pacman

August 14, 2019

Jing (Thomas) Zhang

[101]:
```
from scipy import stats
import numpy as np
import pandas as pd
from collections import defaultdict, Counter
%matplotlib inline
import matplotlib.pyplot as plt
from functools import reduce
from tqdm import tqdm
from sklearn.linear_model import LinearRegression
```

0.1 Problem 4a

Clearly describe your evaluation function. What is the high-level motivation? Also talk about what else you tried, what worked and what didn't.

I tried to manually tune it but I was not very lucky and I received many low scores, so I looked at the helper functions in `util.py` and found Counter, which can be essentially used as a vector, and its `__mul__` method can be used as dot product. Then I manually calculated a list of features, most of which are pretty standard (can be directed acquired from `GameState` class):

- score: Current Score
- distFood: Manhattan Distance to the closest food
- nFood: Number of Food left
- distScared: Manhattan Distance to the closest scared ghost
- distGhost: Manhattan Distance to the closest non-scared ghost
- nCap: Number of capsules left
- distCap: Manhattan Distance to the closest capsule
- DisWall: Distance to wall

Then, I then generated 2000 sample runs basically just by randomly generate the weights and record the scores (print to file), and ran a linear regression on it. The problem with this approach is that I don't really know how to interpret the coefficient, since they are one side of the equation - I need to know how to boost Y, which is the score, but I don't have any control over X, and I don't even know the distribution of X in each game. However, at least they provided me some guidelines for later manual tuning. I simply record the learned coefficient and the weights that generate the max score I've seen, and started manual tuning.

I also tried boosting the magnifying (squared) the number of ghosts or number of food but they didn't really work. If I had more time I would manually craft some strategies and explore more.

Sample Run:

```
Pacman died! Score: 170
Pacman emerges victorious! Score: 1097
Pacman emerges victorious! Score: 897
Pacman died! Score: -302
Pacman emerges victorious! Score: 902
Pacman emerges victorious! Score: 868
Pacman died! Score: -323
Pacman died! Score: -206
Pacman died! Score: 19
Pacman died! Score: -91
Pacman died! Score: -364
Pacman died! Score: -278
Pacman emerges victorious! Score: 1082
Pacman died! Score: -156
Pacman died! Score: 273
Pacman died! Score: -189
Pacman died! Score: -17
Pacman died! Score: -421
Pacman died! Score: -17
Pacman died! Score: 140
Average Score: 154.2
Scores:        170, 1097, 897, -302, 902, 868, -323, -206, 19, -91, -364, -278, 1082, -156, 27
Win Rate:      5/20 (0.25)
Record:        Loss, Win, Win, Loss, Win, Win, Loss, Loss, Loss, Loss, Loss, Loss, Win, Loss, L
Average score of winning games: 0
```

[114]:
```python
X, Y = [], []
with open('./train.csv') as f:
    for line in f:
        Xi, Yi = line.strip().split(':')
        X.append([float(xi.strip()) for xi in Xi.split(',')])
        Y.append(float(Yi))
X = np.array(X)
Y = np.array(Y)
```

[123]:
```python
X.shape
```

[123]: (2001L, 8L)

[115]:
```python
model = LinearRegression()
model.fit(X, Y)
model.score(X, Y)
```

[115]: 0.6133273547242812

[116]:
```
model.coef_
```

[116]:
```
array([68.22991189, -6.45392203, -4.24008784, -0.48176285, -3.49515544,
        3.01077641, -6.35613891, 11.06070588])
```

[122]:
```
[round(x, 1) for x in X[np.argmax(Y)]]
```

[122]:
```
[2.4, -1.7, -2.2, -3.9, -4.0, 0.5, 2.5, 1.2]
```