TopN model + mask generation:

Purpose:

1. validate that baseline non-linear model has very good topN accuracy

2. create masking algorithm for distinguishing specificly top 2 classes

a. one mask for each pair - O(nClasses^2) masks

b. one mask for each class - nClasses masks

In [1]:
```
%load_ext autoreload
%autoreload 2
```

In [27]:
```
import sklearn
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.decomposition import PCA
import time
import sys
import math
import numpy as np
import random
import joblib
from scipy import stats
from tqdm import tqdm_notebook
import matplotlib.pyplot as plt
import copy

import Config
from Config import *
import Dataloader as DL
import HD_basis as HDB
import HD_encoder as HDE
import HD_classifier as HDC
```

In [3]:
```
# train data. with retrain
def train(hdc, traindata, trainlabels, testdata, testlabels, param = Config.conf
    train_accs = []
    test_accs = []
    for _ in tqdm_notebook(range(param["epochs"]), desc='epochs'):
        train_accs.append(hdc.fit(traindata, trainlabels, param))
        test_accs.append(hdc.test(testdata, testlabels, param["kernel"]))
        if len(train_accs) % 5 == 1:
            print("Train: %f \t \t Test: %f"%(train_accs[-1], test_accs[-1]))
        if train_accs[-1] == 1:
            print("Train: %f \t \t Test: %f"%(train_accs[-1], test_accs[-1]))
            break
    return np.asarray(train_accs), np.asarray(test_accs)
```

In [4]:
```
# train data. with retrain
def train_masks(hdc, traindata, trainlabels, testdata, testlabels, param = Confi
    train_accs = []
    test_accs = []

    for _ in range(5):
        train_accs.append(hdc.fit(traindata, trainlabels, param))
        test_accs.append(hdc.test(testdata, testlabels, param["kernel"]))
        if len(train_accs) % 5 == 1:
            print("Train: %f \t \t Test: %f"%(train_accs[-1], test_accs[-1]))
        if train_accs[-1] == 1:
            print("Train: %f \t \t Test: %f"%(train_accs[-1], test_accs[-1]))
            break


    print("Train memasks")
    for _ in tqdm_notebook(range(param["epochs"]), desc='epochs'):
        train_accs.append(hdc.fit_mask(traindata, trainlabels, param))
        test_accs.append(hdc.test(testdata, testlabels, param["kernel"]))
        if len(train_accs) % 5 == 1:
            print("Train: %f \t \t Test: %f"%(train_accs[-1], test_accs[-1]))
        if train_accs[-1] == 1:
            print("Train: %f \t \t Test: %f"%(train_accs[-1], test_accs[-1]))
            break
    return np.asarray(train_accs), np.asarray(test_accs)
```

In [5]:
```
def show_plots(fst_scs, snd_scs, pairs = None):

    new_fst = []
    new_snd = []

    if pairs is not None:
        for sc in fst_scs:
            if (sc[3], sc[4]) in pairs or (int(sc[4]), int(sc[3])) in pairs:
                new_fst.append(sc)
        for sc in snd_scs:
            if (sc[3], sc[4]) in pairs or (int(sc[4]), int(sc[3])) in pairs:
                new_snd.append(sc)
        fst_scs = np.asarray(new_fst)
        snd_scs = np.asarray(new_snd)

    rg = [min(np.min(fst_scs[:,:2]), np.min(snd_scs[:,:2])),
          max(np.max(fst_scs[:,:2]), np.max(snd_scs[:,:2]))]

    # Scatter plots
    plt.figure(figsize = (10, 10))
    plt.plot(rg, rg, ls = "--")
    plt.scatter(fst_scs[:,0], fst_scs[:,1], label = "fst", color = "g", alpha = (
    plt.legend()
    plt.show()

    plt.figure(figsize = (10, 10))
    plt.plot(rg, rg, ls = "--")
    plt.scatter(snd_scs[:,0], snd_scs[:,1], label = "snd", color = "r", alpha = (
    plt.legend()
    plt.show()

    plt.figure(figsize = (10, 10))
    plt.plot(rg, rg, ls = "--")
    plt.scatter(fst_scs[:,0], fst_scs[:,1], label = "fst", color = "g", alpha = (
    plt.scatter(snd_scs[:,0], snd_scs[:,1], label = "snd", color = "r", alpha = (
    plt.legend()
    plt.show()

    # ratio dist
    fst_rt = fst_scs[:,0]/fst_scs[:,1]
    snd_rt = snd_scs[:,0]/snd_scs[:,1]

    # Exclude but record outliers
    sd = np.std(np.append(fst_rt, snd_rt))
    mn = np.mean(np.append(fst_rt, snd_rt))
    z = 2
    print(sd, mn)
    fst_rt_ol = fst_rt[np.abs(fst_rt - mn) > 2 * z * sd]
    fst_rt_cl = fst_rt[np.abs(fst_rt - mn) <= 2 * z * sd]
    snd_rt_ol = snd_rt[np.abs(snd_rt - mn) > 2 * z * sd]
    snd_rt_cl = snd_rt[np.abs(snd_rt - mn) <= 2 * z * sd]

    plt.figure(figsize = (50, 20))
    plt.hist(fst_rt_cl, bins = 200, label = "fst", color = "r", alpha = 0.5)
    plt.hist(snd_rt_cl, bins = 200, label = "snd", color = "g", alpha = 0.5)
    plt.legend(prop={"size":80})
```

```
    # diff dist
    fst_sc = fst_scs[:,0]-fst_scs[:,1]
    snd_sc = snd_scs[:,0]-snd_scs[:,1]
    plt.figure(figsize = (50, 20))
    plt.hist(fst_sc, bins = 200, label = "fst", color = "r", alpha = 0.5)
    plt.hist(snd_sc, bins = 200, label = "snd", color = "g", alpha = 0.5)
    plt.legend(prop={"size":80})
```

In [6]:
```
dl = DL.Dataloader()
nFeatures, nClasses, traindata, trainlabels, testdata, testlabels = dl.getParam(
```

```
Loading dataset UCIHAR from UCIHAR
Loading train data... train data of shape (6213, 561) loaded
Loading test data...  test  data of shape (1554, 561) loaded
Data Loaded. Num of features = 561 Num of Classes = 12
```

In [7]:
```
#Data shuffling
shuf_train = np.random.permutation(len(traindata))
traindata = traindata[shuf_train]
trainlabels = trainlabels[shuf_train]

shuf_test = np.random.permutation(len(testdata))
testdata = testdata[shuf_test]
testlabels = testlabels[shuf_test]
```

In [8]:
```
traindata = traindata[:10000]
testdata = testdata[:5000]
```

In [9]:
```
param = Config.config
param["nFeatures"] = nFeatures
param["nClasses"] = nClasses
print(param)
```

```
{'data_location': '../dataset/', 'directory': 'UCIHAR', 'dataset': 'UCIHAR',
'D': 200, 'vector': 'Gaussian', 'mu': 0, 'sigma': 1, 'binarize': 0, 'lr': 1, 's
parse': 0, 's': 0.1, 'binaryModel': 0, 'checkpoints': False, 'kernel': <Kernel_
T.DOT: 0>, 'width': None, 'height': None, 'nLayers': 5, 'uniform_dim': 1, 'unif
orm_ker': 1, 'dArr': None, 'k': 3, 'kArr': None, 'one_shot': 0, 'data_percentag
es': [1.0, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5], 'train_percent': 1, 'dropout': 0,
'drop_percentages': [0, 0.1, 0.2, 0.5], 'dropout_rate': 0, 'update_type': <Upda
te_T.FULL: 1>, 'iter_per_trial': 3, 'iter_per_encoding': 5, 'epochs': 80, 'nFea
tures': 561, 'nClasses': 12}
```

Sample code to train a model

In [15]:
```
################## VANILLA #################
hdb = HDB.HD_basis(Config.Generator_T.Vanilla, param)
basis = hdb.getBasis()
bid = hdb.getParam()["id"]
# Update param with bid
param = hdb.getParam()
print(bid)
```

```
Generating vanilla HD basis of shape...
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
9238

(200, 561)
Encoding time: 0.02689337730407715
```

In [16]:
```
# Retrieve info upto basis generator, given correct bid
#bid = 6679
#basis, param = HDB.loadBasis("base_%d.pkl"%bid)
```

In [17]:
```
hde = HDE.HD_encoder(basis)

trainencoded = hde.encodeData(traindata)
#HDE.saveEncoded(trainencoded, trainlabels, bid, "train")

testencoded = hde.encodeData(testdata)
#HDE.saveEncoded(testencoded, testlabels, bid, "test")
```

```
Encoding data of shape (6213, 561)
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
Time spent: 0 sec
Encoding data of shape (1554, 561)
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
Time spent: 0 sec
```

In [18]:
```
# Retrieve info upto encoder, given correct bid
#bid = 6679
#basis, param = HDB.loadBasis("base_%d.pkl"%bid)
#trainencoded, trainlabels = HDE.loadEncoded("encoded_%d_train.pkl"%bid)
#testencoded, testlabels = HDE.loadEncoded("encoded_%d_test.pkl"%bid)
```

In [19]:
```
hdc = HDC.HD_classifier(param["D"], param["nClasses"], bid)
train_acc, test_acc = train(hdc, trainencoded, trainlabels, testencoded, testlab
```

```
c:\program files (x86)\microsoft visual studio\shared\python37_64\lib\site-pack
ages\ipykernel_launcher.py:5: TqdmDeprecationWarning: This function will be rem
oved in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  """
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
Fitting with configuration: [('one_shot', 0), ('dropout', 0), ('lr', 1), ('kern
el', <Kernel_T.COS: 1>)]

Train: 0.721552                 Test: 0.428571
Train: 0.878320                 Test: 0.822394
Train: 0.908418                 Test: 0.922780
Train: 0.920650                 Test: 0.945946
Train: 0.935458                 Test: 0.938867
Train: 0.940286                 Test: 0.920849
Train: 0.950427                 Test: 0.923423
Train: 0.954772                 Test: 0.940798
Train: 0.951392                 Test: 0.948520
Train: 0.956704                 Test: 0.953668
Train: 0.959440                 Test: 0.951737
Train: 0.961532                 Test: 0.922136
Train: 0.961532                 Test: 0.941441
Train: 0.961532                 Test: 0.958172
Train: 0.958796                 Test: 0.957529
Train: 0.968453                 Test: 0.957529
```

Top n accuracy

In [11]:
```
def topn_suit(traindata, trainlabels, testdata, testlabels, param):
    hdb = HDB.HD_basis(Config.Generator_T.Vanilla, param)
    basis = hdb.getBasis()
    bid = hdb.getParam()["id"]
    param = hdb.getParam()

    hde = HDE.HD_encoder(basis)
    trainencoded = hde.encodeData(traindata)
    testencoded = hde.encodeData(testdata)

    hdc = HDC.HD_classifier(param["D"], param["nClasses"], bid)
    train_acc, test_acc = train(hdc, trainencoded, trainlabels, testencoded, tes
    accs = []
    cmps = []
    for k in range(5):
        accs.append(hdc.test_topn(testencoded, testlabels, k+1))
        cmps.append(hdc.predict_topn(testencoded, k+1))
        print("Top %d accuracy: "%(k+1), accs[-1])
    return accs, cmps
```

In [ ]:
```
Ds = [200, 500, 1000]
accs_D = dict()
cmps_D = dict()
for D in Ds:
    param["D"] = D
    accs_D[D], cmps_D[D] = topn_suit(traindata, trainlabels, testdata, testlabels
param["D"] = Config.config["D"]
```

```
Generating vanilla HD basis of shape...
```

vectors: 100% 200/200 [00:00<00:00, 6266.56it/s]

```
(200, 561)
Encoding time: 0.03194141387939453
Encoding data of shape (6213, 561)
```

samples encoded: 100% 6213/6213 [00:00<00:00, 11794.78it/s]

```
Time spent: 0 sec
Encoding data of shape (1554, 561)
```

samples encoded: 100% 1554/1554 [00:00<00:00, 10717.75it/s]

```
Time spent: 0 sec
c:\program files (x86)\microsoft visual studio\shared\python37_64\lib\site-pack
ages\ipykernel_launcher.py:5: TqdmDeprecationWarning: This function will be rem
oved in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  """
```

epochs: 91% 73/80 [00:06<00:00, 10.32it/s]

```
Fitting with configuration: [('one_shot', 0), ('dropout', 0), ('lr', 1), ('kern
el', <Kernel_T.DOT: 0>)]

Train: 0.712699                 Test: 0.734878
Train: 0.890713                 Test: 0.803732
Train: 0.900853                 Test: 0.895753
Train: 0.917592                 Test: 0.929215
```

In [ ]:
```
for D in accs_D.keys():
    print(D)
    for i in range(len(accs_D[D])):
        print("Top %d Accuracy:"%(i+1), accs_D[D][i])
```

Mask generation: 2d masks

In [10]:
```
hdb = HDB.HD_basis(Config.Generator_T.Vanilla, param)
basis = hdb.getBasis()
bid = hdb.getParam()["id"]
param = hdb.getParam()

hde = HDE.HD_encoder(basis)
trainencoded = hde.encodeData(traindata)
testencoded = hde.encodeData(testdata)

hdc = HDC.HD_classifier(param["D"], param["nClasses"], bid)
#train_acc, test_acc = train_masks(hdc, trainencoded, trainlabels, testencoded, t
train_acc, test_acc = train(hdc, trainencoded, trainlabels, testencoded, testlabe
hdc.normalizeClasses()
acc_1 = hdc.test(testencoded, testlabels, param["kernel"])
acc_2 = hdc.test_topn(testencoded, testlabels, 2)
print("Normalized top1 and top 2 test accuracy:", acc_1, acc_2)
```

```
Generating vanilla HD basis of shape...
```

vectors: 100% 200/200 [00:08<00:00, 23.55it/s]

```
(200, 561)
Encoding time: 0.0378727912902832
Encoding data of shape (6213, 561)
```

samples encoded: 100% 6213/6213 [00:02<00:00, 3080.18it/s]

```
Time spent: 0 sec
Encoding data of shape (1554, 561)
```

samples encoded: 100% 1554/1554 [00:01<00:00, 1057.73it/s]

```
Time spent: 0 sec
c:\program files (x86)\microsoft visual studio\shared\python37_64\lib\site-pack
ages\ipykernel_launcher.py:5: TqdmDeprecationWarning: This function will be rem
oved in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  """
```

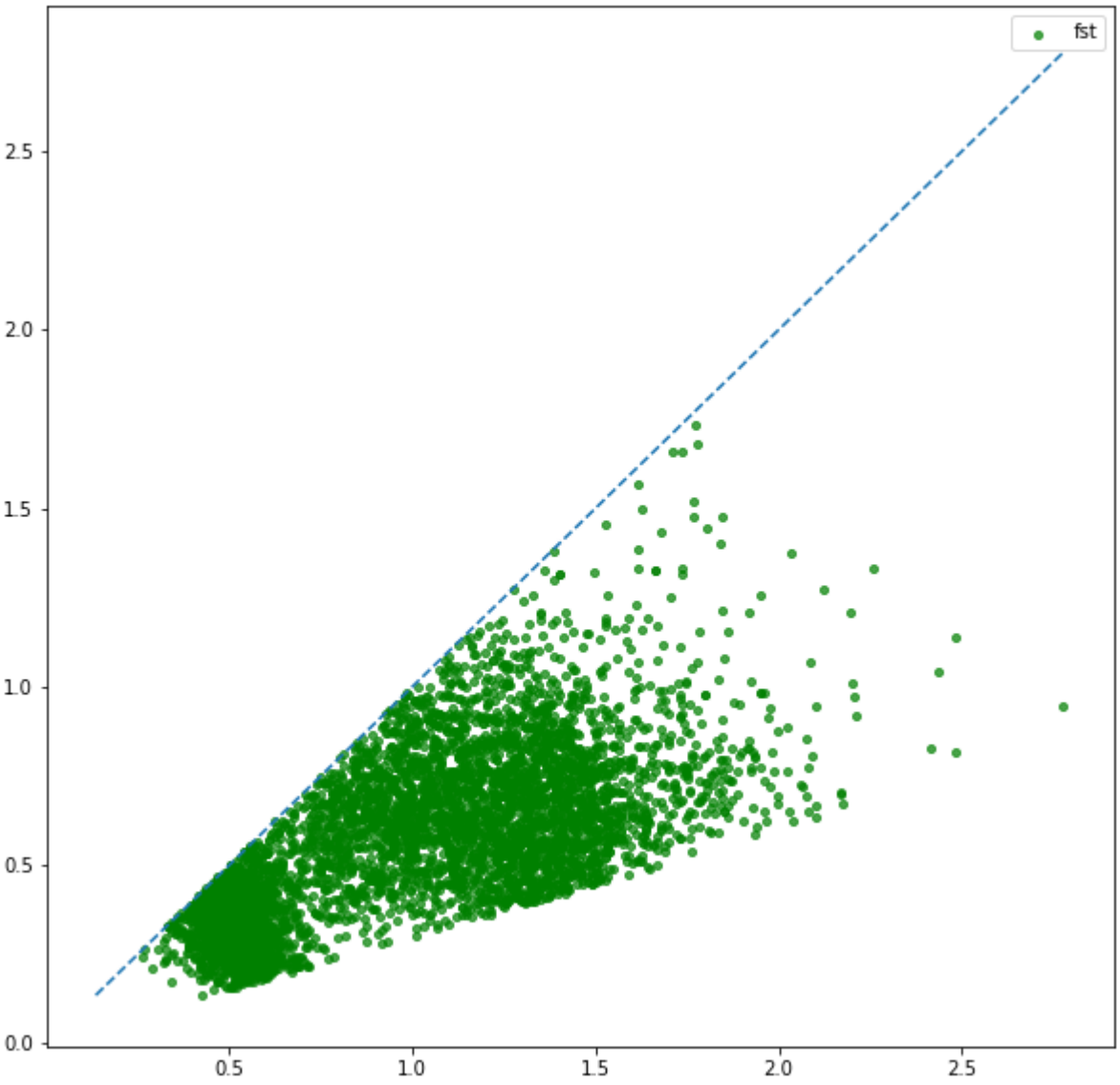
epochs: 100% 80/80 [00:07<00:00, 10.33it/s]

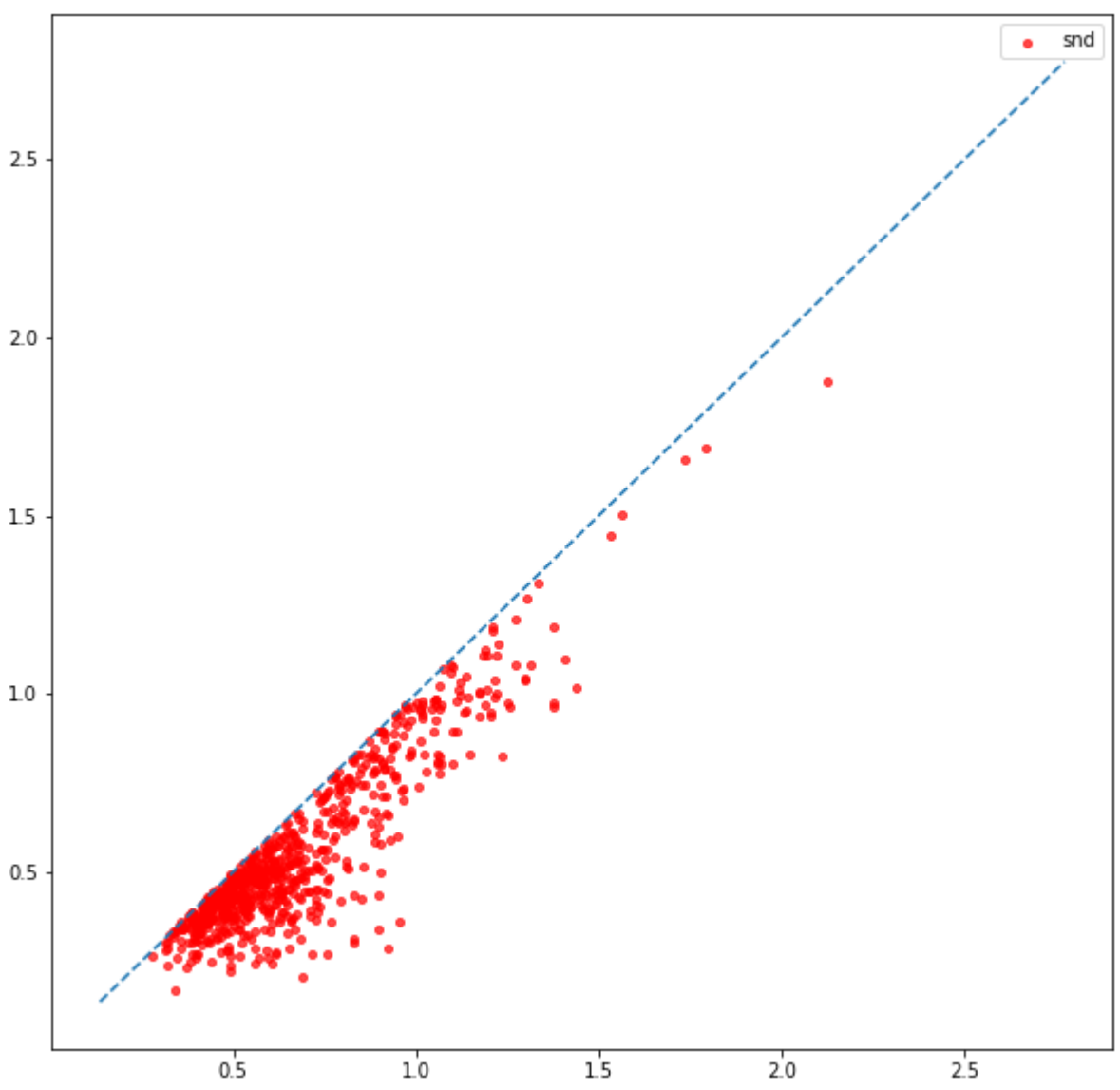
```
Fitting with configuration: [('one_shot', 0), ('dropout', 0), ('lr', 1), ('kern
el', <Kernel_T.DOT: 0>)]
```

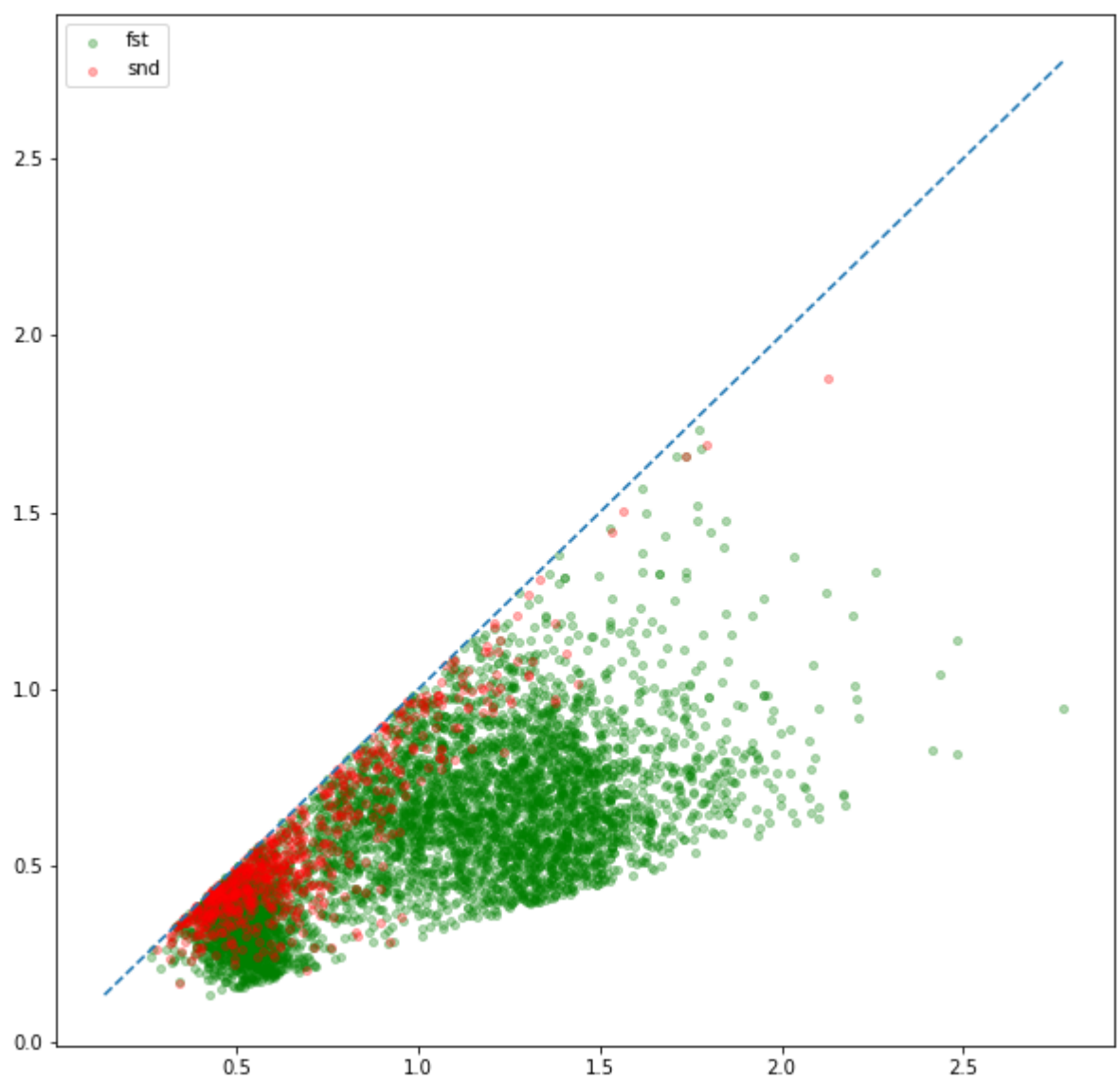
```
Train: 0.699823                Test: 0.808880
Train: 0.887494                Test: 0.900257
Train: 0.903750                Test: 0.942085
Train: 0.917431                Test: 0.897683
Train: 0.923225                Test: 0.949163
Train: 0.934009                Test: 0.902188
Train: 0.938033                Test: 0.937580
Train: 0.948334                Test: 0.895109
Train: 0.947207                Test: 0.955598
Train: 0.952680                Test: 0.954311
Train: 0.949783                Test: 0.937580
Train: 0.959440                Test: 0.959459
Train: 0.957026                Test: 0.901544
Train: 0.958957                Test: 0.949807
Train: 0.955738                Test: 0.953668
Train: 0.959118                Test: 0.958172

Normalized top1 and top 2 test accuracy: 0.8584298584298584 0.9845559845559846
```

In [13]:
```
fst_scs, snd_scs = hdc.analyze(trainencoded, trainlabels)
print("Len of fst, snd: ",len(fst_scs), len(snd_scs))

new_fst = []
beta = np.max(snd_scs[:,0]/snd_scs[:,1])
for sc in fst_scs:
    if sc[0]/sc[1] <= beta:
        new_fst.append(sc)
fst_scs = np.asarray(new_fst)

print("Len of fst after cropping: ",len(fst_scs))

show_plots(fst_scs, snd_scs)
```

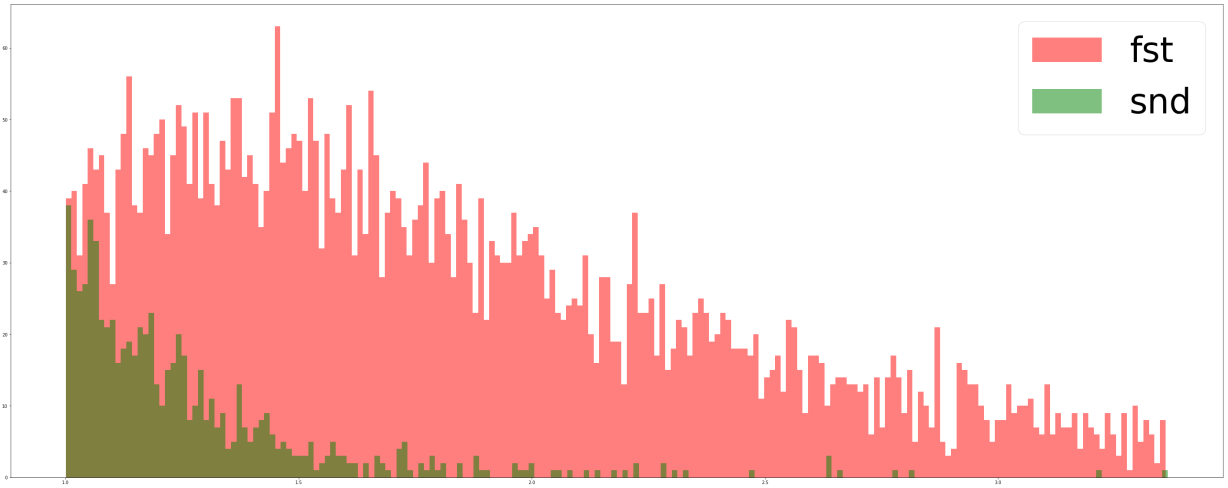
```
Len of fst, snd:  5492 721
Len of fst after cropping:  5137
```

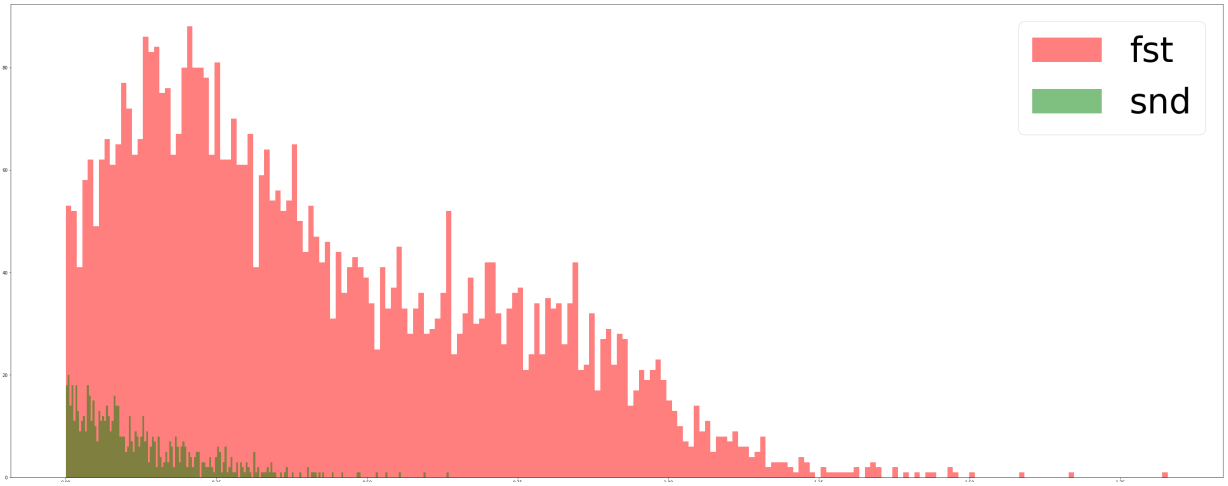



snd
2.5
2.0
1.5
1.0
0.5
0.5
1.0
1.5
2.0
2.5




```
0.5779203126441494 1.7475906960577172
```





In [14]:
```
dominance = 1.5
```

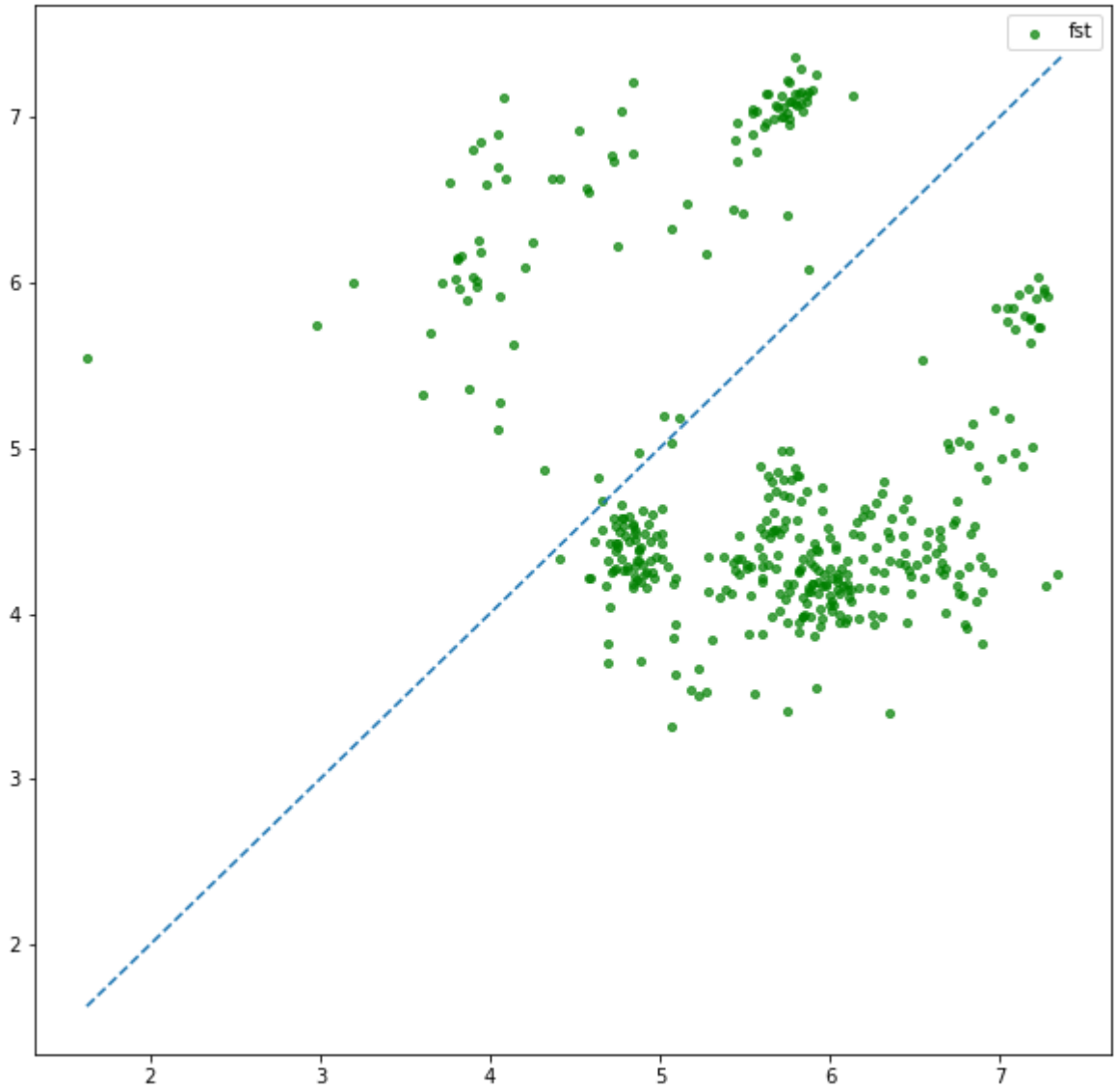
In [15]:
```
fst_scs, snd_scs = hdc.analyze(trainencoded, trainlabels, dominance)
print("Len of fst, snd: ",len(fst_scs), len(snd_scs))
#show_plots(fst_scs, snd_scs)
```

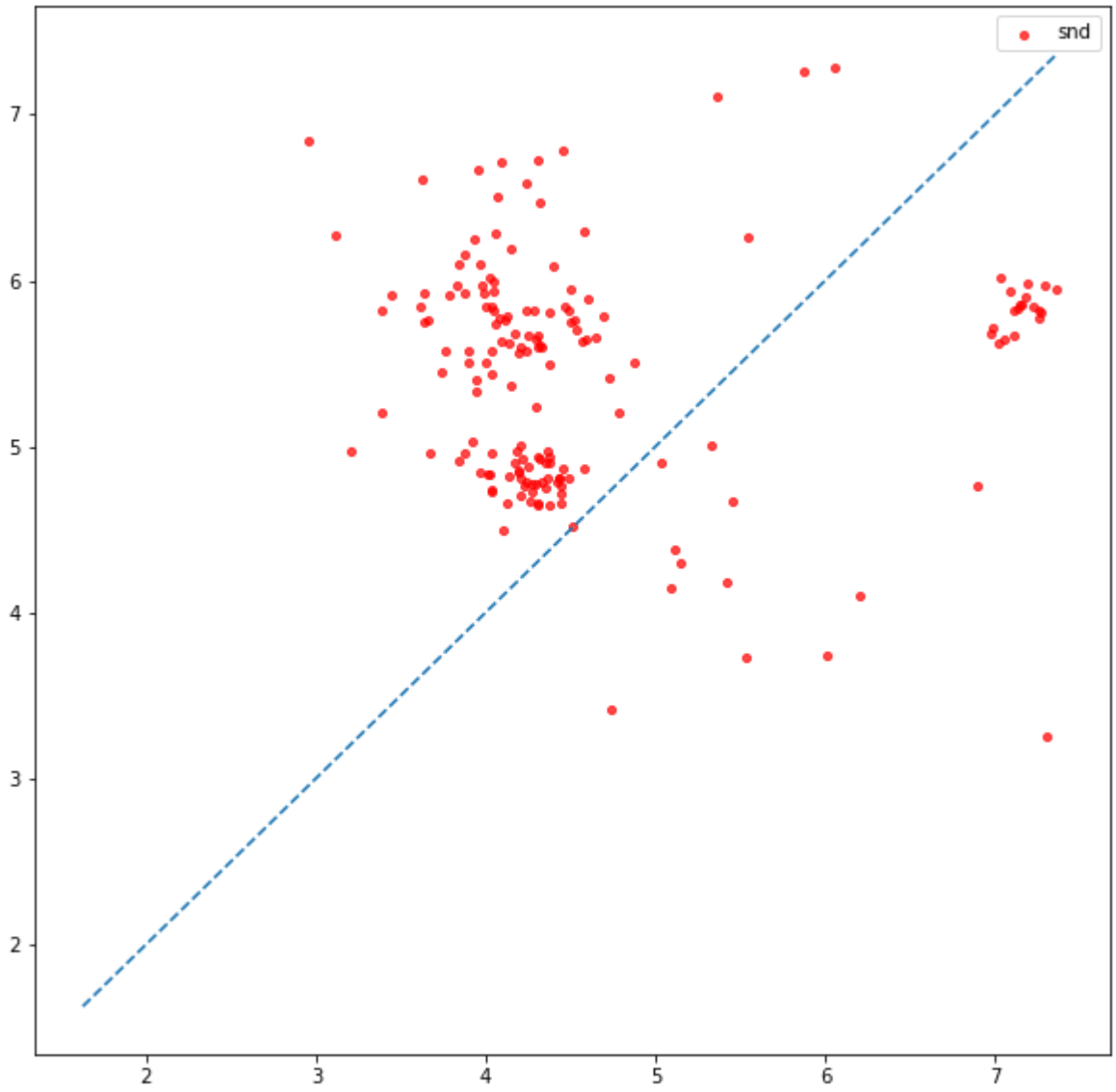
```
Len of fst, snd:  1867 628
```

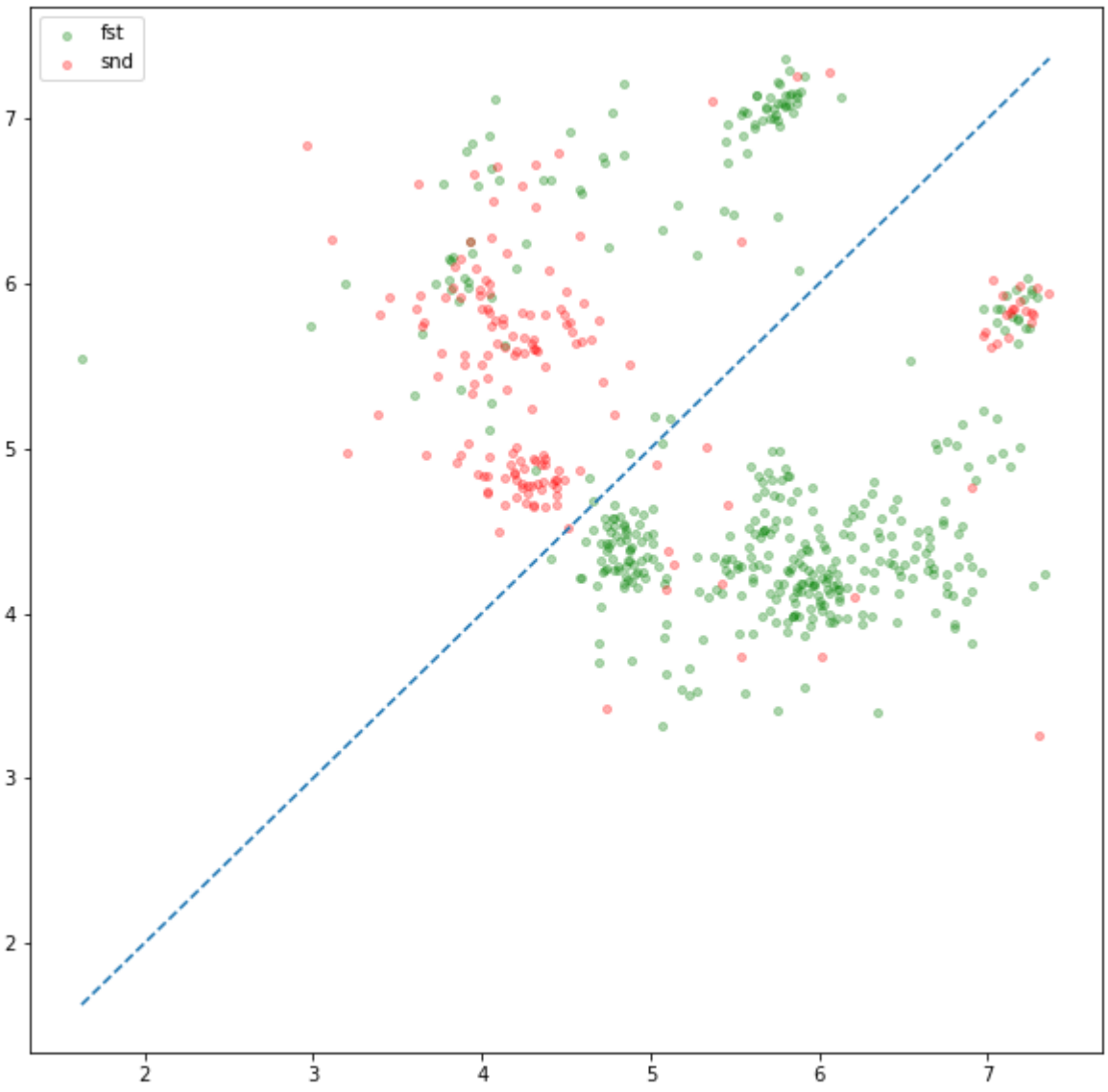
In [28]:
```
percent = 0.5
hdc.make_mask(trainencoded, trainlabels, dominance)
hdc.prep_mask(percent)
```

In [19]:
```
mask_t = "b" # b for binary, o for original
mask_d = 2
threshold = 0.3
hdc.set_decider(trainencoded, trainlabels, threshold, dominance, mask_t, mask_d)
fst_scs, snd_scs = hdc.analyze_topn(testencoded, testlabels, dominance, mask_t, m
print("Length fss, snd: ", len(fst_scs), len(snd_scs))
show_plots(fst_scs, snd_scs)
```

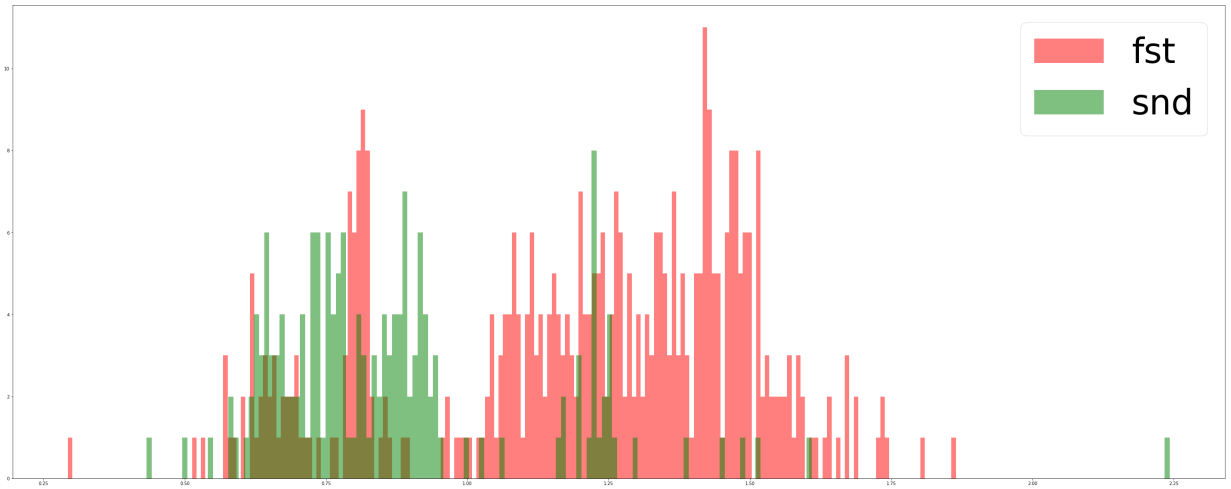
```
set_decider invokes test_mask. You may ignore output for now
[[        nan  1.          1.                nan         nan         nan
   1.                nan         nan         nan         nan         nan]
 [-1.                nan  1.                nan         nan         nan
   1.          1.                nan         nan  0.61904762        nan]
 [-0.9        -0.96428571        nan         nan         nan  1.
   0.9         1.                nan         nan  1.          1.        ]
 [-1.                nan  1.                nan -0.13043478  0.97058824
   0.85245902  0.6         1.          1.          1.                nan]
 [-1.                nan -1.         -0.90566038        nan  1.
   0.94078947        nan         nan         nan         nan         nan]
 [        nan         nan         nan  1.                nan         nan
            nan  1.                nan  0.92592593  1.          1.        ]
 [ 0.89473684  0.98529412  1.          0.59459459  0.95721925 -1.
            nan -0.2        -1.                nan -0.81818182        nan]
 [        nan         nan         nan  1.                nan         nan
  -1.                nan  0.         -1.                nan -1.        ]
 [        nan         nan         nan  1.                nan         nan
  -1.          -0.2               nan  1.          1.          1.        ]
 [        nan         nan         nan         nan         nan  1.
            nan -1.                nan         nan         nan  1.        ]
 [        nan         nan         nan         nan         nan         nan
  -1.          -0.33333333 -0.75              nan         nan  1.        ]
 [        nan         nan         nan  0.                nan  1.
            nan -1.          1.         -0.66666667        nan         nan]]
[[ 0.  1.  1.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
 [-1.  0.  1.  0.  0.  0.  1.  1.  0.  0.  0.  0.]
 [-1. -1.  0.  0.  0.  1.  1.  1.  0.  0.  1.  1.]
 [-1.  0.  1.  0. -1.  1.  1.  0.  1.  1.  1.  0.]
 [-1.  0. -1. -1.  0.  1.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  1.  0.  1.  1.  1.]
 [ 1.  1.  1.  0.  1. -1.  0. -1. -1.  0. -1.  0.]
 [ 0.  0.  0.  1.  0.  0. -1.  0. -1. -1.  0. -1.]
 [ 0.  0.  0.  1.  0.  0. -1. -1.  0.  1.  1.  1.]
 [ 0.  0.  0.  0.  0.  1.  0. -1.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0. -1. -1. -1.  0.  0.  1.]
 [ 0.  0.  0. -1.  0.  1.  0. -1.  1. -1.  0.  0.]]
Analyzing score with mask type b 2
Encounter unforseen mask: 11 6
Encounter unforseen mask: 8 5
Encounter unforseen mask: 4 7
Encounter unforseen mask: 9 10
Encounter unforseen mask: 4 7
Length fss, snd:  411 168
```

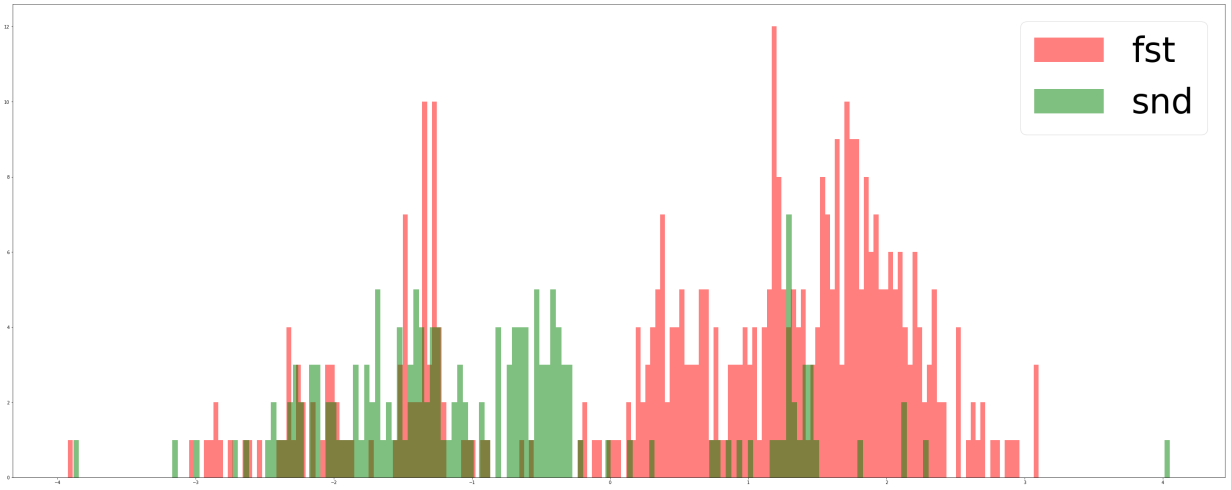

fst
7
6
5
4
3
2
2
3
4
5
6
7


snd
7
6
5
4
3
2
2
3
4
5
6
7


fst
snd
7
6
5
4
3
2
2
3
4
5
6
7

```
0.3179588030841755 1.1021564987205514
```





In [23]:
```
def evaluate(func, data, labels, threshold, dominance, mask_t, mask_d):
    correct, c_1, c_2, c_r, w_1, w_2, w_3, count, mat, net_mat = func(data, labe
    print("Accuracy: ", correct/count)
    print("Original: ", c_1/count)
    print("Correct: ",correct)
    print("First is Correct: ", c_1)
    print("Wrong second is selected: ", w_1)
    print("Second is Correct: ", c_2)
    print("Correct Second is selected: ", c_r)
    print("Wrong first is selected: ", w_2)
    print("Out of top 2: ", w_3)
    print("Net increase in correct cases and accuracy: ", c_r-w_1, (c_r-w_1)/cou
    print("Potential increase in correct cases and accuracy: ", c_2, c_2/count)
    print("Total cases:", count)
    print("Confusion Matrix:\n", mat)
    print("Net correct Matrix:\n", net_mat)
```

In [24]:
```
print("############################ MASK REVEAL ################################"
evaluate(hdc.test_mask, trainencoded, trainlabels, threshold, dominance, mask_t,

print("############################ DECIDER REVEAL ##############################
evaluate(hdc.test_decider, trainencoded, trainlabels, threshold, dominance, mask_
```

```
############################ MASK REVEAL ################################
Accuracy:  0.895219700627716
Original:  0.8839530017704812
Correct:  5562
First is Correct:  5492
Wrong second is selected:  402
Second is Correct:  650
Correct Second is selected:  472
Wrong first is selected:  178
Out of top 2:  71
Net increase in correct cases and accuracy:  70 0.01126669885723483
Potential increase in correct cases and accuracy:  650 0.10461934653146628
Total cases: 6213
Confusion Matrix:
 [[  0.  52.  64.   0.   0.   0. 173.   0.   0.   0.   0.   0.]
 [ 20.   0.  68.   0.   0.   0. 374.   1.   0.   0.  21.   0.]
 [ 20.  56.   0.   0.   0.   1.  20.   1.   0.   0.   1.   2.]
 [  3.   0.   1.   0. 138.  68. 122.   5.   1.  11.   1.   0.]
 [  5.   0.   2. 212.   0.   3. 304.   0.   0.   0.   0.   0.]
 [  0.   0.   0.  29.   0.   0.   0.   4.   0.  54.   1.  10.]
 [ 38. 136.   1.  74. 187.   2.   0.  10.   9.   0.  11.   0.]
 [  0.   0.   0.   4.   0.   0.  10.   0.   2.   3.   0.   1.]
 [  0.   0.   0.   1.   0.   0.  10.   5.   0.   1.  19.   5.]
 [  0.   0.   0.   0.   0.   1.   0.   9.   0.   0.   0.  31.]
 [  0.   0.   0.   0.   0.   0.  28.   3.   8.   0.   0.   6.]
 [  0.   0.   0.   2.   0.   2.   0.   3.   1.  24.   0.   0.]]
Net correct Matrix:
 [[   0.   52.   64.    0.    0.    0.  173.    0.    0.    0.    0.    0.]
 [ -20.    0.   68.    0.    0.    0.  374.    1.    0.    0.   13.    0.]
 [ -18.  -54.    0.    0.    0.    1.   18.    1.    0.    0.    1.    2.]
 [  -3.    0.    1.    0.  -18.   66.  104.    3.    1.   11.    1.    0.]
 [  -5.    0.   -2. -192.    0.    3.  286.    0.    0.    0.    0.    0.]
 [   0.    0.    0.   29.    0.    0.    0.    4.    0.   50.    1.   10.]
 [  34.  134.    1.   44.  179.   -2.    0.   -2.   -9.    0.   -9.    0.]
 [   0.    0.    0.    4.    0.    0.  -10.    0.    0.   -3.    0.   -1.]
 [   0.    0.    0.    1.    0.    0.  -10.   -1.    0.    1.   19.    5.]
 [   0.    0.    0.    0.    0.    1.    0.   -9.    0.    0.    0.   31.]
 [   0.    0.    0.    0.    0.    0.  -28.   -1.   -6.    0.    0.    6.]
 [   0.    0.    0.    0.    0.    2.    0.   -3.    1.  -16.    0.    0.]]
############################ DECIDER REVEAL ##############################
Accuracy:  0.9509093835506197
Original:  0.8839530017704812
Correct:  5908
First is Correct:  5492
Wrong second is selected:  75
Second is Correct:  650
Correct Second is selected:  491
Wrong first is selected:  159
Out of top 2:  71
Net increase in correct cases and accuracy:  416 0.06695638178013842
Potential increase in correct cases and accuracy:  650 0.10461934653146628
```

```
Total cases: 6213
Confusion Matrix:
 [[  0.  52.  64.   0.   0.   0. 173.   0.   0.   0.   0.   0.]
 [ 20.   0.  68.   0.   0.   0. 374.   1.   0.   0.  21.   0.]
 [ 20.  56.   0.   0.   0.   1.  20.   1.   0.   0.   1.   2.]
 [  3.   0.   1.   0. 138.  68. 122.   5.   1.  11.   1.   0.]
 [  5.   0.   2. 212.   0.   3. 304.   0.   0.   0.   0.   0.]
 [  0.   0.   0.  29.   0.   0.   0.   4.   0.  54.   1.  10.]
 [ 38. 136.   1.  74. 187.   2.   0.  10.   9.   0.  11.   0.]
 [  0.   0.   0.   4.   0.   0.  10.   0.   2.   3.   0.   1.]
 [  0.   0.   0.   1.   0.   0.  10.   5.   0.   1.  19.   5.]
 [  0.   0.   0.   0.   0.   1.   0.   9.   0.   0.   0.  31.]
 [  0.   0.   0.   0.   0.   0.  28.   3.   8.   0.   0.   6.]
 [  0.   0.   0.   2.   0.   2.   0.   3.   1.  24.   0.   0.]]
Net correct Matrix:
 [[  0.  52.  64.   0.   0.   0. 173.   0.   0.   0.   0.   0.]
 [ 20.   0.  68.   0.   0.   0. 374.   1.   0.   0.  21.   0.]
 [ 18.  54.   0.   0.   0.   1.  18.   1.   0.   0.   1.   2.]
 [  3.   0.   1.   0.  18.  66. 104.   3.   1.  11.   1.   0.]
 [  1.   0.   2. 192.   0.   3. 286.   0.   0.   0.   0.   0.]
 [  0.   0.   0.  29.   0.   0.   0.   4.   0.  50.   1.  10.]
 [ 34. 134.   1. -74. 179.  -2.   0. -10.   9.   0.   3.   0.]
 [  0.   0.   0.   4.   0.   0.  -2.   0.  -2.   3.   0.   1.]
 [  0.   0.   0.   1.   0.   0.  10.   1.   0.   1.  19.   5.]
 [  0.   0.   0.   0.   0.   1.   0.   9.   0.   0.   0.  31.]
 [  0.   0.   0.   0.   0.   0.  28.   1.   6.   0.   0.   6.]
 [  0.   0.   0.  -2.   0.   2.   0.   3.   1.  16.   0.   0.]]
```

In [25]:
```
print("############################ MASK REVEAL ################################"
evaluate(hdc.test_mask, testencoded, testlabels, threshold, dominance, mask_t, ma

print("############################ DECIDER REVEAL ##############################
evaluate(hdc.test_decider, testencoded, testlabels, threshold, dominance, mask_t
```

```
############################ MASK REVEAL ################################
Accuracy:  0.8835263835263836
Original:  0.8584298584298584
Correct:  1373
First is Correct:  1334
Wrong second is selected:  97
Second is Correct:  196
Correct Second is selected:  136
Wrong first is selected:  60
Out of top 2:  24
Net increase in correct cases and accuracy:  39 0.025096525096525095
Potential increase in correct cases and accuracy:  196 0.12612612612612611
Total cases: 1554
Confusion Matrix:
 [[ 0. 10. 18.  0.  1.  0. 42.  0.  0.  0.  0.  0.]
 [11.  0. 15.  0.  0.  0. 69.  1.  0.  0.  4.  0.]
 [11. 13.  0.  0.  0.  0.  3.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0. 37. 27. 27.  2.  1.  0.  0.  1.]
 [ 0.  0.  0. 44.  0.  1. 68.  2.  0.  0.  0.  0.]
 [ 0.  0.  0.  8.  0.  0.  1.  0.  0. 10.  0.  2.]
 [11. 38.  0. 18. 51.  0.  0.  1.  2.  1.  5.  0.]
 [ 0.  1.  0.  0.  0.  1.  2.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  3.  2.  0.  0.  2.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  1.  3.]
 [ 0.  3.  0.  0.  0.  0.  4.  0.  7.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  1.  1.  1.  0.  8.  1.  0.]]
Net correct Matrix:
 [[  0.   8.  18.   0.  -1.   0.  42.   0.   0.   0.   0.   0.]
 [-11.   0.  11.   0.   0.   0.  69.   1.   0.   0.  -2.   0.]
 [ -1. -11.   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.]
 [  0.   0.   1.   0.  -1.  27.  25.   0.   1.   0.   0.   1.]
 [  0.   0.   0. -38.   0.   1.  68.   2.   0.   0.   0.   0.]
 [  0.   0.   0.   8.   0.   0.  -1.   0.   0.   8.   0.   2.]
 [  9.  36.   0.  12.  47.   0.   0.  -1.  -2.  -1.  -5.   0.]
 [  0.   1.   0.   0.   0.   1.  -2.   0.   0.  -1.   0.   0.]
 [  0.   0.   0.   0.   0.   1.  -3.  -2.   0.   0.   2.   0.]
 [  0.   0.   0.   0.   0.  -1.   0.   0.   0.   0.   1.   1.]
 [  0.  -1.   0.   0.   0.   0.  -4.   0.  -5.   0.   0.   0.]
 [  0.   0.  -1.   0.   0.   1.   1.  -1.   0.  -4.   1.   0.]]
############################ DECIDER REVEAL ##############################
Accuracy:  0.9298584298584298
Original:  0.8584298584298584
Correct:  1445
First is Correct:  1334
Wrong second is selected:  19
Second is Correct:  196
Correct Second is selected:  130
Wrong first is selected:  66
Out of top 2:  24
Net increase in correct cases and accuracy:  111 0.07142857142857142
Potential increase in correct cases and accuracy:  196 0.12612612612612611
```

```
Total cases: 1554
Confusion Matrix:
 [[ 0. 10. 18.  0.  1.  0. 42.  0.  0.  0.  0.  0.]
 [11.  0. 15.  0.  0.  0. 69.  1.  0.  0.  4.  0.]
 [11. 13.  0.  0.  0.  0.  3.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0. 37. 27. 27.  2.  1.  0.  0.  1.]
 [ 0.  0.  0. 44.  0.  1. 68.  2.  0.  0.  0.  0.]
 [ 0.  0.  0.  8.  0.  0.  1.  0.  0. 10.  0.  2.]
 [11. 38.  0. 18. 51.  0.  0.  1.  2.  1.  5.  0.]
 [ 0.  1.  0.  0.  0.  1.  2.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  3.  2.  0.  0.  2.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  1.  3.]
 [ 0.  3.  0.  0.  0.  0.  4.  0.  7.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  1.  1.  1.  0.  8.  1.  0.]]
Net correct Matrix:
 [[  0.   8.  18.   0.  -1.   0.  42.   0.   0.   0.   0.   0.]
 [ 11.   0.  11.   0.   0.   0.  69.   1.   0.   0.   4.   0.]
 [  1.  11.   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.]
 [  0.   0.   1.   0.   1.  27.  25.   0.   1.   0.   0.   1.]
 [  0.   0.   0.  38.   0.   1.  68.   2.   0.   0.   0.   0.]
 [  0.   0.   0.   8.   0.   0.  -1.   0.   0.   8.   0.   2.]
 [  9.  36.   0. -18.  47.   0.   0.  -1.   2.  -1.   1.   0.]
 [  0.  -1.   0.   0.   0.  -1.   0.   0.   0.   1.   0.   0.]
 [  0.   0.   0.   0.   0.   1.   3.   2.   0.   0.   2.   0.]
 [  0.   0.   0.   0.   0.  -1.   0.   0.   0.   0.   1.   1.]
 [  0.  -3.   0.   0.   0.   0.   4.   0.   5.   0.   0.   0.]
 [  0.   0.  -1.   0.   0.   1.   1.   1.   0.   4.  -1.   0.]]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Scratch Paper

In [ ]:
```
# Select two classes to separate
cl1 = 3
cl2 = 4

ncl1 = 0
cl1_sum = np.zeros(hdc.D)
for sc in fst_scs:
    if sc[3] == cl1 and sc[4] == cl2:
        cl1_sum += trainencoded[int(sc[2])]
        ncl1 += 1
for sc in snd_scs:
    if sc[3] == cl2 and sc[4] == cl1:
        cl1_sum += trainencoded[int(sc[2])]
        ncl1 += 1
ncl2 = 0
cl2_sum = np.zeros(hdc.D)
for sc in fst_scs:
    if sc[3] == cl2 and sc[4] == cl1:
        cl2_sum += trainencoded[int(sc[2])]
        ncl2 += 1
for sc in snd_scs:
    if sc[3] == cl1 and sc[4] == cl2:
        cl2_sum += trainencoded[int(sc[2])]
        ncl2 += 1
print("Component-wise sum of data points, sum1 sum2")
#print(cl1_sum)
#print(cl2_sum)

#print("Component-wise ratio sum1:sum2")
#print(cl1_sum/cl2_sum)
cl1_nm = sklearn.preprocessing.normalize(np.asarray([cl1_sum]), norm='l2')[0]
cl2_nm = sklearn.preprocessing.normalize(np.asarray([cl2_sum]), norm='l2')[0]
print("Component-wise normalized ratio normed(sum1):normed(sum2)")
print(cl1_nm/cl2_nm)
#print("Component-wise normalized diff normed(sum1) - normed(sum2)")
#print(cl1_nm - cl2_nm)
```

In [ ]:
```
og_mask1 = copy.deepcopy(hdc.mask2d[cl1][cl2])
og_mask2 = copy.deepcopy(hdc.mask2d[cl2][cl1])
hdc.mask2d[cl1][cl2] = mask
hdc.mask2d[cl2][cl1] = -mask
```

In [ ]:
```
hdc.mask2d[cl1][cl2] = og_mask1
hdc.mask2d[cl2][cl1] = og_mask2
```

In [ ]:
```

```

In [ ]:
```
np.set_printoptions(linewidth=85, suppress = True)
fst_mat = np.zeros((hdc.nClasses, hdc.nClasses))
for sc in fst_scs:
    fst_mat[int(sc[3]), int(sc[4])] += 1

snd_mat = np.zeros((hdc.nClasses, hdc.nClasses))
for sc in snd_scs:
    snd_mat[int(sc[3]), int(sc[4])] += 1
print("fst confusion matrix: top 2 choices (row, col) whose answer is first choi
print(fst_mat)
print("snd confusion matrix: top 2 choices (row, col) whose answer is second cho
print(snd_mat)
```

In [ ]:
```
```

In [ ]:
```
fst_r, snd_r = hdc.analyze(testencoded, testlabels)

print(len(fst_r), len(snd_r))
#beta = max(snd_r)
#fst_r = fst_r[fst_r <= beta]
print(len(fst_r))

fst_mr, snd_mr, fst_scs, snd_scs = hdc.analyze_topn(testencoded, testlabels, dom

print(len(fst_mr), len(snd_mr))
print(len(fst_scs), len(snd_scs))
#print(fst_mr)
#print(snd_mr)
#print(fst_scs)
#print(snd_scs)


plt.figure(figsize = (100, 20))
plt.hist(np.append(fst_mr,snd_mr), bins = 1000, label = "fst")
plt.hist(snd_mr, bins = 1000, label = "snd")
plt.legend()

plt.figure(figsize = (100, 20))
plt.hist(np.append(fst_scs,snd_scs), bins = 1000, label = "fst")
plt.hist(snd_scs, bins = 1000, label = "snd")
plt.legend()
```

In [ ]:
```
```

In [21]:
```
np.set_printoptions(threshold=sys.maxsize)
```

In [ ]:
```
```