CS 70 Discrete Mathematics and Probability Theory
Summer 2011 Kamil PA 3

Due Thursday, August 4, 11:59pm

You must complete this programming assignment on your own. Remember that you have three slip days that you can allocate between the three programming assignments. Note that there is a short grace period (on the order of minutes, not hours) after the above deadline to account for clock discrepancies, last minute submission hiccups, etc.

This assignment will likely take you longer to complete than PA1 or PA2, so we recommend that you get started early.

Overview

Consider a round of two-player Texas hold 'em poker. (Those of you unfamiliar with the game may wish to refer to the last section for a brief overview of the rules.) Each player receives two cards, and there are five community cards on the table at the end of the round. A player only sees his or her own cards as well as the table cards and must evaluate this information in order to determine the likelihood that the player's hand is better than the opponent's.

The distribution of cards in a particular round is the outcome of a random experiment.

(a) The cards in a round consist of two cards for each player and five common cards. What is the size of the sample space $|\Omega|$?

There are far too many outcomes to analyze in a reasonable amount of time, so instead, we work in the conditional sample space Ω' given G, where G is the event that the player receives the two given cards and the table contains the five given cards. In this new sample space, there are $\binom{45}{2} = 990$ equally likely outcomes corresponding to the $\binom{45}{2}$ possible pairs of cards that the opponent can receive from the remaining 45 cards in the deck.

In this programming assignment, you will analyze all 990 possible outcomes. Let O be a random variable that is 0 if the opponent's hand type is a high card, 1 if it is a pair, 2 if it is a two pair, and so on in order of decreasing probability for five-card hands (see the Poker Hands section below), up to 8 if it is a straight flush. Let W be a random variable that is 1 if the player wins, -1 if the opponent wins, and 0 if they draw. Your job is to compute the distributions[1] of O and W, as well as their expectations and variances.

As with previous assignments, you may use any of the following programming languages: Scheme, Java, Python, C, or C++. (Note that though you may use any of these languages, we can only provide you skeletons in a subset of the languages. It may be beneficial to look over a skeleton for another language if one is not available in the language of your choice.) You have access to `stk`, `javac` and `java`, `python`, `gcc`, and `g++` from your class accounts. Skeletons for Scheme, Java, and Python with some useful auxiliary

[1] Note that in the old sample space Ω, these are actually conditional distributions given G.

functions are provided at `http://www.cs.berkeley.edu/~kamil/cs70/pa/pa3/`. We strongly recommend that you use one of the skeletons for this assignment. The skeletons already implement the code to analyze a poker game, such as determining each player's specific hand, the type of each player's hand, and the winner of the game.

If you use one of our skeletons, then there are only two tasks you need to complete:

1. Generate the 990 possible outcomes, using our skeleton code to construct each outcome.
2. Perform the probability calculations over all these outcomes, with the opponent's hand type and the winner of an outcome already determined by the skeleton code.

For the languages other than Scheme, input will be passed in on the command line, with the first two inputs the string representations of the player's cards and the remaining five that of the community cards. A card is represented by a capital letter corresponding to the first letter of its suit followed by its value, so the 2 of diamonds is "D2" and the ace of clubs is "CA". The skeletons for Java and Python process these arguments for you.

Your program should produce output similar to the following:

```
> java pa3sol D5 SQ S8 S3 D3 C7 D4
Player's cards: D5 SQ
Table cards:    S8 S3 D3 C7 D4
Player's hand:  (S3, D3, C7, S8, SQ)
Player's type:  Pair

Number of games analyzed: 990

Distribution of O:
  High Card              Pr[O = 0] = 0.0
  Pair                   Pr[O = 1] = 0.5060606060606089
  Two Pair               Pr[O = 2] = 0.3848484848484869
  Three of a Kind        Pr[O = 3] = 0.06868686868686871
  Straight               Pr[O = 4] = 0.012121212121212118
  Flush                  Pr[O = 5] = 0.0
  Full House             Pr[O = 6] = 0.027272727272727264
  Four of a Kind         Pr[O = 7] = 0.00101010101010101
  Straight Flush         Pr[O = 8] = 0.0
E(O) = 1.7010101010100829
Var(O) = 0.9954535251505852

Distribution of W:
  Lose   Pr[W = -1] = 0.756565656565661
  Draw   Pr[W = 0] = 0.033333333333333326
  Win    Pr[W = 1] = 0.21010101010101104
E(W) = -0.5464646464646495
Var(W) = 0.6680430568309381
```

You will find code in the skeletons to print out these statistics in the proper format.

Your program need not perform any error checking (e.g. that you receive exactly seven cards as input, and that each input is the proper string representation of a card). Note that your program may take a little while to crunch through the numbers, especially if you use Scheme or Python. As long as it runs in a reasonable amount of time, say within a minute or so on the instructional machines, you need not worry about making it any faster.

Special Notes for Scheme

If you use Scheme, you should instead define a function called `analyze-game` that takes in a list of cards, each of which is represented as a string, as input. You should then display the required output. You can find pre-defined functions in the skeleton to process input as well as analyze individual games. Your task, if you use our skeleton, is to fill in the `construct-all-games` function that generates a list of all possible game outcomes and the `compute-stats` function that takes in this list and computes the required probabilities, expectations, and variances and returns a string that contains this information. You may find the `stats->string` function in the skeleton helpful in generating a string from the distributions of O and W and their expectations and variances. Some of the other skeleton functions may also prove useful. (Note that while some of the skeleton functions may use features of Scheme that you are unfamiliar with, such as looping, you should be able to complete your solution using just recursion. You are welcome to use more advanced features if you like, however.)

Testing Your Code

We have made a poker game generator written in Java available at `http://www.cs.berkeley.edu/~kamil/cs70/pa/pa3/pa3gen.java` to help you test your code. Once you copy it to your class account, you can compile it as follows:

```
javac pa3gen.java
```

You can then run it as follows:

```
java pa3gen
```

The program produces seven distinct cards, the first two of which are the player's with the remaining five the community cards. This output can then be passed to your program as input at the command line.

There is also a Python version of the game generator at `http://www.cs.berkeley.edu/~kamil/cs70/pa/pa3/pa3gen.py`. The Scheme skeleton contains a `generate-game` function that similarly generates a random set of seven cards.

Submission Instructions

You must submit your assignment electronically with the following command from your class account:

```
submit pa3
```

Two files are required in the directory from which you run the above command. A `README` file described below and a ZIP archive called `pa3.zip`. This archive should include all files necessary to compile and run your program. You can construct this archive using the following command:

```
zip pa3.zip <file1> ...  <fileN>
```

If you are adding directories, you will need to pass the `-r` flag to `zip`. As a sanity check, run

```
unzip -l pa3.zip
```

afterwards to make sure that all intended files are included in the archive. **We will not be able to grade incomplete submissions, so make sure to double check that all files are included.**

The README File

In your `README` file, include explicit instructions on how to build and run your code. In addition, answer the question (a) above as well as the following questions:

(b) Did you prefer using a program to compute probabilities or calculating them on paper?

(c) How long did this assignment take you to complete?

A Primer on Poker

In this section, we review the rules of poker for those of you who are unfamiliar with them. Note that you do not actually have to be familiar with the rules in order to complete the assignment, since we provide the logic to analyze a game in the skeletons.

Poker Hands

A poker *hand* consists of five cards, which are scored according to type. (There are other rules to break ties, but we won't consider them here.) The possible types of hands are a high card, a pair, a two pair, a three of a kind, a straight, a flush, a full house, a four of a kind, and a straight flush. A deck of cards contains 52 cards. There are thirteen different values (the values 2 to 10, the jack, the queen, the king, and the ace), and four *suits* (diamonds, clubs, hearts, and spades), and each card has a value and a suit. As a result, there are $\binom{52}{5} = 2598960$ possible distinct poker hands. A random five-card poker hand can be any one of these 2595960 possibilities with equal probability.

(a) A *straight flush* is the best type of poker hand. It is a set of cards that are all the same suit and that are in order, $\{C4, C5, C6, C7, C8\}$ for example. The ace is allowed to be either 1 or 14, so $\{DA, D2, D3, D4, D5\}$ and $\{D10, DJ, DQ, DK, DA\}$ are straights, but $\{DJ, DQ, DK, DA, D2\}$ is not. A straight flush is uniquely determined by its suit and its high card, and there are 4 choices of suit and 10 choices of high card (5 through A), so there are 40 straight flushes.

(b) A hand that contains four of the same value, such as $\{C2, DA, SA, CA, HA\}$, is a *four of a kind*. There are 13 choices for which value is repeated and 48 choices for the *kicker* (the remaining card), so there are $13 \times 48 = 624$ four of a kinds

(c) A *full house* is a hand that contains a triple and a pair, such as $\{D3, S3, C3, H7, D7\}$. There are 13 choices for the value of the triple, 12 for the pair, $\binom{4}{3}$ for the cards that make up the triple, and $\binom{4}{2}$ for those that make up the pair, for a total of $13 \times 12 \times \binom{4}{3} \times \binom{4}{2} = 3744$ full houses.

(d) A *flush* is a hand in which all the cards are of the same suit but in which the cards are not in order. There are four choices for the suit, and then five values must be chosen out of 13, so the number of flushes is $4 \times \binom{13}{5} = 5148$. However, we have included the straight flushes in this count, so the total number of plain flushes is just $5148 - 40 = 5108$.

(e) A *straight* is a set of cards in order but not all of the same suit. As with straight flushes, there are 10 choices for the high value in a straight, which forces the values of the other cards. Then all that is left is to choose the suits of each card, and there are 4 choices for each card. Thus the number of straights is $10 \times 4^5 = 10240$. However, we have included straight flushes once again, so the number of plain straights is actually 10200.

(f) A *three of a kind* has three cards of the same value, but with the remaining cards having different values, unlike in a full house. There are 13 choices for the the value of the triple and $\binom{12}{2}$ for each of the remaining values. There are $\binom{4}{3}$ choices of cards for the repeated value and 4 for each of the singles. Thus, there are a total of $13 \times \binom{12}{2} \times \binom{4}{3} \times 4^2 = 54912$ different three of a kinds.

(g) A *two pair* is a hand that contains two pairs and a remaining card that has a different value from either of the pairs. There are $\binom{13}{2}$ choices for the two repeated values and 11 choices for the remaining value. There are $\binom{4}{2}$ card choices for each of the pairs and 4 for the single. So the total number of two pairs is $\binom{13}{2} \times \binom{11}{1} \times \binom{4}{2}^2 \times 4 = 123552$.

(h) A *pair* is a hand that contains one value that occurs twice and three values that occur once. There are 13 choices for the value of the repeat and $\binom{4}{2}$ for the cards with that value. Then there are $\binom{12}{3}$ choices for the remaining values, each of which has a choice of 4 cards, so there are $13 \times \binom{4}{2} \times \binom{12}{3} \times 4^3 = 1098240$ pairs.

(i) The last type of hand is a *high card*, in which all five values are different and not in order and the cards don't all have the same suit. There are $\binom{13}{5} - 10$ choices for the values (subtracting the choices that result in straights). There are $4^5 - 4$ choices for the actual cards (subtracting the choices that result in flushes). Thus the number of high cards is $(\binom{13}{5} - 10) \times (4^5 - 4) = 1302540$.

In five-card poker, the unconditional probability of getting any particular type of hand is the number of hands of that type divided by the total number of hands, since we have a uniform probability space. Note that in seven-card poker, the probabilities will be different, since a player's hand is the best possible five-card combination of the the seven cards. (In fact, a high card hand is less likely than a pair or even a two pair.)

The less likely a hand is in five-card poker, the stronger it is (so in seven-card poker, a pair still beats a high card, since the five-card probabilities are used to compute strength). If two hands are of the same type, then there are tie breakers. For example, a pair of aces will beat a pair of tens. We won't go into detail on tie breakers here.

Texas Hold 'Em

There are many variants of poker, and perhaps the most popular today is *Texas hold 'em*. In this game, each player receives two cards, which are hidden from the rest of the players. Then three common cards, known as the *flop* are dealt on the table. Another card, the *turn*, is dealt, after which a final card, the *river* is dealt, for a total of five community cards on the table. Each player's hand is the best five card combination from the player's two private cards and the five common cards. At each point in a particular round (before the flop, after the flop, after the turn, and after the river), players can place bets, which the others must either *call*, matching the bet, *fold*, forfeiting the round and any money previously bet in it, or *raise*, increasing the

size of the bet. (There are more specific rules as to what is actually allowed, but we won't concern ourselves with them here.) At the end of the round, the player who has the best hand among the remaining players wins all money bet in the round. (The money that has been bet at any point is known as the *pot*.)

In order to be successful at Texas hold 'em, a player must use a strategy based on mathematical probabilities as well as psychology. In this assignment, we have focused on the mathematics.

See `http://en.wikipedia.org/wiki/Texas_hold_%27em` for more details on Texas hold 'em poker.