ECE 50024: Homework 6
Manish Kumar Krishne Gowda, 0033682812 (Spring 2023)

Exercise 1.

1.a.

a) since learning algo picks hyp that match D the most,
& there are 3 y_n in D that are •,
the algo picks $h(x) = h_1 = \bullet$ for all 3 x's i.e x_6, x_7, x_8.

Thus final $g = [\circ, \bullet, \bullet, \circ, \bullet, \bullet, \bullet, \bullet]$

g will match.

3 out-samples once (f_8)
2 out-samples thrice (f_4, f_6, f_7)
1 out-samples thrice (f_2, f_3, f_5)
no out-sample once (f_1)

b) now the algo picks $h(x) = h_2 = \circ$

$\Rightarrow g = [\circ, \bullet, \bullet, \circ, \bullet, \circ, \circ, \circ]$

g will match

3 out-samples once (f_1)
2 out-samples thrice (f_2, f_3, f_5)
1 out-samples thrice (f_4, f_6, f_7)
no out-sample once (f_8),

c) when $\mathcal{H} = H = \{h\}$, where h is XOR opr^n

$g = [\circ, \bullet, \bullet, \circ, \bullet, \circ, \circ, \bullet]$

Now g will match

3 out-samples once (f_2)
2 out-samples thrice (f_1, f_4, f_6)
1 out-samples thrice (f_3, f_5, f_8)
no out-sample once (f_7)

Exercise 2.

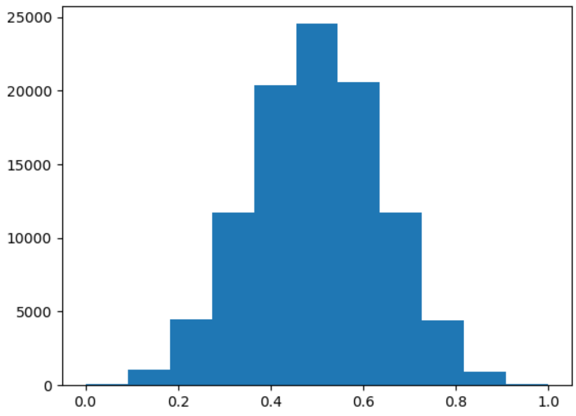
```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
from random import randint
import math

TOTAL_EXP_NUM = 100000
NUM_EACH_COIN_FLIP = 10
TOTAL_NUM_COINS = 1000
x_err = np.arange(0,0.55,0.05)
prob_v1 = list(range(x_err.shape[0]))
prob_vrand = list(range(x_err.shape[0]))
prob_vmin = list(range(x_err.shape[0]))
hoeffding_bound = 2*np.exp(-2*NUM_EACH_COIN_FLIP*np.square(x_err))
exp_arr = np.zeros((TOTAL_NUM_COINS,NUM_EACH_COIN_FLIP))
v1 = list(range(TOTAL_EXP_NUM))
v_rand = list(range(TOTAL_EXP_NUM))
v_min = list(range(TOTAL_EXP_NUM))


print(exp_arr.shape)

def run_experiment():
        for coin in range (0,TOTAL_NUM_COINS):
                for flip_num in range (0,NUM_EACH_COIN_FLIP):
                        exp_arr[coin,flip_num] = randint(0, 1)

v_value = [0,0,0]
def get_proportions():
        exp1 = exp_arr[0,:]
        #print(exp1)
        v_value[0] = (NUM_EACH_COIN_FLIP-
np.count_nonzero(exp1))/NUM_EACH_COIN_FLIP
        rand_coin = randint(0,TOTAL_NUM_COINS-1)
        exp_rand =  exp_arr[rand_coin,:]
        #print(rand_coin)
        #print(exp_rand)
        v_value[1] = (NUM_EACH_COIN_FLIP-
np.count_nonzero(exp_rand))/NUM_EACH_COIN_FLIP
        tails_for_coin = np.count_nonzero(exp_arr,axis=1)
        min_heads = NUM_EACH_COIN_FLIP - tails_for_coin.max()
        v_value[2] = min_heads/NUM_EACH_COIN_FLIP
        #print(v_value)
        return v_value

for exp_num in range (0,TOTAL_EXP_NUM):
        print(exp_num)
        run_experiment()
        v_value = get_proportions()
        v1[exp_num] = v_value[0]
        v_rand[exp_num] = v_value[1]
        v_min[exp_num] = v_value[2]

v1 = np.array(v1)
v_rand = np.array(v_rand)
```

```python
v_min = np.array(v_min)
bins=np.arange(0,1.2,0.1)


hist_v1, bin_count = np.histogram(v1, bins=11, range=(0,1))
pdf_v1 = hist_v1/sum(hist_v1)
cdf_v1 = np.cumsum(pdf_v1)
'''
plt.plot(bins[:-1], pdf_v1, color="red", label="PDF")
plt.show()
'''

hist_vrand,bin_count = np.histogram(v_rand, bins=11, range=(0,1))
pdf_vrand = hist_vrand/sum(hist_vrand)
cdf_vrand = np.cumsum(pdf_vrand)

hist_vmin, bin_count = np.histogram(v_min, bins=11, range=(0,1))
pdf_vmin = hist_vmin/sum(hist_vmin)
cdf_vmin = np.cumsum(pdf_vmin)


plt.hist(v1, bins=11, range=(0,1))
plt.show()
plt.hist(v_rand, bins=11, range=(0,1))
plt.show()
plt.hist(v_min, bins=11, range=(0,1))
plt.show()


for i in range (x_err.shape[0]):
        low_bound = math.ceil((0.5-x_err[i])*10-1)
        up_bound = math.floor((0.5+x_err[i])*10+1)
        if low_bound < 0:
                low_bound = 0
        if up_bound > 10:
                up_bound = 10
        prob_v1[i] = cdf_v1[low_bound] + (1-cdf_v1[up_bound])
        prob_vrand[i] = cdf_vrand[low_bound] + (1-cdf_vrand[up_bound])
        prob_vmin[i] = cdf_vmin[low_bound] + (1-cdf_vmin[up_bound])


plt.plot(x_err,prob_v1, color="blue")
plt.plot(x_err,prob_vrand, color="green")
plt.plot(x_err,prob_vmin, color="orange")
plt.plot(x_err,hoeffding_bound, color="red")
plt.show()
```

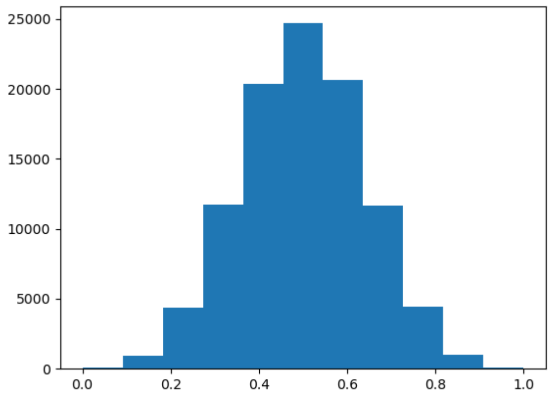
a) $\mu_1 = 0.5$, $\mu_{rand} = 0.5$ and $\mu_{min} = 0.5$

b)

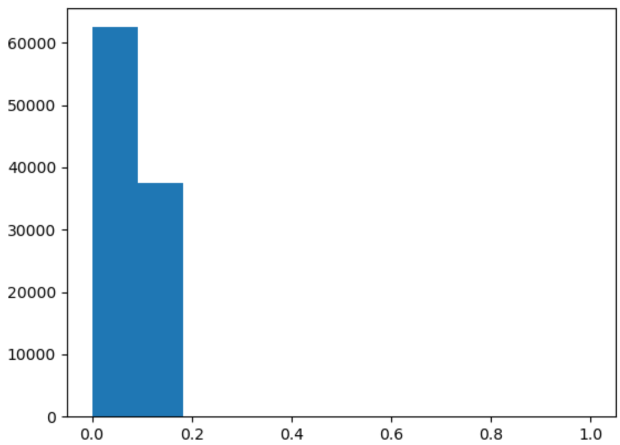
V1 histogram



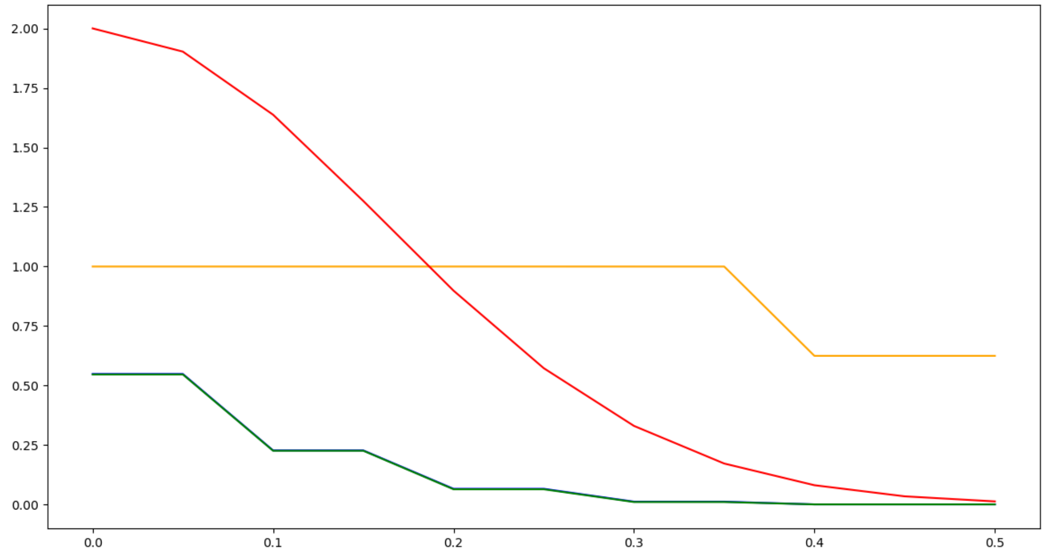
Vrand histogram



Vmin histogram



C)



Red → Hoeffding's bound
Blue and Green → P(|V1 −µ1| > ϵ), P(|Vrand −µrand| > ϵ) respectively
Orange → P(|Vmin −µmin| > ϵ)

D)
Clearly both V1 and Vrand are randomly picked coins. Hence the fraction of heads for a large number of such experiments will be 0.5. But in Case of Vmin, the coin with minimum frequency of head is always chose. So, the choice is restricted to a particular coin out of the whole lot and the distribution of that is considered. Thus E[V] is a low value 0 < x < 0.1. Thus, in sample error deviates from out sample error by a lot.

Exercise 3. Project Checkpoint #6

1.

The paper focuses on the problem of training deep neural networks on datasets that contain class imbalance or noisy labels, which can lead to biased models that perform poorly on minority classes or on unseen examples at test time. The paper proposes a method to reweight the training examples so that the neural network learns to give more importance to the minority classes and to the examples that are more difficult to classify correctly.
The paper does identify and clearly describe similar works in the related work section. The authors discuss several existing techniques that aim to address the problem of class imbalance or noisy labels, including:

- Data augmentation: Generating synthetic examples to balance the dataset or to reduce the impact of noisy labels
- Cost-sensitive learning: Assigning different misclassification costs to different classes or examples to reduce the impact of class imbalance or noisy labels
- Sample reweighting: Assigning different weights to different examples to reduce the impact of class imbalance or noisy labels

For each technique, the paper provides a brief description of the method and its advantages and disadvantages. The authors also compare their proposed method with these existing techniques in the experimental section of the paper to show the effectiveness of their approach.

2.

The paper explains the mathematical derivation of the proposed method in a clear and detailed manner.
The paper presents a new loss function that reweights the training examples based on their difficulty to classify correctly. The authors derive the new loss function from a general formulation of the expected loss over the training set, and they show how the reweighting factor can be computed using an auxiliary network that predicts the difficulty of each example.
The paper provides a step-by-step explanation of the mathematical derivation, including the derivation of the new loss function, the formulation of the reweighting factor, and the computation of the auxiliary network. The authors also provide visual aids and examples to help the reader understand the concepts and formulas.
In addition, the paper provides a theoretical analysis of the proposed method, showing that it can reduce the generalization error and the impact of class imbalance and noisy labels. The authors also provide a comprehensive experimental evaluation of the method on several benchmark datasets to validate the effectiveness of the proposed approach.

3.

The paper does not explicitly discuss technical difficulties that may arise in the reimplementation of the proposed method. However, the paper provides a detailed description of the proposed method and includes all the necessary equations, hyperparameters, and implementation details to facilitate the reproducibility of the results.

4. Reimplementation

I am implemnenting the project in PyTorch, along with code for running experiments on various datasets and evaluating the performance of the method. It also includes pre-trained models like resnet, Lenet, etc for reproducing the results reported in the paper.

All these info will be covered in detail in the final paper