CMPSC 448: Machine Learning. Homework 3: Randomization.
Due: February 7, 2020

1. Instructions

- You cannot look at anyone else's code.
- Fill in and upload hw3.py to gradescope.
- All code (except import statements) in h3.py should be inside functions (importing homework1.py should not cause code to execute).
- Code must have comments and any constants should be stored in a variable defined near the top of your file.

Important! Read this! Do not ask how many iterations to run your code. Part of the assignment is to figure this out. Gradescope provides feedback on expected answers and how close they should be. Your job is to figure out how many iterations you need to **reliably** get low variance and accurate answers.

2. Programming Randomness

A good statistical random number library is **numpy.random** (however, do not use it for security applications). It can be used to sample from a variety of distributions. In particular:

```
import numpy as np
# Get 1 coin flip with heads (i.e., 1) probability = p
x = 1 if np.random.random() <= p else 0
# Get an array of 100 coin flips with heads probability = 0.8
np.random.binomial(n=1, p=0.8, size=100)
# Get the sum of 200 coin flips with heads probability = 0.8
np.random.binomial(n=200, p=0.8)
```

It can also be used to permute arrays:

```
import numpy as np
# create an array
x = np.array([1,2,3,4,5]) # could also do x=np.arange(1,6)
np.random.shuffle(x) # modifies x, permutes it
# we can also shuffle columns of matrices:
y = np.array([[1, 2],
              [2, 4],
              [3, 6],
              [4, 8]]
np.random.shuffle(y[:, 0]) # modifies y, shuffles first column only
np.random.shuffle(y[:, 1]) # modifies y, shuffles second column only
np.random.shuffle(y[0, :]) # modifies y, shuffles the first row only
```

3. Randomization and Permutation

Question 1. *Suppose we have n models. Each model has* 60% *generalization accuracy in the sense that for any new data point, it has a* 60% *chance of getting it correct. So, on a random validation set of size 10, the accuracy of the model looks like 10 coin flips with p(heads)*= 0.6. *The empirical accuracy of a model is the number it got right divided by the validation set size. For any random validation set of size 10, some models will have an empirical accuracy of 0.6, some will underperform (i.e., empirical accuracy* < 0.6*, and some will overperform). The empirically best performing model is the one with highest empirical accuracy on the validation set.*

We are interested in finding the expected empirical accuracy of the empirically best performing model for test sets of size 10. In hw3.py, fill in `question1(num_models)` *that will use simulations to estimate (and return) this number (note that* `num_models` *is the same as n.). The point of this homework is to demonstrate why you should not trust of models over validation sets that are used for model selection.*

Note that you must choose the right number of simulations runs so that your result is stable (without timing out on gradescope). You should also write your code efficiently (hint: see the discussion of numpy).

Question 2. *Suppose we have* 1 *"good" model that has* 60% *generalization accuracy and* $n-1$ *"so-so" models with* 50% *generalization accuracy. In hw3.py, fill in* `question2(num_models)` *which should return a tuple (a,b,c) where*

- ***a*** *is the probability (estimated via simulations) that the "good" model outperforms (has empirical accuracy strictly better than) the other models on a random test set of size 20.*
- ***b*** *is the estimated probability that the "good" model gets tied for the best empirical accuracy on a random test set of size 20.*
- ***c*** *is the estimated probability that the "good" model loses on a random test set of size 20.*

Note that `num_models` *is the same as* n*.*

Question 3. *We have 2 models evaluated on a test set of size* n*. Their prediction results are stored in a 2-dimensional numpy array* ***results****. The first column contains results for the first model and the second column contains results for the second model.* ***results****[i,0] is 1 if the first model got example* i *correct (otherwise it is 0) and* ***results*****$[j, 1]$ *is 1 if the second model got example* j *correct (otherwise it is 0). We are interested in determining if the models are performing correlated predictions on this dataset.*

Our test statistic is the number of rows for which the two models had the same performance (i.e. the number of rows for which results$[j, 0] ==$*results*$[j, 1]$*.*

In hw3.py, fill in `question3(results)` *that estimates the p-value (probability, under the null hypothesis, of seeing a test statistic this extreme).*