EE660_HW6_Hardik_2678294168

November 6, 2021

0.1 HW6-EE660: Hardik Prajapati (2678294168)

[1]:
```
from matplotlib import pyplot as plt
import numpy as np
```

[57]:
```
def E_ab(N_t,N_s,alpha,print_flag):
    d_vc=10
    tol=0.1
    diff_measure=0.1

    #given total counts of labeled target and source domain data points, N_t &␣
↪N_s respectivel, we can calulate Beta
    total_labeled=N_t+N_s
    beta=N_t/total_labeled

    #assuming number of unlabeled data points in each domain equal to labeled␣
↪points in respective domain
    #total_unlabeled=N_t+N_s

    term_1=2*(1-alpha)*(0.5*diff_measure)
    term_2_i=4*np.sqrt(((alpha*alpha)/beta)+(((1-alpha)*(1-alpha))/(1-beta)))
    term_2_ii=np.sqrt((2*d_vc*np.log(2*(total_labeled+1))/total_labeled)+(2*np.
↪log(8/tol)/total_labeled))
    term_2=term_2_i*term_2_ii
    #term_3=8*(1-alpha)*np.sqrt((2*d_vc*np.log(total_unlabeled)/
↪total_unlabeled)+(np.log(8/tol)/total_unlabeled))
    e_ab=term_1+term_2
    if print_flag==True:
        print("E_ab for N_t ({}), N_s ({}), alpha ({}) = {:.2f}".
↪format(N_t,N_s,alpha,e_ab))
    return e_ab
```

[58]:
```
#Q2_a_i
n_t=1
n_s=100
alphas=[0.1,0.5,0.9]
for a in alphas:
```

```
    E_ab(n_t,n_s,a,True)
```

```
E_ab for N_t (1), N_s (100), alpha (0.1) = 5.86
E_ab for N_t (1), N_s (100), alpha (0.5) = 21.62
E_ab for N_t (1), N_s (100), alpha (0.9) = 38.64
```

[59]:
```
#Q2_a_ii
n_t=10
n_s=1000
alphas=[0.1,0.5,0.9]
for a in alphas:
    E_ab(n_t,n_s,a,True)
```

```
E_ab for N_t (10), N_s (1000), alpha (0.1) = 2.25
E_ab for N_t (10), N_s (1000), alpha (0.5) = 8.12
E_ab for N_t (10), N_s (1000), alpha (0.9) = 14.46
```

[60]:
```
#Q2_a_iii
n_t=100
n_s=10000
alphas=[0.1,0.5,0.9]
for a in alphas:
    E_ab(n_t,n_s,a,True)
```

```
E_ab for N_t (100), N_s (10000), alpha (0.1) = 0.86
E_ab for N_t (100), N_s (10000), alpha (0.5) = 2.94
E_ab for N_t (100), N_s (10000), alpha (0.9) = 5.19
```

[61]:
```
#Q2_a_iv
n_t=1000
n_s=100000
alphas=[0.1,0.5,0.9]
for a in alphas:
    E_ab(n_t,n_s,a,True)
```

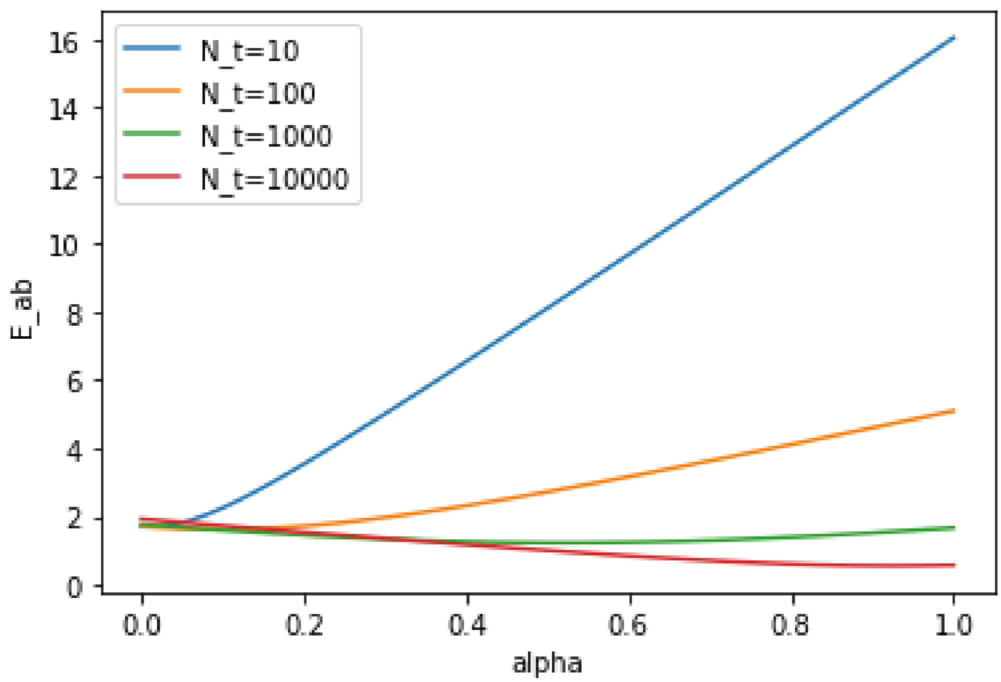
```
E_ab for N_t (1000), N_s (100000), alpha (0.1) = 0.36
E_ab for N_t (1000), N_s (100000), alpha (0.5) = 1.06
E_ab for N_t (1000), N_s (100000), alpha (0.9) = 1.82
```

0.1.1 Q2__a__v: Only one of the sets of numbers above assure degree of generalization (i.e e__ab<0.5). Case: N__t = 1000, N__s = 100000, alpha=0.1

[62]:
```
#Q2_b
n_t=[10,100,1000,10000]
n_s=1000
alphas=np.linspace(0,1)
for n in n_t:
    e_ab_n=[]
    for a in alphas:
```

```
        val=E_ab(n,n_s,a,False)
        e_ab_n.append(val)
    plt.plot(alphas,e_ab_n)
    plt.legend(["N_t=10","N_t=100","N_t=1000","N_t=10000"])
    plt.xlabel("alpha")
    plt.ylabel("E_ab")
    idx=np.argmin(e_ab_n)
    print("Optimal value of alpha for N_t ({}) = {:.4f}".format(n,alphas[idx]))
```

```
Optimal value of alpha for N_t (10) = 0.0204
Optimal value of alpha for N_t (100) = 0.1020
Optimal value of alpha for N_t (1000) = 0.5306
Optimal value of alpha for N_t (10000) = 0.9184
```



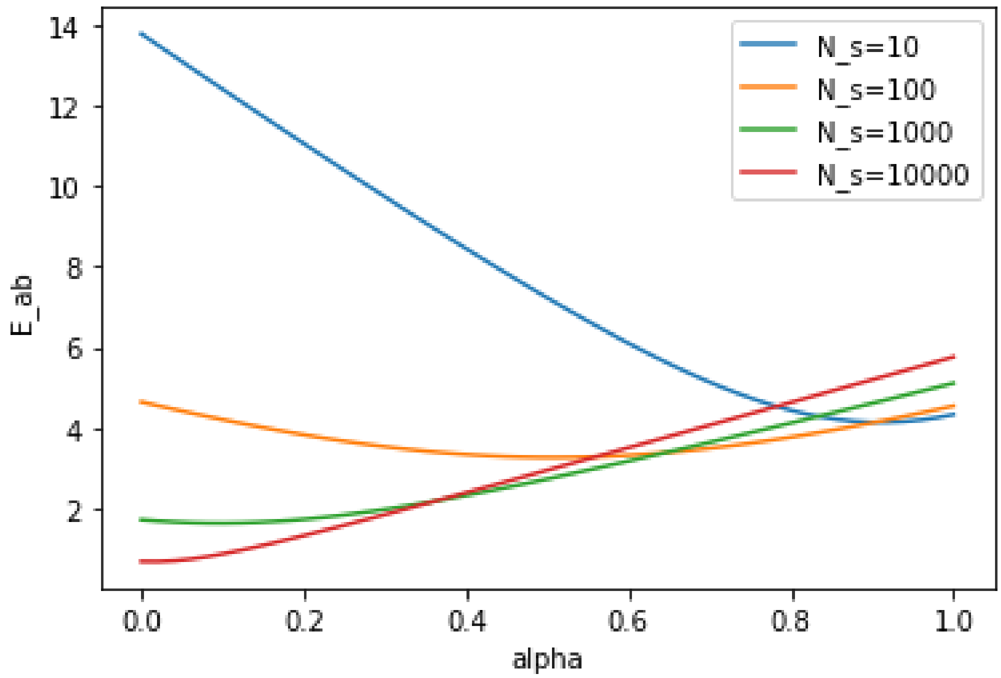
0.2 **We know that alpha is importance of error in target domain (with small # data points) compared with error in source domain (with large # data points). Hence, with greater number of labeled data points in target domain, importance of error in target domain increases with constant number of labeled points in source domain since our goal is to adapt the target domain and generalize better in target domain.**

0.3 **It can be observed that for a specific value of N__t, once the optimum value of alpha is reached, generalization error bound increases.**

0.4 **Also, as N__t increases, optimum value of alpha increases.**

[63]:
```
#Q2_c
n_t=100
n_s=[10,100,1000,10000]
alphas=np.linspace(0,1)
for n in n_s:
    e_ab_n=[]
    for a in alphas:
        val=E_ab(n_t,n,a,False)
        e_ab_n.append(val)
    plt.plot(alphas,e_ab_n)
    plt.legend(["N_s=10","N_s=100","N_s=1000","N_s=10000"])
    plt.xlabel("alpha")
    plt.ylabel("E_ab")
    idx=np.argmin(e_ab_n)
    print("Optimal value of alpha for N_s ({}) = {:.4f}".format(n,alphas[idx]))
```

```
Optimal value of alpha for N_s (10) = 0.9184
Optimal value of alpha for N_s (100) = 0.5102
Optimal value of alpha for N_s (1000) = 0.1020
Optimal value of alpha for N_s (10000) = 0.0204
```



0.5 We know that alpha is importance of error in target domain (with small # data points) compared with error in source domain (with large # data points). Hence, with lower number of labeled data points in source domain, importance of error in target domain increases since our goal is to adapt to the target domain and generalize better in target domain. With less number of labeled data points in source domain, error in source domain won't help us to provide any useful information to generalize better in target domain.

0.6 It can be observed that for a specific value of N__s (lower #), e__ab decreases as alpha increases till the optimum value is reached. For N__s (higher #), e__ab increases as alpha increase once the optimum value is reached and hence tends to have a very low alpha.

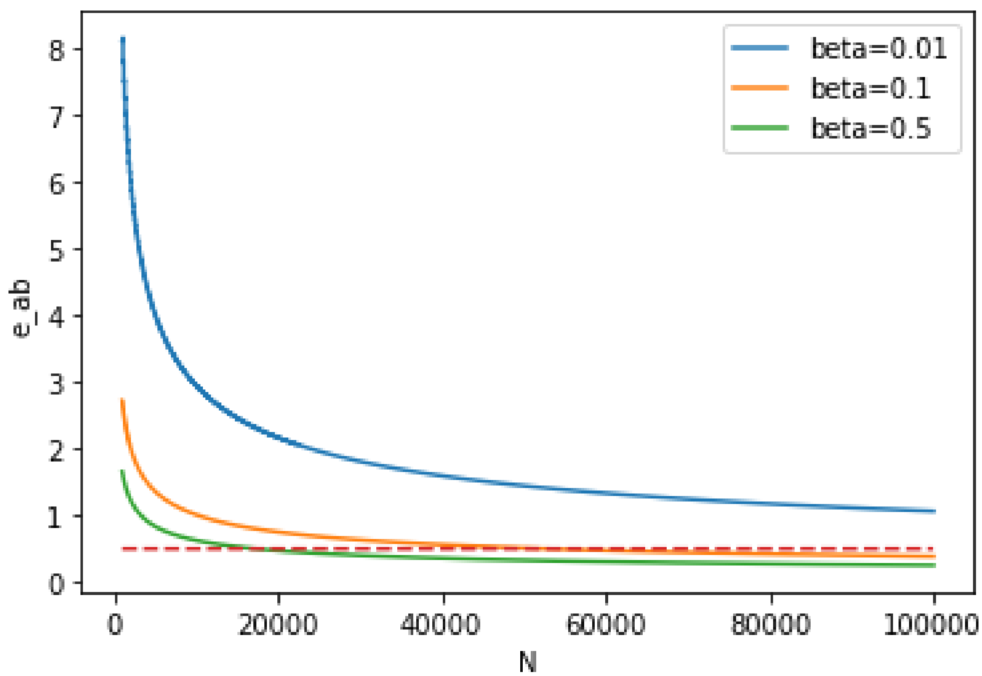
0.7 Also, as N__s increases, optimum value of alpha decreases, hence giving higher importance to error in source domain.

———Xxxx—xxxxx—–xxxxxx——xxxxxx—— ## Q2_d_i ### According to the observation above, alpha=beta seems to be a good default choice for minimizing the cross-domain error-Bound. ### This choice seems to be reasonably quite consistent with all the sets, for minimizing the cros-domain error bound.

——xxxx——-xxxxx———xxxxx—– ## Q2_d_ii ### (solved and submitted in pen-paper type questions)

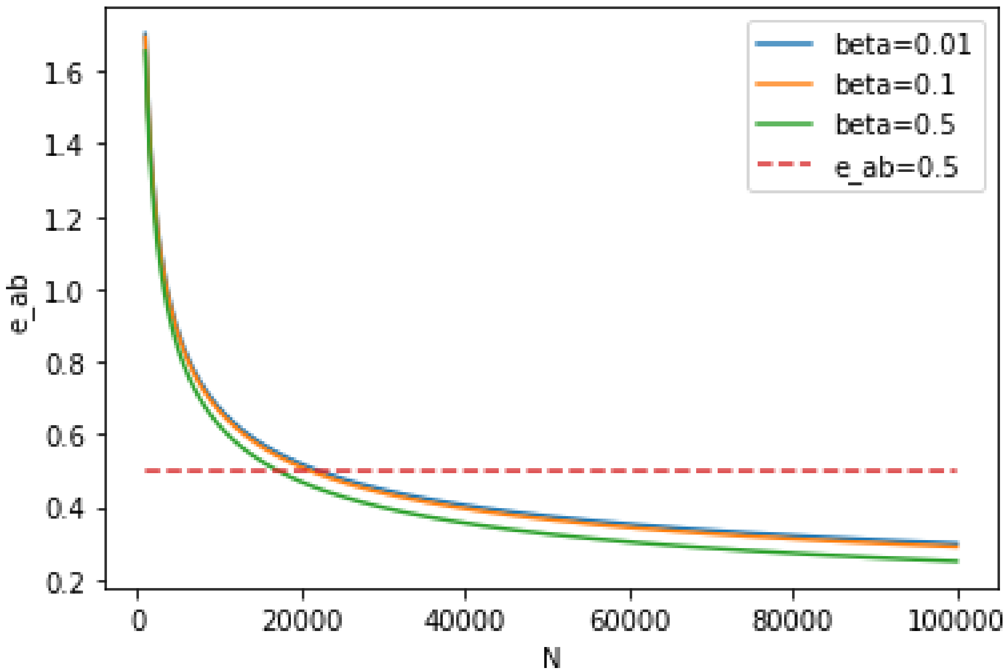
[64]:
```
#Q2_d_iii (alpha =0.5)
alpha_=0.5
```

```
N=list(range(1000,100001))
beta=[0.01,0.1,0.5]
for b in beta:
    e_ab_beta=[]
    for n in N:
        N_t=int(b*n)
        N_s=n-N_t
        val=E_ab(N_t,N_s,alpha_,False)
        e_ab_beta.append(val)
    plt.plot(N,e_ab_beta)
plt.plot(N,[0.5]*len(N),'--')
plt.xlabel("N")
plt.ylabel("e_ab")
plt.legend(["beta=0.01","beta=0.1","beta=0.5"])
plt.show()
```



[65]:
```
#Q2_d_iii (alpha =beta)
N=list(range(1000,100001))
beta=[0.01,0.1,0.5]
for b in beta:
    e_ab_beta=[]
    for n in N:
        N_t=int(b*n)
        N_s=n-N_t
```

```
        val=E_ab(N_t,N_s,b,False)
        e_ab_beta.append(val)
    plt.plot(N,e_ab_beta)
plt.xlabel("N")
plt.ylabel("e_ab")
plt.plot(N,[0.5]*len(N),'--')
plt.legend(["beta=0.01","beta=0.1","beta=0.5","e_ab=0.5"])
plt.show()
```



0.7.1 On comparing the above plots, we can say that Alpha=beta is a better choice in terms of minimizing the cross-domain error even for beta<1/2. From part (ii), algebrically it was shown that, for beta>1/2, E_ab(alpha=beta) is always less than E_ab(alpha=0.5)

0.7.2 Hence, in general we can conclude that, alpha=beta is a good default choice when minimizing cross-domain generalization error bounds

```
[ ]:
```