COMP 2019 Assignment 2 – Pacman Learning

Please submit your solution via LEARNONLINE. Submission instructions are given at the end of this assignment.

This assessment is due on **Sunday, 11 June 2017, 11:55 PM.**
This assessment is worth 20% of the total marks.

This assessment consists of six questions. You will be asked to complete coding tasks, interpret the results of the machine learning algorithms, and provide a brief written analysis of your findings.

In this assignment, you will be applying machine learning techniques to learn how to imitate a Pacman agent based on recorded games. You will create a Pacman agent that can learn to copy another agent's behaviour simply by observing examples of that behaviour. You will be using this idea to mimic various Pacman agents by using recorded games as training examples for a classifier. Your agent will then run the classifier at each game state to try and determine which action would be taken by the observed agent. You will compare and evaluate the outcomes of different machine learning techniques at this task.

Before starting this assignment, ensure that you have a good understanding of the Python programming language, the Pacman game that was used in Practical 2 (how the GameState class works), and an overall understanding of machine learning training and evaluation methods using the scikit-learn python library (Practical 3). You will need a working Python 3.x system with the sklearn package installed.

Download the Pacman code for the assignment from the course website (file Pacman Code for Assignment 2.zip). This is a minor extension of the program used in Practical 2.

The archive includes a set of recorded games for five different Pacman agents in the `data/` directory:

- FoodAgent: moves towards the closest food, not caring about anything else
- LeftTurnAgent: turns left at every opportunity
- RandomAgent: selects an action with uniform probability
- StopAgent: sits idle
- SuicideAgent: moves towards the closest ghost

For each agent, you are given two data sets, one for training the agent (`*_train.pkl`), and one for evaluating its performance (`*_test.pkl`).

You can load a data set file using the function `load_dataset(filepath)`.

Each data set contains a `dict` object with two keys: key 'states' maps to a list of `GameState` objects, and 'actions' maps to a list of actions taken by the agent in the corresponding game state.

The recorded games are made available in folder recorded_games/. These should not be used in any way for answering this assignment. You can however replay these games to get a better understanding of how the individual agents behave. To replay a recorded game, run

```
python3 pacman.py --replay recorded_games/<recorded-game-*>
```

Question 1: Feature Extraction

To apply machine learning for behaviour cloning, we must first convert each `GameState` object into a set of features that the classifier will use. You will be extracting a set of features of your choosing, one set for each action (North, East, South, West, Stop). You will be using the same feature extraction function to learn classifiers for all the agents.

Implement function `extract_action_features(gameState,action)` that extracts the features reflecting the outcome of applying action 'action' in 'gameState'. If action cannot legally be performed in gameState, return some default values.

Your function must work correctly when called from function '`prepare_dataset`' to construct the full set of features for the gameState. This feature set will combine the features extracted from gameState for each action into an overall feature set used by the classifiers in subsequent questions.

Add code to the relevant section in `assignment2.py`

Write a short section in your technical report document in which you briefly document the features you have chosen.

Hint: When extracting features, you may consider actual values of key information (e.g., number of food) or indicators derived from differences between that information before and after applying the action (e.g. was food eaten?). Think about what information each of the agents may require for choosing an action and tailor your feature selection accordingly.

Question 2: Training

Train a classifier that predicts the action given the features extracted in Question 1.

Use a Decision Tree Classifier (`sklearn.tree.DecisionTreeClassifier`).

Add code to the relevant section in `assignment2.py`

Question 3: Evaluation

Evaluate the accuracy of each classifier on the training set and test set, and report your findings.

The training and test sets you shall use depend on your student identifier number (1XXXXXXXX).
If the last digit in your student identifier is

- odd: use data sets for agents FoodAgent, RandomAgent, StopAgent, SuicideAgent
- even: use data sets for agents LeftTurnAgent, RandomAgent, StopAgent, SuicideAgent

If you find considerably different performance (> 5% difference in accuracy) for different agents, or differences between training and test set performance, explain why these differences may arise.

What is the expected accuracy of the classifier if we applied it to a new game of Pacman?

Moreover, try to assess if the resulting classifier may have over-fitted the training data. Describe how you assessed this and what could be done to improve the situation. You may want to refer to the parameter description in the documentation of the `DecisionTreeClassifier` learner for inspiration.

You may want to calculate accuracy measures, consult contingency tables, and inspect the decision tree model to inform your answers.

Add code to the relevant section in `assignment2.py`

Document the outcomes of your analysis in a separate section in your technical report.

Hint: You should be achieving accuracy of close to 80% or higher for most agents (except RandomAgent). Otherwise, revisit Question 1 and verify that the features that you have extracted indeed correlate with the target actions to be predicted.

Hint: You may find that the classifier may predict an action that is illegal in the given game state. To ensure that only legal actions are predicted, use the decision tree classifier to generate probabilities (use method `predict_proba`) for each of the five actions, and select the *legal* action with highest probability as the output of the classifier.

Hint: Function `extract_decisiontree_rules()` may be useful to inspect the decision tree.

Question 4: Training #2

Construct an additional classifier and compare its performance with that of the classifier created in Question 2.

The classifier you should train depends on your student identifier number (1XXXXXXXX).
If the last digit in your student identifier is

- 0,1,2,3, or 4: use a K-Nearest-Neighbour (`sklearn.neighbors.KNeighborsClassifier`) classifier
- 5,6,7,8, or 9: use a Multi-Layer-Perceptron (`sklearn.neural_network.MLPClassifier`) classifier

Compare the classifier's performance with the classifier created in Question 2.

Add code to the relevant section in `assignment2.py`

Document your analysis in the technical report.

Hint: If you get a warning about convergence of the `MLPClassifier`, increase its `max_iter` parameter. (5000 iterations should be more than sufficient.)

Question 5: Parameter Optimisation

For optimal performance of the classifier trained in Question 4 it is beneficial to select the optimal value of some key parameters.

- For the k-Nearest-Neighbour classifier, the value of *n_neighbors* should be determined. Consider $n\_neighbors \in [1, \dots 10]$.
- For the multi-layer perceptron, the number of hidden layers and number of units in each hidden layer should be determined (parameter *hidden_layer_sizes*). Consider 1-2 hidden layers, with up to 10 units in the first hidden layer and up to 5 units in the second hidden layer.

Determine the optimal value(s) of the parameter(s) relevant for the classifier trained in Question 4, and assess the performance of the optimised classifier relative to that trained in Question 4.

Ensure that you follow an appropriate training and optimisation procedure that will ensure results are unbiased. Document what you have done.

Add code to the relevant section in `assignment2.py`

Document how you have determined the optimal parameter values in the technical report.

Question 6: Game integration

Integrate the best performing classifier into the Pacman game.

Implement the relevant methods in class `ClassifierAgent`.

The file path to the training set should be passed in via the command line as follows:

```
python3 pacman.py -p ClassifierAgent -a training_set=/path/to/data/file.pkl
```

Add code to the relevant section in `classifierAgents.py`

Submission Instructions

Submit a single ZIP file containing the following:

1. The **source code** files (`assignment2.py` and `classifierAgents.py`) and any other source code files (*.py) required to run your project.
2. A **PDF** file named `report.pdf` containing the **technical report** containing your answers to Questions 1-5. Each question should be answered in a separate section delimited by a heading.

Do *not* include compiled bytecode (*.pyc), layout files, recorded games, training and test data (*.pkl), and Eclipse project files.

Your program should not be writing to any files or access the network.

Your program may use only packages that are installed in the standard Anaconda Python 3.x distribution, unless approved by the Course Coordinator.

Please structure your program well, and comment your code where appropriate.

Marking Criteria

Programs that do not run on Mac OS X in the Mac pool (Anaconda Python 3.x) will receive zero marks.

Question 1: Appropriate features extracted Program runs correctly Features well documented in report	20%
Question 2: Program runs correctly Appropriate training procedure was used	20%
Question 3: Program runs correctly Appropriate evaluation procedure was used Results well documented and analysed in the report	20%
Question 4: Program runs correctly Outcomes documented and analysed in the report	10%
Question 5: Program runs correctly Appropriate optimisation procedure was used Outcomes documented and analysed in the report	20%
Question 6: Program runs correctly and game can be played	10%