicts the player responsible for each game.

🛑 Core Challenge: Generalization Ability

- Your model should not rely on memorizing the specific moves or sequences of particular players.
- The training and test sets do not contain the same games — test samples come from **different, unseen game records**.
- Your model must demonstrate the ability to **generalize player identification** to games that were **not observed during training**.

☑ Task Requirements

1. Build a Machine Learning model (e.g., classification or embedding-based) to predict which player played a given Go game.
2. Evaluate your model's performance in terms of:
    - Accuracy — the percentage of games where the predicted player identity matches the true player.

🗁 Dataset

- Training Data
    - The training set consists of 200 files (1.sgf, 2.sgf, …, 200.sgf), each corresponding to a single player.
    - Each file contains approximately 2,000 games played by that player.

    - All game records within the same file are guaranteed to come from the same individual, ensuring consistent player identity labels.
- Test Data
    - The test data is divided into two subsets: query and candidate.
    - Each subset contains 600 files (1.sgf, 2.sgf, …, 600.sgf).
    - Each file represents a single player, containing approximately 100 games from that player.
- Prediction Task
    - For each file in the query set, your model must determine which player (in the candidate set) played those games.
    - Each prediction should output the ID of the matching candidate player (e.g., if the correct player corresponds to 1.sgf in the candidate set, the output label should be 1).
- Data Format

  Example snippet from 2.sgf in training set:

```
plpl666666
(;GM[1]FF[4]  SZ[19]  GN[]  DT[2018-07-28]  PB[plpl666666]  PW[goddns] KM[0]HA[0]RU[Japanese]AP[GNU
(;GM[1]FF[4]  SZ[19]  GN[]  DT[2020-07-30]  PB[WDY0102]  PW[plpl666666] KM[0]HA[0]RU[Japanese]AP[GNU
(;GM[1]FF[4]  SZ[19]  GN[]  DT[2019-07-31]  PB[plpl666666]  PW[陆战一旅] KM[0]HA[0]RU[Japanese]AP[GN
(;GM[1]FF[4]  SZ[19]  GN[]  DT[2019-12-04]  PB[sunday6512]  PW[plpl666666] KM[0]HA[0]RU[Japanese]AP[
(;GM[1]FF[4]  SZ[19]  GN[]  DT[2021-06-06]  PB[crh321]  PW[plpl666666] KM[375]HA[0]RU[Chinese]AP[GNU
(;GM[1]FF[4]  SZ[19]  GN[]  DT[2021-05-01]  PB[plpl666666]  PW[早行人] KM[375]HA[0]RU[Chinese]AP[GNU
(;GM[1]FF[4]  SZ[19]  GN[]  DT[2021-06-16]  PB[plpl666666]  PW[DOCTOR01] KM[375]HA[0]RU[Chinese]AP[G
(;GM[1]FF[4]  SZ[19]  GN[]  DT[2019-12-01]  PB[plpl666666]  PW[春日阳] KM[0]HA[0]RU[Japanese]AP[GNU
(;GM[1]FF[4]  SZ[19]  GN[]  DT[2019-12-01]  PB[plpl666666]  PW[春日阳] KM[0]HA[0]RU[Japanese]AP[GNU
```

- SGF File Structure Description
    - Each file corresponds to a single Go player.
    - The first line of the file contains the player's name or identifier (e.g., plpl666666).
    - The remaining lines record multiple games played by that player, stored in the Smart Game Format (SGF).
    - Each game record is enclosed within parentheses (; ...), containing metadata about the match such as date, rules, and player information.
    - Within each SGF record:
        - PB[...] denotes the player who played as Black.
        - PW[...] denotes the player who played as White.
        - For example:
            - For the first sgf games

        - **Black player (PB):** plpl666666
        - **White player (PW):** goddns
    - Therefore, in this particular game, the player plpl666666 was playing as **Black**.

🏆 Evaluation

Your submission will be scored based on:

- **Player Identification Accuracy:** the fraction of query samples where the predicted player identity correctly matches the true player in the candidate set.

◇ Environment Setup (Optional but Recommended)

◇ Environment Setup (Optional but Recommended)

- You may use the provided Go game environment to simplify feature extraction and training.
- However, this environment is optional — you are free to implement your own version if you prefer.

🔗 Repository

```
git@github.com:gcobs104628/ML-Assignment2-Q5.git
```

🥊 1. Launch Podman container

- After cloning, you can enter the pre-configured Podman environment with:

```
./scripts/start-container.sh
```

- This container includes all necessary dependencies for both C++ compilation and Python bindings.

  ⚙️ 2. Build the Go environment

  - Inside the container, compile the C++ Go engine with:

    ```
    ./scripts/build.sh go
    ```

  - This command builds the C++ backend and generates the pybind11 interface under build/go/.

  🧠 3. Using it from Python

  - Once built, you can directly access the C++ Go environment in Python via pybind:

    ```
    _temps = __import__(f'build.{game_type}', globals(), locals(), ['style_py'], 0)

    style_py = _temps.style_py

    style_py.load_config_file(conf_file)
    ```

  - The style_py module provides full access to the backend, including SGF loading, feature extraction, and data iteration.

⚠️ Note:
This environment is provided for your convenience.
You may use it directly, modify it, or build your own implementation — as long as your code functions correctly and follows the assignment specification.

🛠️ What You Need to Implement

📌 You must submit your prediction result (e.g. submission.csv) to Kaggle.
(https://www.kaggle.com/t/3561ff53a80d466c9f929e473600abcb)

- You are only responsible for submitting the predicted results of your model.
- Summit Rule
    - Team Name
        - Please use your student ID as the team name.
        - If you are an auditor, add _a after your student ID (e.g., b12345678_a).
        - Submissions that do not follow the team naming rule will be considered invalid and treated as not submitted.
    - Submission Limit
        - Each team may submit up to 10 times per day.

📝 Grading Policy (30 Points + Up to 20 Bonus Points)

🎯 Base Score (based on private leaderboard accuracy)

- ≥ 63% → 5 points
- ≥ 68% → 10 points
- ≥ 73% → 15 points
- ≥ 78% → 20 points
- ≥ 83% → 25 points
- ≥ 88% → 30 points
- ≥ 90% → eligible for leaderboard bonus (see below)

🏆 Leaderboard Bonus

- Only participants with private leaderboard accuracy ≥ 90% are eligible for extra bonus points.
- Bonus is based on private leaderboard ranking:
    - Top 5%: +20 points
    - Next 5%: +19 points

    - Next 5%: +18 points
    - … decreasing by 1 point for every additional 5%
    - Until bonus reaches 0

Note:

- The private leaderboard will be revealed on **2025/11/07**.
- ⚠ If you do **not** make a Kaggle submission, you will receive **0 points** for this question, regardless of other files you submit.

📥 Submission Requirements (To NTU Cool)

Your submission should be compressed into a .zip file named after your student ID.

- Example: If your student ID is D13922036, your submission file should be named D13922036.zip.

📦 Submission Requirements

- Required Files:
    - Q5.py
        - Inference code
        - Must be executable and directly produce submission.csv (in the correct Kaggle format).
        - This is to ensure that your Kaggle submission is indeed generated by your own code.
        - ⚠ If Q5.py cannot generate a valid submission.csv, your score for Q5 will be 0, regardless of your Kaggle leaderboard performance.
- Training Code(s):
    - You must also submit your training script. This can be:
        - A separate training.py, or
        - Integrated into Q5.py (in which case, Q5.py must default to evaluation mode, and training must only be enabled via an additional switch/flag).

        - Your training files may consist of multiple scripts or even entire folders (e.g., models_architecture/, utils/, etc.) if needed.
        - Please provide a readme with instructions for running training
    - ⚠ If no training code is submitted, or if the training code cannot be executed, your score for Q5 will be 0, regardless of leaderboard results.
- Weight Files:
    - If Q5.py loads a pre-trained weight file (e.g., .pkl, .txt, .npy), you must include this file in your .zip submission.
    - If the weight file is too large to upload, include an automatic download command inside Q5.py (for example, using gdown or another command-line download utility).
- Environment:
    - You may include your own environment or the files cloned from the provided GitHub repository
      o You are allowed but not required to use this environment — it is optional.
      o However, whether you use your own implementation or the provided one, you must include the corresponding environment files in your submission (even if you did not modify them).

- Important Note:
    - ❌ Do not include your Q1–Q4 .ipynb files in the .zip submission. Only files related to Q5 should be included.
- Example
    - <student_id>.zip

      ├── Q5.py # Required. Must generate submission.csv when run.

      ├── training files # (Optional) If training is separated from Q5.py

      ├── weight files # any required pretrained weights

      ├── readme # Instructions (How to run your training?)

├──Environment # Your own environment, or the files cloned from the provided GitHub repository. (Even if you do not modify them, you must still include these files in your submission.)

⚠ Any submission format errors will result in score deductions.

🎮 Enjoy the assignment! 😀