q1

April 8, 2021

Part A: Probability of all three labels predicted correctly

Consider the case where we draw 3 independent samples randomly from the population. Each sample has a probability 3/4 of being correctly labeled. Thus, to the probability that k samples are labeled correctly is modeled by a binomial distribution with the following pmf:

$$P_X(k) = \binom{3}{k}(\frac{3}{4})^k(\frac{1}{4})^{3-k}$$

and therefore the probability of all 3 successes is given by

$$P_X(k) = \binom{3}{3}(\frac{3}{4})^3(\frac{1}{4})^0 = \frac{27}{64} \approx .4219$$

Part B: Probability of 7 correct samples in 10 draws (independently selected)

Again, we are counting the number of successes in a fixed number of trials, which leads us to a binomial distribution. There are 10 choose 7 ways to get 7 correct samples out of 10 samples, while the probability of an individual correct prediction is 3/4, and the probability of an individual incorrect prediction is 1/4. Therefore,

$$P_X(7) = \binom{10}{7}(\frac{3}{4})^7(\frac{1}{4})^3 = \frac{27}{64} \approx .2503$$

q2

April 8, 2021

```
[1]: from sklearn.datasets import load_wine
     from math import sqrt
     from scipy.stats import binom
     from random import randint
     from numpy import array
     from numpy.random import shuffle
```

Part A: Working with “Class 2” of the Wine Dataset

```
[2]: wine = load_wine()

     X = wine['data']
     target = wine['target']
```

```
[3]: # determine what fraction of the data instances belong to class 2
     fraction_of_2 = sum(target == 2) / len(target)
     print("Fraction of Instances belonging to Class 2: {}".format(fraction_of_2))
```

```
Fraction of Instances belonging to Class 2: 0.2696629213483146
```

Suppose we take a sample size of n=40, and for each independent sample, the probability of getting an instance of class 2 is the value we just determined. Then, the number of class 2 samples is modeled by a binomial distribution with

$$E[X] = np = 40(.2697)$$

```
[4]: # determine the integer closed to the expected value of the number of class 2␣
      ↪instances in a sample size of 40

     print("Int closest to expected value: {}".format(round(fraction_of_2 * 40)))
```

```
Int closest to expected value: 11
```

Part B: Probability that the fraction will be $<$ 5 units from p

```
[5]: interval = (fraction_of_2 - .05, fraction_of_2 + .05)
     print(interval)
```

```
# this is the intervals of frequencies that are less than or equal to .05 units␣
↪away from p
# we can iterate through every value between 0 and 40 and see if the current␣
↪value falls within the range

satisfying_integers = []
bound1 = interval[0]
bound2 = interval[1]

for k in range(0, 40+1):

    # if we have k successes, we can write this as a ratio
    proportion = k/40

    if proportion > bound1 and proportion < bound2:
        satisfying_integers.append(k)

print(satisfying_integers)
```

```
(0.2196629213483146, 0.3196629213483146)
[9, 10, 11, 12]
```

[11]:
```
# now that we have the integers satifsying this condtion, we can use the pmf of␣
↪the binomial distribution to find the probability that we get this many␣
↪successes

probability_in_range = sum([binom.pmf(k, 40, fraction_of_2) for k in␣
↪satisfying_integers])
print("Probability of being in range: {}".format(round(probability_in_range,␣
↪3)))
```

```
Probability of being in range: 0.524
```

Part C: Use Monte Carlo Simulation to Estimate the probability value in the preceding part

For choosing the number of iterations, we need to consider that the magnitude of error is given by $1/\sqrt{N}$ where N is the number of iterations. We set up the following inequality to determine how many iterations we should use, considering that we want to be within three decimals places:

$$\frac{1}{\sqrt{N}} \leq .0005 \implies N \geq 4,000,000$$

[19]:
```
num_iter = 4 * 10**6
successes = 0

# create a copy of the target array so we can shuffle but preserve the original␣
↪target values
```

```
target_copy = target
for _ in range(num_iter):

    # simulate process of getting a sampling a single set of size 40 from the␣
 ↪original dataset
    sample = []

    # shuffle(target_copy)
    # sample = target_copy[:40]

    for __ in range(40):
        index = randint(0, len(target_copy)-1)
        # shuffle(target_copy)
        sample.append(target_copy[index])

    # typecast sample array to numpy array for boolean indexing
    sample = array(sample)

    num_success = sum(sample == 2)

    if num_success in satisfying_integers:
        successes += 1

print("Probability of being in range (monte): {}".format(round(successes/
 ↪num_iter, 3)))
```

```
Probability of being in range (monte): 0.524
```

```
[ ]:
```

Q3: Check How Replacement of Missing Values w/ Mean Affects the Distribution

Suppose we have a data sample of size N = m + r, where m of the samples are numerical values and r of the samples are missing values. Suppose that the mean of the numerical values is μ and the stdev is σ. Then we can find the mean of the entire dataset as follows:

$$\mu_r = \frac{1}{m+r}\sum_{i=1}^{m+r} = \frac{1}{m+r}\left(\sum_{i=1}^{m} + \sum_{i=m+1}^{r}\right) = \frac{1}{m+r}\left(m\mu + r\mu_{rp}\right)$$

where μ_{rp} is the mean value of the missing values (whatever we choose them to be). If we set all the missing values to be the mean of the numerical attributes, then we get that

$$\mu_r = \frac{1}{m+r}(m\mu + r\mu) = \frac{\mu(m+r)}{m+r} = \mu$$

We see that replacing the missing values with the mean does not affect the overall mean of the data,however, we can proceed similarly to show that the standard deviation (and variance) do in fact change when we replace missing values with the mean.

$$\sigma_r^2 = \frac{1}{m+r-1}\sum_{i=1}^{m+r}(X_i-\mu)^2 = \frac{1}{m+r-1}\left(\sum_{i=1}^{m}(X_i-\mu)^2 + \sum_{i=m+1}^{r}(X_i-\mu)^2\right)$$

However, remember that we replaced each missing value with the mean of the non-missing data, μ. So the entire summation from m + 1 to r is just 0. Therefore,

$$\sigma_r^2 = \frac{1}{m+n-1}\sum_{i=1}^{m}(X_i-\mu) = \frac{1}{m+n-1}\sigma^2$$

and we conclude that

$$\sigma_r = \frac{\sigma}{\sqrt{N-1}}$$

So, even though the mean of the dataset does not change when we replace the missing values with the average of the non-missing values, the variances becomes vastly smaller, by a factor of

$$\sqrt{N-1}$$

where N is the number of samples total

Loading [MathJax]/jax/output/HTML-CSS/jax.js

q4

April 8, 2021

Part A: Sampling With/Without Replacement

[50]:
```python
from numpy import mean, std, array
from numpy.random import choice, sample
from sklearn.datasets import load_wine
from random import sample
```

[51]:
```python
# load the wine dataset
wine = load_wine()
target = wine['target']
```

[52]:
```python
# generate a size 40 sample w/ replacement
def X():

    return sum(array([choice(target) for _ in range(40)]) == 2)
```

[53]:
```python
# generate a size 40 sample w/o replacement
def Y():

    return sum(array(sample(list(target), 40)) == 2)
```

Part B: Return Monte Carlo Estimates of the Expected Value and Stdev of a Given RV

[54]:
```python
def rvStats(RV):

    num_iter = 10**4

    outcomes = array([RV() for _ in range(num_iter)])

    # I know how to do these on paper but I also know how to write efficient␣
↪code
    E_e = mean(outcomes)
    S_e = std(outcomes)

    return E_e, S_e
```

Part C: Compute the Expectation and Stdev for the RV's described Previously

[55]:
```python
# exp, stdev of sampling w/ replacement
exp, stdev = rvStats(X)

print("Expectation (w/ replacement): {}".format(exp))
print("Stdev (w/ replacement): {}".format(stdev))
```

```
Expectation (w/ replacement): 10.8047
Stdev (w/ replacement): 2.7822936419436393
```

[56]:
```python
# exp, stdev of sampling w/O replacement
exp, stdev = rvStats(Y)

print("Expectation (w/o replacement): {}".format(exp))
print("Stdev (w/o replacement): {}".format(stdev))
```

```
Expectation (w/o replacement): 10.7761
Stdev (w/o replacement): 2.485431308646449
```

Based on the monte carlo simulations, we see that the expected value between sampling with and without replacement are about the same, while the standard deviation for sampling without replacement is slightly lower than that of sampling with replacement. The expected value for both the binomial distribution and hypergeometric distribution are np, so this makes sense.