HW4 — ISYE6416

Ashish Dhiman — ashish.dhiman9@gatech.edu

May 7, 2023

Contents

1 Question 1: Conceptual questions

1.1 Data fit complexity in regression tree

We control the data fit complexity in a regression tree by first growing the tree greedily until a minimum node size is reached. Then pruning of the tree is performed by optimising the cost function:

$$C(\alpha) = MSELoss(J) + \alpha(|nodes_j|)$$

where $nodes_j$ is number of terminal nodes in sub tree J.
We can control overfitting by various parameters such as:

1. cost complexity parameter or α, increasing α decreases overfitting
2. min number of children of a node, again this limits the depth of the tree and only allows further splits, if ther is enough population at a node.

The data fit complexity in a regression tree can be controlled by a variety of parameters such as:

1.2 Bagging and Boosting

Bagging and Boosting are both ensemble techniques, where multiple models are built and then combined to get prediction.

While in Bagging the different models are independent and the end predictions are simple averaged.

On the other hand in case of Boosting the individual models are not independent, and the successive models put more weight on erroneous data points from previous models. The result at the end too is a weighted average of the individual model predictions.

1.3 OOB Error

While building Random forest, each individual tree model is built on data sampled randomly with replacement, or bootstrapped data. The data points which do not get sampled for this particular model are referred to as Out of Bag points, or OOB.

When we evaluate the indvidual model on these OOB points we land up with OOB error rate. Because the OOB points were not used to train the original model, OOB error is similar to the Test error rate.

We can thus see the drop OOB error with increasing number of trees, and choose the point where the drop in OOB error vanishes.

1.4 Optimal α_t

We know

Classifier at iteration t is :

$$f_t(x) = \sum_{i=1}^{t} \alpha_i h_i(x)$$

Exponential Loss for above classifier is:

$$L_t(f) = \frac{1}{m}\sum_{i=1}^{m}(\exp(-y_i * f_t(x_i))$$

$$L_t(f) = \frac{1}{m}\sum_{i=1}^{m}(\exp(-y_i * [f_{t-1}(x_i) + \alpha_t h_t(x_i)])$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left[(\exp(-y_i * f_{t-1}(x_i)) * \exp(-y_i * \alpha_t h_t(x_i))\right]$$

Defining weight for i'th point at iteration t as :

$$D_t(i) = \frac{\exp(-y_i * f_{t-1}(x_i))}{Z_t}$$

$$\implies L_t(f) = \frac{1}{m}\sum_{i=1}^{m}\left[(D_t(i) * Z_t * \exp(-y_i * \alpha_t h_t(x_i))\right]$$

Taking derivative wrt α

$$\frac{\partial L}{\partial \alpha_t} = \frac{-1}{m}\sum_{i=1}^{m}\left[(D_t(i) * Z_t * \exp(-y_i * \alpha_t h_t(x_i)) * y_i * h_t(x_i)\right] \tag{1}$$

$$= \frac{-1}{m}\sum_{y_i = h_i}(D_t(i) * \exp(-\alpha_t) + \frac{1}{m}\sum_{y_i \neq h_i}(D_t(i) * \exp(\alpha_t)$$

We also have:

$$\epsilon_t = \sum_{i=1}^{m} D_t(i)\mathcal{I}(y_i \neq h_t(x_i))$$

Therefore :

$$\implies \epsilon_t = \sum_{y_i \neq h_i} D_t(i)$$

$$\implies 1 - \epsilon_t = \sum_{y_i = h_i} D_t(i)$$

$$= \frac{(\epsilon_t - 1) * \exp(-\alpha_t)}{m} + \frac{(\epsilon_t) * \exp(\alpha_t)}{m}$$

$$\implies \frac{1-\epsilon_t}{\epsilon_t} = \exp(2\alpha_t)$$

$$\implies \alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$$

2 Question 2: Random forest for email spam classifier

2.1 CART model

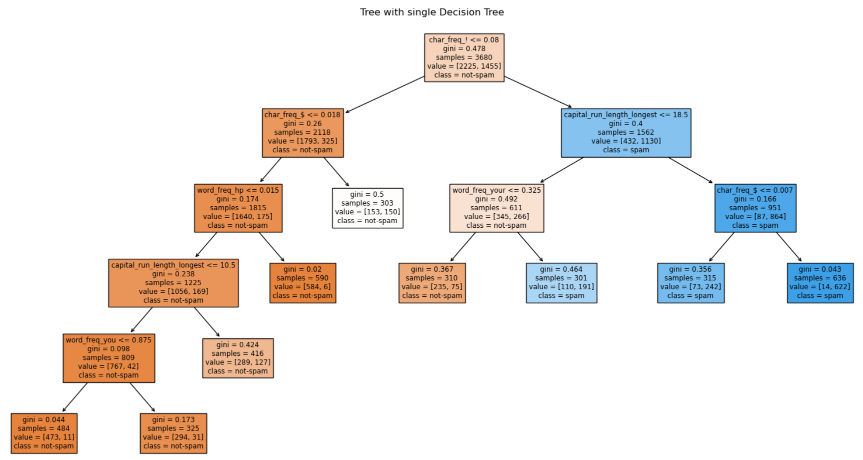
Tree with single Decision Tree
char_freq_! <= 0.08 gini = 0.478 samples = 3680 value = [2225, 1455] class = not-spam
char_freq_$ <= 0.018 gini = 0.26 samples = 2118 value = [1793, 325] class = not-spam
capital_run_length_longest <= 18.5 gini = 0.4 samples = 1562 value = [432, 1130] class = spam
word_freq_hp <= 0.015 gini = 0.174 samples = 1815 value = [1640, 175] class = not-spam
gini = 0.5 samples = 303 value = [153, 150] class = not-spam
word_freq_your <= 0.325 gini = 0.492 samples = 611 value = [345, 266] class = not-spam
char_freq_$ <= 0.007 gini = 0.166 samples = 951 value = [87, 864] class = spam
capital_run_length_longest <= 10.5 gini = 0.238 samples = 1225 value = [1056, 169] class = not-spam
gini = 0.02 samples = 590 value = [584, 6] class = not-spam
gini = 0.367 samples = 310 value = [235, 75] class = not-spam
gini = 0.464 samples = 301 value = [110, 191] class = spam
gini = 0.356 samples = 315 value = [73, 242] class = spam
gini = 0.043 samples = 636 value = [14, 622] class = spam
word_freq_you <= 0.875 gini = 0.098 samples = 809 value = [767, 42] class = not-spam
gini = 0.424 samples = 416 value = [289, 127] class = not-spam
gini = 0.044 samples = 484 value = [473, 11] class = not-spam
gini = 0.173 samples = 325 value = [294, 31] class = not-spam

Figure 1: Tree Model with single Decision tree

2.2 Random Forest

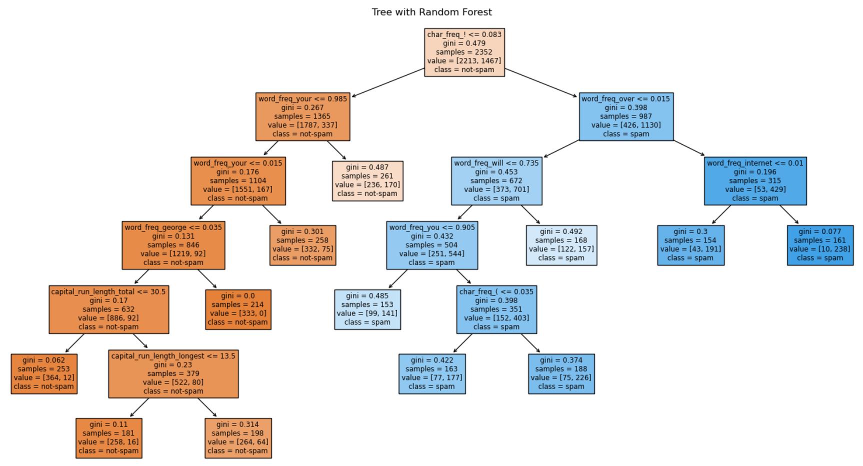
Tree with Random Forest
char_freq_! <= 0.083 gini = 0.479 samples = 2352 value = [2213, 1467] class = not-spam
word_freq_your <= 0.985 gini = 0.267 samples = 1365 value = [1787, 337] class = not-spam
word_freq_over <= 0.015 gini = 0.398 samples = 987 value = [426, 1130] class = spam
word_freq_your <= 0.015 gini = 0.176 samples = 1104 value = [1551, 167] class = not-spam
gini = 0.487 samples = 261 value = [236, 170] class = not-spam
word_freq_will <= 0.735 gini = 0.453 samples = 672 value = [373, 701] class = spam
word_freq_internet <= 0.01 gini = 0.196 samples = 315 value = [53, 429] class = spam
word_freq_george <= 0.035 gini = 0.131 samples = 846 value = [1219, 92] class = not-spam
gini = 0.301 samples = 258 value = [332, 75] class = not-spam
word_freq_you <= 0.905 gini = 0.432 samples = 504 value = [251, 544] class = spam
gini = 0.492 samples = 168 value = [122, 157] class = spam
gini = 0.3 samples = 154 value = [43, 191] class = spam
gini = 0.077 samples = 161 value = [10, 238] class = spam
capital_run_length_total <= 30.5 gini = 0.17 samples = 632 value = [886, 92] class = not-spam
gini = 0.0 samples = 214 value = [333, 0] class = not-spam
gini = 0.485 samples = 153 value = [99, 141] class = spam
char_freq_(<= 0.035 gini = 0.398 samples = 351 value = [152, 403] class = spam
gini = 0.062 samples = 253 value = [364, 12] class = not-spam
capital_run_length_longest <= 13.5 gini = 0.23 samples = 379 value = [522, 80] class = not-spam
gini = 0.422 samples = 163 value = [77, 177] class = spam
gini = 0.374 samples = 188 value = [75, 226] class = spam
gini = 0.11 samples = 181 value = [258, 16] class = not-spam
gini = 0.314 samples = 198 value = [264, 64] class = not-spam

Figure 2: Tree Model with Random Forest

A good rule of thumb value is:

- For Classification, $\nu \approx \sqrt{p}$
- For Regression, $\nu \approx \lfloor p \rfloor$

2.3 Random Forest

With 80% data for Train and 20% data for Test the accuracy results are as follows:

- Decision Tree, 80.8%
- Random Forest, 92.1%

The Accuracy is better with Random Forest, and both the values can be improved further with proper tuning of model hyper parameters.

The OOB error curve is given below:

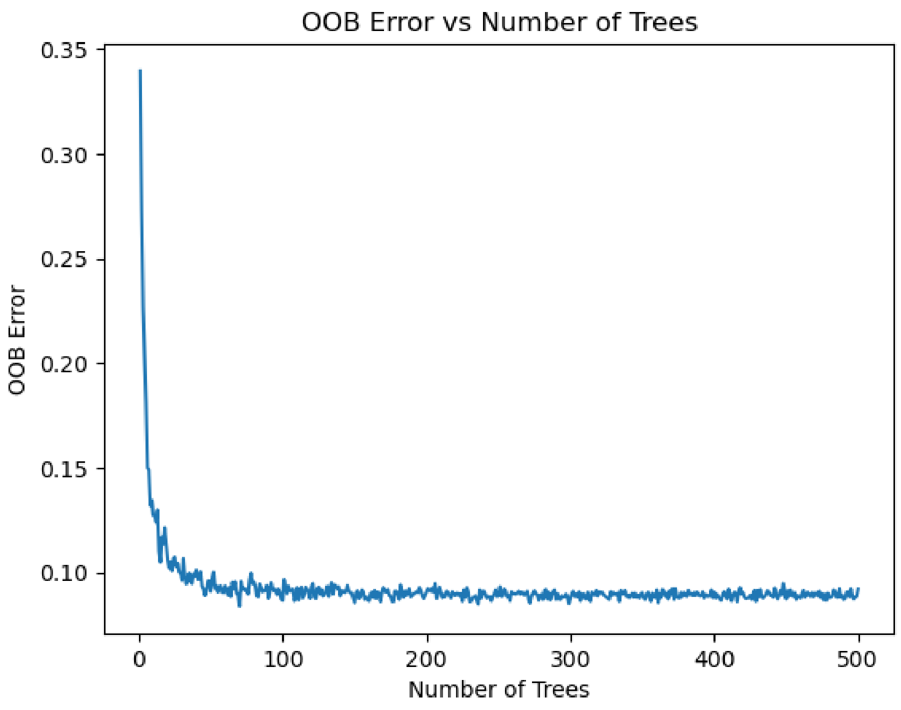

Figure 3: OOB error curve

3 Question 3: AdaBoost

3.1 Adaboost iterations fitting

I am using the following code to fit and calucate Adaboost params:

```
def get_eps(D,y,h):
    ind_yh = (y!=h).astype("int")
    return (np.sum(D*ind_yh))

def get_alpha(eps):
    return 0.5*np.log((1-eps)/(eps))
```

Iteration 1

```
h1 = np.array([1,1,-1,-1,-1,-1,-1,-1])
i = 1

get_base_fig()
plt.axvline(-0.25, color="black",lw = 4)
plt.title(f"iteration {i}")

w1 = np.array([1]*8)
w1 = w1/w1.sum()
eps1 = get_eps(w1,y,h1)
alpha1 = get_alpha(eps1)

print(f"$\epsilon_{i} = {eps1:.5f}$")
print(f"$D_{i}: {w1}$")
print(rf'$ \alpha_{i} = {alpha1:.5f}$')
```

$\epsilon_1 = 0.25000$
$D_1 : [0.1250.1250.1250.1250.1250.1250.1250.125]$
$\alpha_1 = 0.54931$

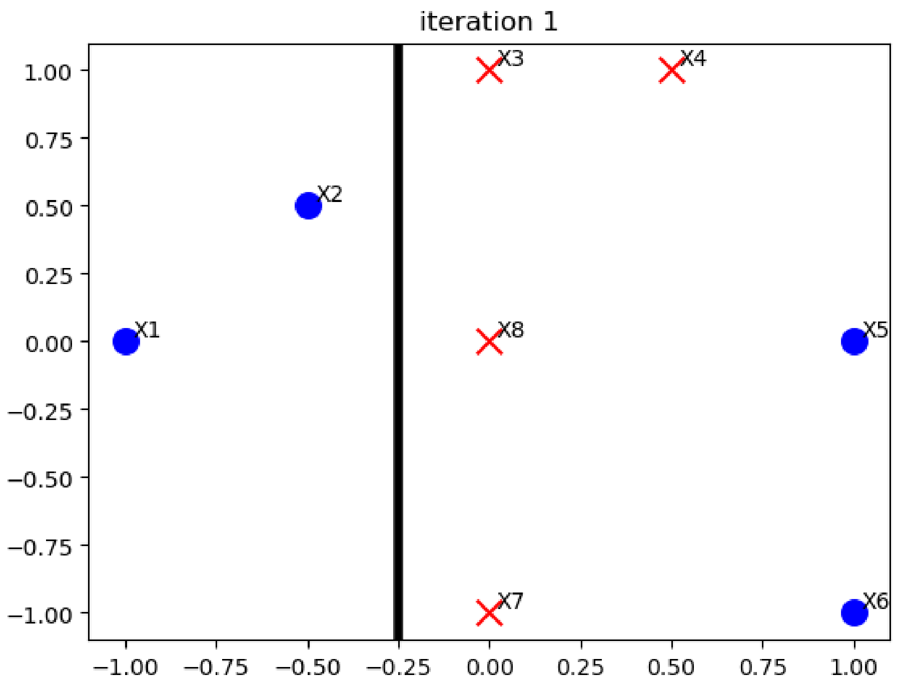

Figure 4: Decision Stump for iteration 1

Iteration 2

```
h2 = np.array([-1,-1,-1,-1,1,1,-1,-1])
i = 2

get_base_fig()
plt.axvline(0.75, color="black",lw = 4)
plt.title(f"iteration {i}")

w2 = w1*np.exp(-alpha1*h1*y)
z1 = w2.sum()
w2 = w2/z1
eps2 = get_eps(w2,y,h2)
alpha2 = get_alpha(eps2)

print(f"$\epsilon_{i} = {eps2:.5f}$")
print(f"$D_{i}: {w2}$")
print(f"$Z_{i-1}: {z1}$")
print(rf'$ \alpha_{i} = {alpha2:.5f}$')
```

$\epsilon_2 = 0.16667$
D_2 : [0.083333330.083333330.083333330.083333330.250.250.083333330.08333333]
Z_1 : 0.86603
$\alpha_2 = 0.80472$

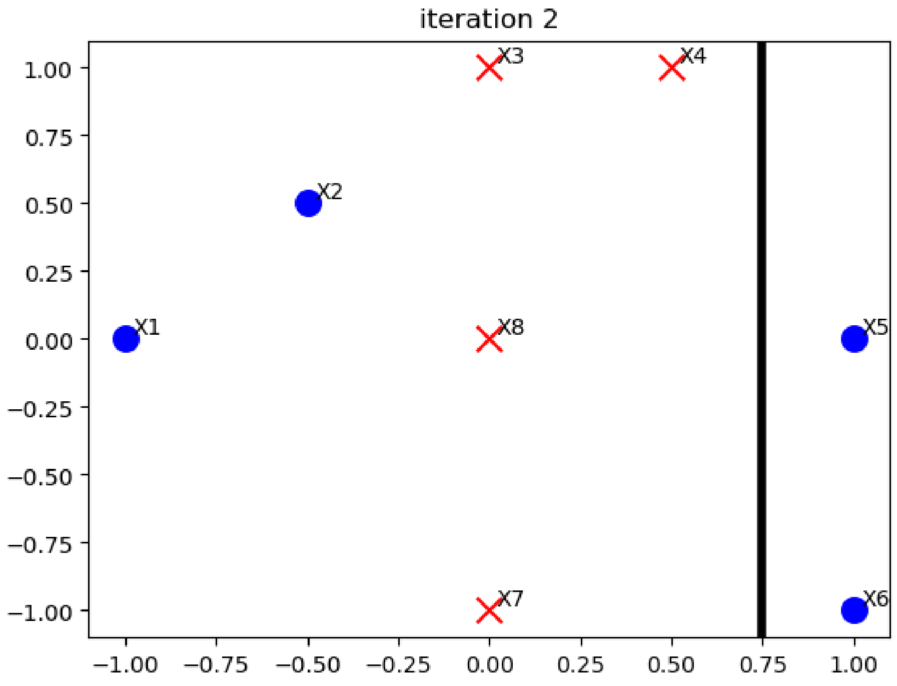

Figure 5: Decision Stump for iteration 2

Iteration 3

```
h3 = np.array([1,1,-1,-1,1,1,1,1])
i = 3

get_base_fig()
plt.axhline(0.75, color="black",lw = 4)
plt.title(f"iteration {i}")

w3 = w2*np.exp(-alpha2*h2*y)
z2=w3.sum()
w3 = w3/z2
eps3 = get_eps(w3,y,h3)
alpha3 = get_alpha(eps3)

print(f"$\epsilon_{i} = {eps3:.5f}$")
print(f"$D_{i}: {w3}$")
print(f"$Z_{i-1}: {z2:.5f}$")
print(rf'$ \alpha_{i} = {alpha3:.5f}$')
```

$\epsilon_3 = 0.10000$
D_3 : [0.250.250.050.050.150.150.050.05]
Z_2 : 0.74536
$\alpha_3 = 1.09861$

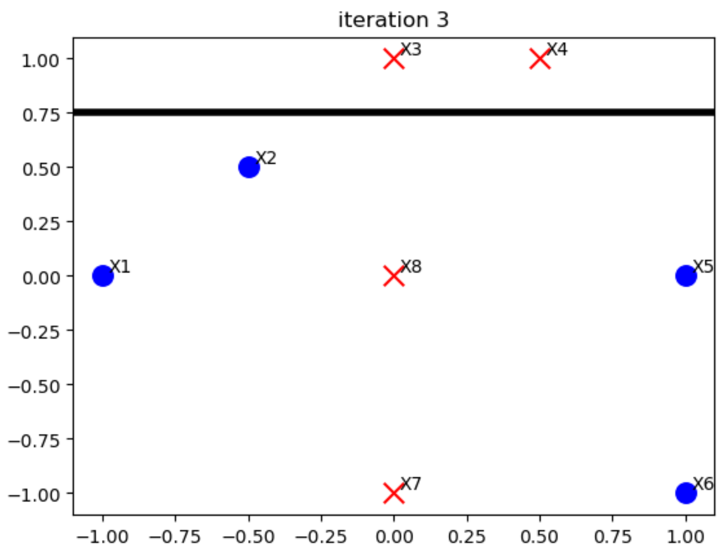

Figure 6: Decision Stump for iteration 3

Similarly $Z_3 = 0.59999$

i	ϵ_t	α_t	Z_t	$D_t(1)$	$D_t(2)$	$D_t(3)$	$D_t(4)$	$D_t(5)$	$D_t(6)$	$D_t(7)$	$D_t(8)$
1	0.25	0.54931	0.86603	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
2	0.16667	0.80472	0.74536	0.0833	0.0833	0.0833	0.0833	0.25	0.25	0.0833	0.0833
3	0.1	1.09861	0.6	0.25	0.25	0.05	0.05	0.15	0.15	0.05	0.05

Table 1: AdaBoost iteration results

3.2 Final classifier

The final classifier is $H_f = Sign(\sum_{t=1}^{3} \alpha_t * h_t)$

$$H_f = 0.54931 * h_1 + 0.80472 * h_2 + 1.09861 * h_3$$

with

$$h1 = [1, 1, -1, -1, -1, -1, -1, -1]$$

$$h2 = [-1, -1, -1, -1, 1, 1, -1, -1]$$

$$h3 = [1, 1, -1, -1, 1, 1, 1, 1]$$

Hence we get

$$H_f = [1., 1., -1., -1., 1., 1., -1., -1.]$$

Thus our classification accuracy is 100%.

Note: This accuracy is on train data.

Adaboost is a common ensemble learning method that combines numerous weak learners to generate a strong learner. Because Adaboost combines many weak learners, each of which might make mistakes on separate areas of the dataset it is able to increase overall accuracy by combining these learners.

The indvidual decision trees are also prone to be influenced by outliers, however the combination of multiple learners with different data points weighted differently is unlikely to do so.

4 Question 4: Importance Sampling

We have a RV X distributed as $\mathcal{N}(\mu_0, 1)$. We want the right tail probability $\alpha = \mathbb{P}\{X \geq z\}$.

For $z \gg \mu_0$, α is very small, and estimating this small probability accurately is not easy. Now to improve accuracy, we samples from another Gaussian random variable Y with mean $\mu_1 = z$ and variance equal to 1.

4.1 Importance Ratio

Importance Ratio is given by the ratio of likelihood of saying values above z under the original distribution $\mathcal{N}(\mu_0, 1)$ vs the new distribution $\mathcal{N}(z, 1)$.

$$\begin{aligned}
&\text{We have } X \sim \mathcal{N}(\mu_0, 1), Y \sim \mathcal{N}(z, 1) \\
\implies importance_{ratio} &= \frac{P(X >= z)}{P(Y >= z)} \\
&= \frac{1 - P(X <= z)}{1 - P(Y <= z)} \\
\implies importance_{ratio} &= \frac{1 - F_x(z)}{1 - F_y(z)} \\
&\text{where } F_x, F_y \text{ are CDF for } X, Y \text{ respectively}
\end{aligned} \tag{2}$$

4.2 Importance Sampling Algorithm

The algorithm is given below:

Algorithm 1 Importance Sampling

```
Require: $\mu_0, z, N$
  Initialise Y-list=[], importance-ratio-list = []
  for i = 1 to N do
      $y_i \leftarrow \mathcal{N}(z, 1)$    (Sample from Y)
      importance$_{ratio}(i) \leftarrow \frac{1-F_x(z)}{1-F_y(z)}$    (calculate importance ratio)
        Append $y_i$ and importance ratio to lists initialised above
    end for
  I $\leftarrow \frac{1}{N}\sum_i N\mathrm{I}(y_i >= z) * importance_{ratio}(i)$
```

The python implementation is as follows:

```
def importance_sampling(mu_x,z,N=100):
    x_f = norm(loc=mu_x,scale = 1)
    y_g = norm(loc=z,scale = 1)
    # draw monte carlo samples
    x_samples = x_f.rvs(size = N)
    y_samples = y_g.rvs(size = N)
    #get importance ratio
    pr_f_x = x_f.sf(x=y_samples) #Pr(x>=z) from original
    pr_g_y = y_g.sf(x=y_samples) #Pr(x>=z) from sampling
    importance_rt = pr_f_x/pr_g_y
    #approxaimte I1
    indicataor_x = (x_samples>=z).astype("int")
    I1 = np.mean(indicataor_x)
    #approxaimte I2
    indicataor_y = (y_samples>=z).astype("int")
    I2 = np.mean(indicataor_y*importance_rt)
    return I1,I2
```

4.3 Numerical Example

For $\mu_0 = 1$ and $z = 10.0$, we get the following results:

```
True value 1.1285884059538324e-19
For Monte Carlo, mean = 0.0, std = 0.0
For Importance Sampling, mean = 1.0772267698651663e-20, std =
    3.469453274231522e-21
```

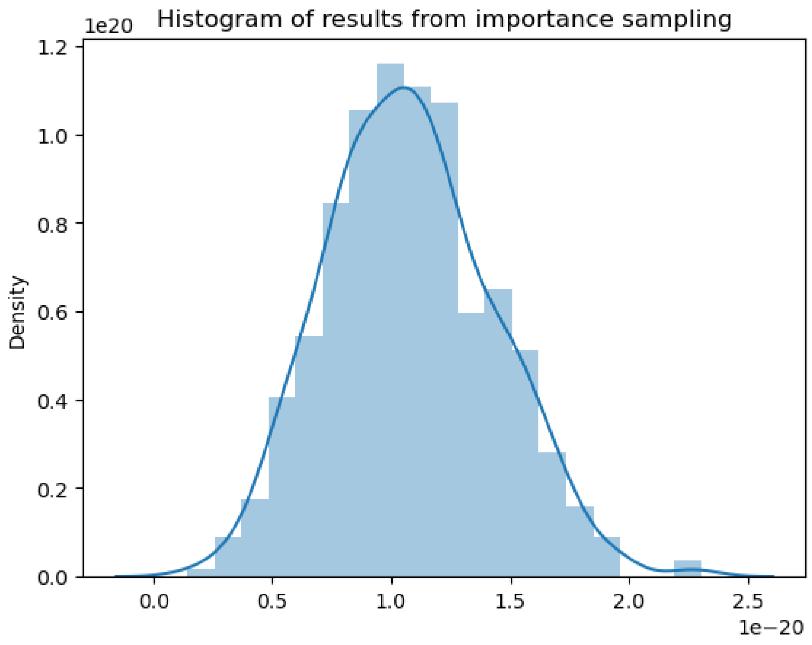

Figure 7: Distribution of results from importance sampling

5 References:

1. **Collaborators**: Yibei, Dipendra