STAT/CS 94 Fall 2015 Adhikari HW07, Due: 10/14/15

NAME: SID:

As usual: Please write your answers on a printed copy of this assignment, in the space provided. If you print this assignment yourself, please print exactly this document *double-sided* on a single sheet of paper. (There will be a small penalty for not following this instruction; it makes the graders' jobs more difficult.)

Suggestion: It's hard to write code without a computer, but it's a useful skill, and it will be necessary to write some code on the exams for this class. We recommend you try the coding questions in this assignment that way first for practice, then check your answer on a computer (you can use a cell in a blank notebook or a copy of a lab at `ds8.berkeley.edu`).

Problem 1 *Punnett Plants?*

A species of plant comes in four different varieties. According to a genetics model, the varieties appear like random draws with replacement from a population in which there is a 9:3:3:1 ratio of varieties A, B, C, and D respectively.

A lab has n plants of this species. You can assume that n is large.

(a) True or false (explain): To test whether the model is good, we should use a permutation test.

(b) Fill in the blank with a fraction, and provide your statistical reasoning: If the model were good, the number of plants of variety A would be around _____ of n.

(c) A researcher tests the null hypothesis that the model is good, and gets a P-value of 0.5% (that's one-half of 1%). Circle all that are true among the statements below, and explain.

(i) There is only a 0.5% chance that the model is good.

(ii) There is a 99.5% chance that the model is bad.

(iii) The P-value of 0.5% was computed assuming the model is good.

(iv) The result of the test is highly statistically significant, and the data support the hypothesis that the model is not good.

Answer:

(a) False. Permutation tests are useful for checking whether an apparent association between two linked datasets (like the income and water usage of water districts) would be likely to happen even if the two datasets were actually independent. In this case, we have just one dataset – a list of n varieties – and we want to check whether it is sampled from a particular distribution. So a permutation test is not useful here.

(b) $\frac{9}{16}$. Since n is large, the law of averages says that the empirical proportion of the event that variety A appears is close to the probability of that event. A 9:3:3:1 ratio means that the probability of the first event (that variety A appears) is $\frac{9}{9+3+3+1} = \frac{9}{16}$, at least *assuming the model is correct.*

(c) (iii) and (iv) are true.

A P-value is the probability of observing a test statistic as extreme or more extreme than the test statistic we computed from the sample, *assuming the null is true.* (Exactly which test statistic was computed was left unspecified in the problem, but it's not relevant.) (i) and (ii) are exactly equivalent statements, and they assert that the P-value is the probability that the null hypothesis is true once we've seen the data. That's just not the same probability as the definition of a P-value.

(iii) matches the definition, so it's true.

The first statement in (iv) is true because a P-value of 0.5% is typically considered highly statistically significant (though really that depends on the situation). The second statement in (iv) is true because a small P-value does roughly mean that the data support rejecting the null hypothesis. It's a slightly subtle fact that this is different from (i) and (ii). A small P-value means that we should doubt the null hypothesis *more than we did before we saw the data*, but our level of doubt about the null hypothesis also depends on how confident we were in it *before we saw the data*.

Problem 2 *Heads Helper*

Define a function `heads` that takes a positive integer `n` as its argument and returns a (random) simulated value of the number of heads in `n` tosses of a fair coin.

Answer:

```
def heads(n):
    outcomes = Table([["heads","tails"]],["toss"])
    tosses = outcomes.sample(n, with_replacement=True)
    return np.count_nonzero(tosses["toss"] == "heads")
```

A less verbose way to do this is:

```
def heads(n):
    return sum(np.random.randint(2, size=n))
```

Or, if you happen to know that this has a distribution named the *binomial* distribution:

```
def heads(n):
    HEADS_PROBABILITY = .5
    return np.random.binomial(n, HEADS_PROBABILITY)
```

Problem 3 *Soda Scramble*

In "blind taste tests," participants are given unmarked containers of two different beverages that look identical and taste similar, and are asked to identify which is which by drinking from the containers. The Pepsi Challenge of the 1970's compared Pepsi and Coke in this way.

Suppose n people take a blind taste test, and k of them make the correct identification. How would you test whether or not the result was just due to chance? Answer the question in the following steps.

(a) State the null hypothesis as a clearly defined set of assumptions.

(b) State an alternative hypothesis. Please be consistent with the question posed in the statement of the problem.

(c) Pick a test statistic and justify your choice. You're welcome to assume that n is even.

(d) Write code to perform the test by repeated sampling. Use r repetitions of the sampling process. You can use any function that you have defined in this homework, but please do not simply call functions defined in class. Your code should return an empirical P-value and a conclusion. Use 3% as your cutoff for "small" P.

Answer:

(a) Null hypothesis: All participants in the study cannot distinguish between Pepsi and Coke in a blind taste test, and each chooses one of the two drinks uniformly at random to identify as Pepsi. (This hypothesis implies that if we asked these participants to take the test again, the number of correct identifications would have the same distribution as the number of heads in n flips of a fair coin. We will really test this

implication of the hypothesis; if the conclusion of an implication is false, then the thing that implied it must also be false.)

Another reasonable null hypothesis is that all *people* (not just the n participants in the study) cannot distinguish between Pepsi and Coke in a blind taste test, and they choose one of the two drinks uniformly at random to identify as Pepsi, and the n people we saw were an arbitrarily-selected (not necessarily random) sample of that population. In this case the tests will be the same, since both hypotheses imply that the chance of a correct identification is 1/2.

(b) Alternative hypothesis: All participants have chance, say, .1 of actually knowing which beverage is which, and a .9 chance of choosing randomly, so that the chance of a correct identification is $.1 + .9 \times .5 = .55$.

(Note that there are many potential alternative hypotheses. It's also okay to write down an alternative hypothesis that covers a wide range of possibilities. For example, two very reasonable alternative hypotheses are: "for each participant, the probability of that participant making a correct identification is p, where p is some probability other than .5," and "for each participant, the probability of that participant making a correct identification is p, where p is some probability strictly greater than .5." Also note that, thus far in this class, we haven't actually used an alternative hypothesis for anything in the statistical tests we've been doing, so identifying alternatives is just an exercise for thought. That is not always true.)

(c) If n is large and the null hypothesis is true, the proportion of correct identifications (which is $\frac{k}{n}$ in the data we observed) should be close to 1/2, the probability that an identification is correct (by the law of averages). So one reasonable test statistic is the total variation distance between the empirical distribution of guesses (correct or incorrect) and the probability distribution of guesses. For the data we observed, this is just $|\frac{1}{2} - \frac{k}{n}|$.

Another reasonable test statistic is just the number of correct identifications. (So the value of this test statistic for the data we observed was k.) If the null hypothesis is true, this should be close to $n/2$, and otherwise it will be (typically) something else. There are many variations on this (like the number of correct identifications minus $n/2$, or the proportion of correct identifications, or the proportion of correct identifications minus 1/2) that are essentially equivalent, in the sense that the P-values we compute will be exactly the same.

(We haven't seen in this class how to compare the efficacy of different test statistics. Unfortunately that's a somewhat technical thing and a subject for Statistics 135 and 210A, among other classes. For now, just remember that a good test statistic should be a number that is *typically one way* if the null hypothesis is true and *typically a different way* if the null hypothesis is false.)

(d)
```
def test_statistic(num_correct_identifications, n):
    return abs((num_correct_identifications / n) - .5)

# The proportion of numbers in number_array greater than num.
def proportion_greater(number_array, num):
    return np.count_nonzero(number_array > num) / len(number_array)

# Returns the P-value associated with the null hypothesis.
def test_taste_null_hypothesis(data, num_repetitions):
    n = len(data)
    actual_test_statistic = test_statistic(np.count_nonzero(data), n)
    statistics_under_null = []
    for test_index in np.arange(num_repetitions):
        # We can reuse the heads() function we wrote earlier!
        num_correct_identifications = heads(n)
        statistics_under_null.append(test_statistic(num_correct_identifications, n))
    p_value = proportion_greater(np.array(statistics_under_null), actual_test_statistic)
    return p_value
```

```
# We assume that taste_test_data is the name of the identification dataset.
# We further assume that it is an array of boolean values, True if the
# corresponding participant make a correct identification and False
# otherwise.
p_value = test_taste_null_hypothesis(taste_test_data, r)
accept_null = p_value >= 0.03
[p_value, accept_null]
```

Note: The problem used imprecise language to describe how your code should produce the P-value and the outcome of the test. It should have said something like: "Your code can contain any number of Python statements. The last statement should be an expression whose value somehow contains the P-value and a boolean value that is False if you reject the null." As is, we accepted any answer that computed (or defined a function that computed) the P-value and a boolean value that somehow said whether you rejected the null.