PCA analysis

April 7, 2020

Navigate to directory containing .mat data.

[1]:
```
cd ../_data/matlabData/
```

```
/home/gustav/Documents/DD142X/code/_data/matlabData
```

Function definitions for data extraction and feature extraction. Basic imports.

[2]:
```
import numpy as np
import matplotlib.pyplot as plt

from h5py import File
def getMatlabValues(fileName):
    with File(fileName, "r") as data:
        return {
            key: np.array(data[key]["values"]).flatten() for key in data.keys()
        }

# Fourier Feature Vector
def ffv(xs, Fs = 16000., epoch_size = 2 ** 11, fft_n = 2 ** 14):

    # Pad with zeroes for more frequency outputs
    # Compare np.fft.fftfreq(n, 1/16000) for n = 2**11, 2**14
    fft_in = np.zeros((xs.shape[0], fft_n))
    fft_in[ : , 0:epoch_size] = xs

    frqs = np.fft.fftfreq(fft_n, 1./Fs)
    lo = np.where(frqs > 12)[0][0]
    hi = np.where(frqs > 30)[0][0]
    fftxs = np.abs(np.fft.fft(fft_in)[:,lo:hi])

    return fftxs, frqs[lo:hi]
```

I/O

[3]:
```
mlDict = getMatlabValues("NPR-075.b11.mat")

vals_str = np.array([v for k, v in mlDict.items() if "str_lfp" in k])
```

```
vals_gp  = np.array([v for k, v in mlDict.items() if "gp_lfp"  in k])
vals_all = np.concatenate((vals_str, vals_gp), axis = 0)

print(vals_str.shape)
print(vals_gp.shape)
print(vals_all.shape)
```

```
(11, 1587696)
(15, 1587696)
(26, 1587696)
```

First, consider PCA of "raw" data. Trim excess datapoints in order to work with 2^n length epochs.

[4]:
```
print( vals_str.shape[1] / 2 ** 14 )
print(  2 ** 14 / 16000 )
```

```
96.9052734375
1.024
```

2 ** 14 seems like a reasonable maximum epoch size.

[5]:
```
vals_str = vals_str[ : , 0 : 2 ** 14 * 96]
vals_gp  = vals_gp [ : , 0 : 2 ** 14 * 96]
vals_all = vals_all[ : , 0 : 2 ** 14 * 96]

print(vals_str.shape)
print(vals_gp.shape)
print(vals_all.shape)
```

```
(11, 1572864)
(15, 1572864)
(26, 1572864)
```

Consider amount of required principal components to explain variance to some degree.

[6]:
```
from sklearn.decomposition import PCA

print("### PCA explained variance ratio per component and sum, n_components =␣
 ↪10")
for epoch_size in [
    2 ** 14, 2 ** 13, # ~1s, 0.5s
    2 ** 12, 2 ** 11, # ~0.25s, 0.125s
    2 ** 10, 2 ** 9   # Very short, may cause memory issues - hardly optimized
]:
    # Investigate for large n_components
    print("### Epoch size " + str(epoch_size))
```

```
    pca_str = PCA(n_components = 10).fit(vals_str.copy().reshape((-1,␣
↪epoch_size)))
    pca_gp  = PCA(n_components = 10).fit(vals_gp.copy().reshape((-1,␣
↪epoch_size)))
    pca_all = PCA(n_components = 10).fit(vals_all.copy().reshape((-1,␣
↪epoch_size)))
    print("\tStriatum")
    print(pca_str.explained_variance_ratio_)
    print(pca_str.explained_variance_ratio_.sum())
    print("\tGlobus pallidus")
    print(pca_gp.explained_variance_ratio_)
    print(pca_gp.explained_variance_ratio_.sum())
    print("\tAll")
    print(pca_all.explained_variance_ratio_)
    print(pca_all.explained_variance_ratio_.sum())
    print()
```

```
### PCA explained variance ratio per component and sum, n_components = 10
### Epoch size 16384
        Striatum
[0.13201564 0.09549128 0.08267939 0.0709507  0.06386331 0.05941619
 0.0441119  0.03142206 0.02750077 0.0236469 ]
0.6310981406886702
        Globus pallidus
[0.13970142 0.09407339 0.08341252 0.05959512 0.05431434 0.04960502
 0.04085061 0.03476214 0.02840412 0.02579093]
0.6105096176335426
        All
[0.13595805 0.09411479 0.08162108 0.06524052 0.05891072 0.05358219
 0.042126   0.03069342 0.02821657 0.02719612]
0.6176594572434613

### Epoch size 8192
        Striatum
[0.15187149 0.12482922 0.1079342  0.09913861 0.08795753 0.05288573
 0.03481608 0.01860035 0.0158545  0.01430079]
0.7081884955711628
        Globus pallidus
[0.15902287 0.11923743 0.10791143 0.08742827 0.07933465 0.0506975
 0.03192745 0.02011317 0.0173706  0.01659357]
0.6896369353644543
        All
[0.154862   0.11412229 0.1134704  0.09218164 0.0860637  0.05165942
 0.03317962 0.01921937 0.01633495 0.01554248]
0.6966358805628426

### Epoch size 4096
```

```
        Striatum
[0.24321588 0.1905     0.16359195 0.06412213 0.02667111 0.0245737
 0.02319225 0.02064487 0.01704692 0.01646161]
0.7900204243349113
        Globus pallidus
[0.26572307 0.17554352 0.13646579 0.05858759 0.0294942  0.02782842
 0.02592714 0.02423092 0.02231361 0.02044605]
0.7865603148086454
        All
[0.25507889 0.18175209 0.14923832 0.06116898 0.02756925 0.02579797
 0.02503227 0.02305841 0.01966268 0.01857207]
0.786930917423995

### Epoch size 2048
        Striatum
[0.40333083 0.23222896 0.06722281 0.03842049 0.0361776  0.03197513
 0.02217892 0.01402925 0.01211575 0.01169958]
0.869379317838865
        Globus pallidus
[0.41422713 0.20006564 0.0651903  0.04534713 0.04263925 0.03980649
 0.02984995 0.0143692  0.00859884 0.00816665]
0.8682605721736838
        All
[0.40943561 0.21405838 0.06612751 0.04061722 0.04050694 0.03719184
 0.02663702 0.0142366  0.00992589 0.00965878]
0.868395782377294

### Epoch size 1024
        Striatum
[0.59032811 0.14154979 0.06555062 0.04238349 0.01959164 0.01372395
 0.0126338  0.01261415 0.01141206 0.00879597]
0.9185835912056073
        Globus pallidus
[0.57932412 0.1326198  0.08038562 0.05201119 0.01938242 0.01218263
 0.00965168 0.00853804 0.00824053 0.00717727]
0.9095133056389798
        All
[0.58428032 0.13632042 0.0738323  0.04791357 0.0194909  0.01283283
 0.01079399 0.01031238 0.00977479 0.00792491]
0.9134764181012277

### Epoch size 512
        Striatum
[0.71040784 0.11626763 0.0421993  0.02107727 0.01403551 0.0135176
 0.0115724  0.00671003 0.0045563  0.00453135]
0.944875222720114
        Globus pallidus
[0.69615742 0.12981686 0.04567163 0.0176672  0.01094913 0.00946829
```

```
 0.00841649 0.00533999 0.00359789 0.00348624]
0.930571149900974
        All
[0.70249968 0.12381821 0.0441189  0.0191505  0.01190735 0.0114102
 0.0100411  0.00594364 0.00401489 0.00392155]
0.9368260280254997
```

At n >= 2048 points per epoch, lots of variance can be explained with very few components Even at lower greater n, much variance can be explained with somewhat few components. Could be an interesting alternative to fourier transform. High dimensionality (large number of principal components) not a big problem - very large dataset.

The weight of each PC in a resulting PC-based feature vector could be scaled by explained variance ratio, or some other technique along those lines.

PCA, however, ignores spectral domain. This could be problematic. PCA gives information only on "entire" LFP, not beta-range, which we are targeting(?). PCA can be applied to Fourier-transform-based feature vector "ffv".

[7]:
```python
# Number of features extracted remains (in this example) constant at 18

print("### PCA explained variance ratio per component and sum, for DFT-based␣
 ↪feature vector, n_components = 10")
for epoch_size in [
    2 ** 14, 2 ** 13,
    2 ** 12, 2 ** 11,
    2 ** 10, #2 ** 9   # Greater risk of memory issues, workaround not␣
 ↪implemented
]:
    print("### Epoch size " + str(epoch_size))
    ffv_str, _ = ffv(vals_str.reshape((-1, epoch_size)), epoch_size=epoch_size)
    ffv_gp,  _ = ffv(vals_gp.reshape((-1, epoch_size)), epoch_size=epoch_size)
    ffv_all, _ = ffv(vals_all.reshape((-1, epoch_size)), epoch_size=epoch_size)

    pca_str = PCA(n_components = 10).fit(ffv_str)
    pca_gp  = PCA(n_components = 10).fit(ffv_gp)
    pca_all = PCA(n_components = 10).fit(ffv_all)

    print("\tStriatum")
    print(pca_str.explained_variance_ratio_)
    print(pca_str.explained_variance_ratio_.sum())
    print("\tGlobus pallidus")
    print(pca_gp.explained_variance_ratio_)
    print(pca_gp.explained_variance_ratio_.sum())
    print("\tAll")
    print(pca_all.explained_variance_ratio_)
```

```python
    print(pca_all.explained_variance_ratio_.sum())
    print()
```

```
### PCA explained variance ratio per component and sum, for DFT-based feature
vector, n_components = 10
### Epoch size 16384
        Striatum
[0.1282068  0.10920921 0.09645611 0.08493102 0.07250854 0.06865115
 0.06047625 0.05241997 0.04776931 0.0458104 ]
0.7664387619805524
        Globus pallidus
[0.12801003 0.11145797 0.1017971  0.08601633 0.08069299 0.06828279
 0.06331946 0.05063642 0.04935059 0.0448874 ]
0.7844510793717202
        All
[0.12412244 0.10926272 0.10029431 0.08297152 0.07811252 0.06657244
 0.0614023  0.05190005 0.05048729 0.04812489]
0.7732504782016224

### Epoch size 8192
        Striatum
[0.1748784  0.12133393 0.10323539 0.08420862 0.08053167 0.06975029
 0.05514768 0.05259114 0.04747176 0.04044812]
0.8295969892428818
        Globus pallidus
[0.17082097 0.12842248 0.10912594 0.08594918 0.07538095 0.06941832
 0.06502151 0.04974524 0.04680639 0.04200086]
0.8426918253390822
        All
[0.1725584  0.12378346 0.11180106 0.08366169 0.07423475 0.06750929
 0.06530015 0.05014109 0.04681424 0.04113517]
0.8369392858187162

### Epoch size 4096
        Striatum
[0.273996   0.17234378 0.13692129 0.11220554 0.09711271 0.06560395
 0.05480532 0.03874422 0.02356104 0.01272468]
0.988018543257701
        Globus pallidus
[0.26164328 0.18845243 0.14646079 0.11317954 0.09679361 0.0653134
 0.04899552 0.03669224 0.02074158 0.01018742]
0.9884598037893423
        All
[0.26624254 0.18184926 0.14380045 0.1129786  0.09698456 0.06495389
 0.05114059 0.03749518 0.02175697 0.01113666]
```

```
0.9883387127388763

### Epoch size 2048
        Striatum
[0.41683415 0.24872985 0.16154208 0.09506797 0.048629   0.02002439
 0.00429341 0.00217353 0.00112592 0.00061941]
0.9990396995637371
        Globus pallidus
[0.41338799 0.27232533 0.15905927 0.09093397 0.04092755 0.01537856
 0.00375176 0.00191762 0.00095964 0.00052202]
0.9991637007728513
        All
[0.41318286 0.26542191 0.15930363 0.09251834 0.04384442 0.017278
 0.00396417 0.00201686 0.00102439 0.0005598 ]
0.9991143726761063

### Epoch size 1024
        Striatum
[5.72572710e-01 3.04698605e-01 9.62531583e-02 2.11363273e-02
 3.27128953e-03 1.21383789e-03 4.43804900e-04 1.92001837e-04
 9.52299538e-05 4.86289962e-05]
0.9999255943749663
        Globus pallidus
[6.01913009e-01 2.95083555e-01 8.02725612e-02 1.82400094e-02
 2.88568605e-03 9.45966296e-04 3.40439810e-04 1.50135749e-04
 7.16472555e-05 4.02464541e-05]
0.9999432562034543
        All
[5.88729630e-01 3.00031812e-01 8.69380948e-02 1.94399456e-02
 3.05832724e-03 1.06001920e-03 3.85091125e-04 1.67044029e-04
 8.17557527e-05 4.39151784e-05]
0.9999356353150095
```

Good potential for feature reduction for sufficiently short epoch sizes. 10 components do a very good job at n = 4096. 6 components good enough for ~85% variance. 6 components do a very good job at n = 2048. 4 components good enough for >90% variance. At n = 1024, 2 components ~85% variance. 3 components >90%. Memory issues at n = 512. Workaround pending.

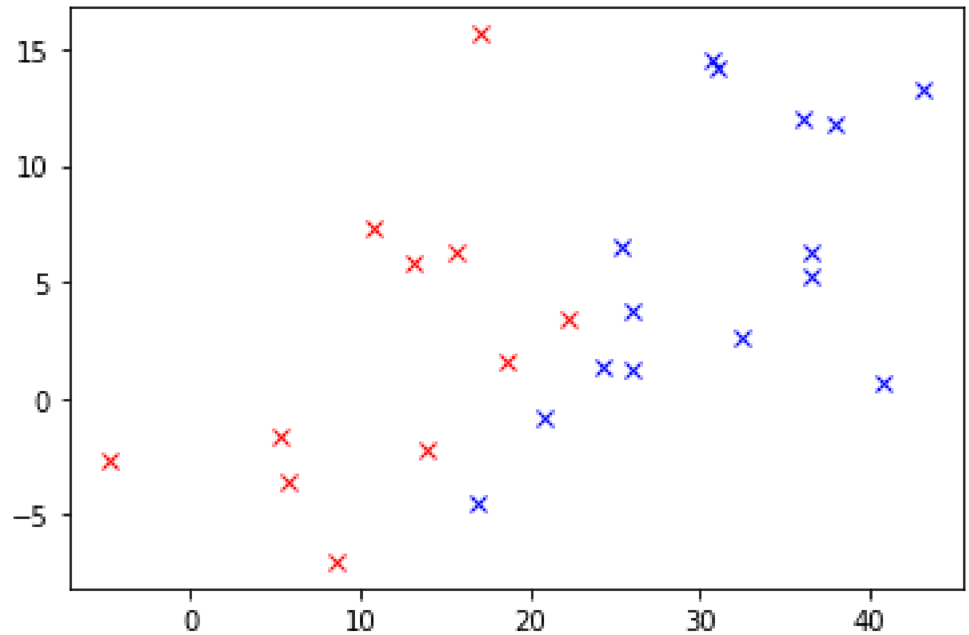
If used as feature vectors, should take explained variance ratio into account (as previously discussed).

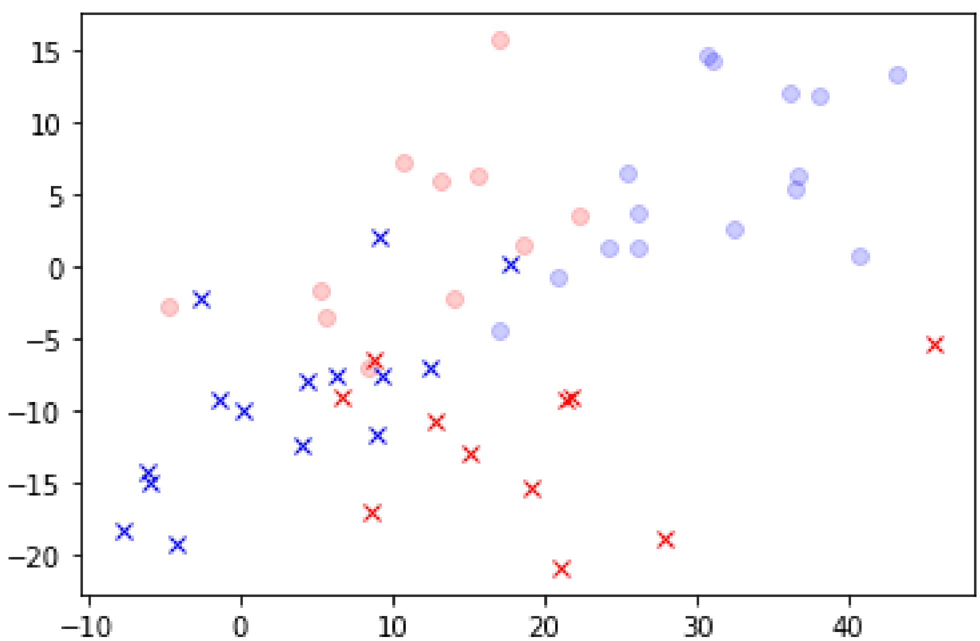
[8]:
```
ffv_all, _ = ffv(
        vals_all.reshape((-1, 1024)),
        epoch_size = 1024
    )
```

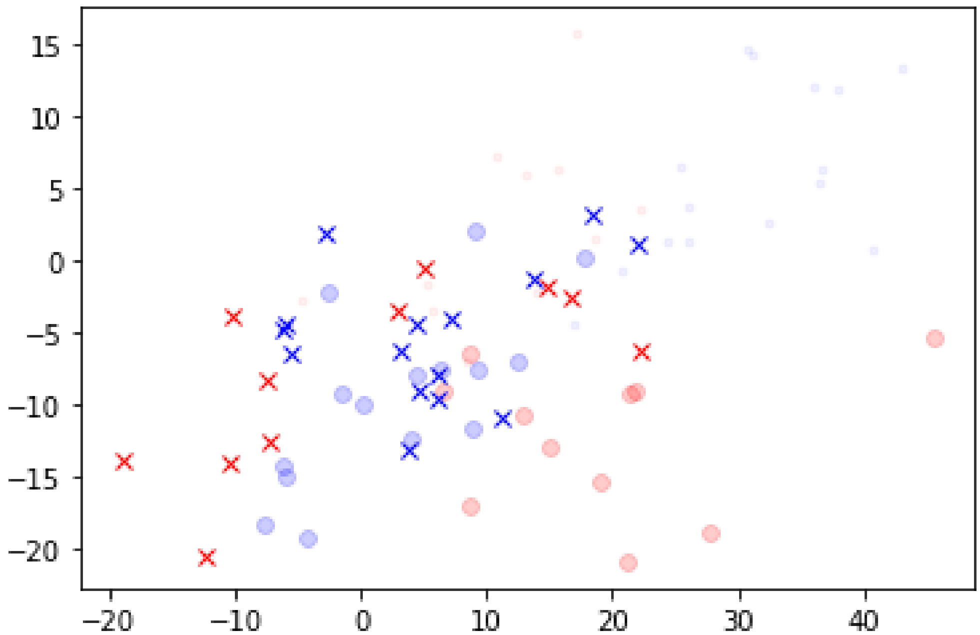
```python
pca_plot = PCA(n_components=2).fit(ffv_all)
embeddings = pca_plot.transform(ffv_all)

# 26 channels * n epochs * 2 principal components
embeddings = embeddings.reshape((26, -1, 2))

# Split into channel groups
str_tmp = embeddings[0:11 , : , : ]
gp_tmp  = embeddings[11:26, : , : ]
```

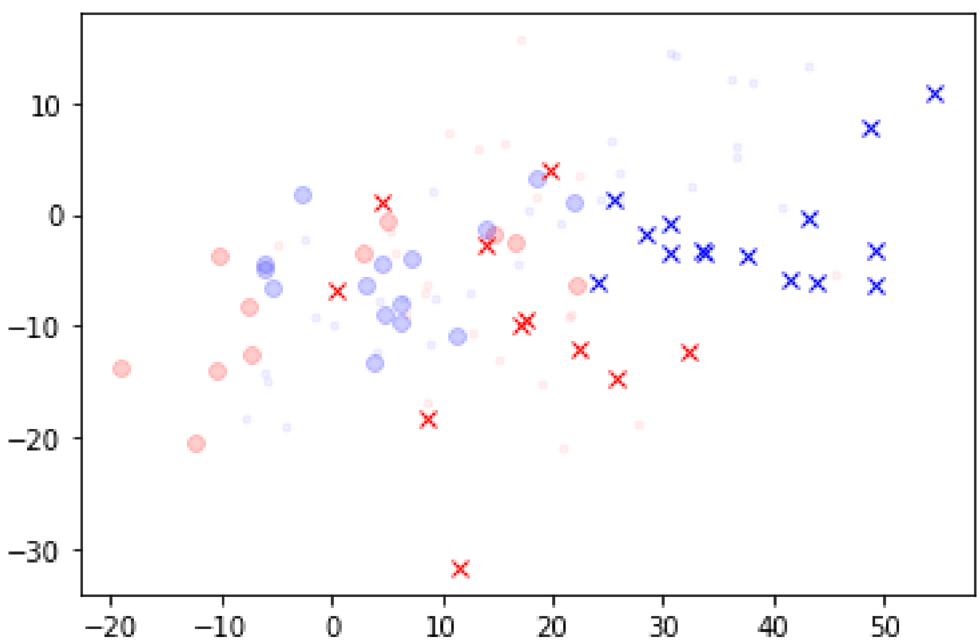
[9]:
```python
EPOCH_MAX = 20

plt.clf()
for hi in range(0, EPOCH_MAX):
    for ep in range(0, hi):

        alpha = 0.05
        shape = '.'
        if ep == hi - 1:
            alpha = 1
            shape = 'x'
        elif ep == hi - 2:
            shape = 'o'
            alpha = 0.2

        plt.plot(
            [x for x,_ in str_tmp[ : , ep]],
            [y for _,y in str_tmp[ : , ep]],
            'r' + shape,
            alpha=alpha
        )

        plt.plot(
            [x for x,_ in gp_tmp[ : , ep]],
            [y for _,y in gp_tmp[ : , ep]],
            'b' + shape,
            alpha=alpha
        )
    plt.show()
```

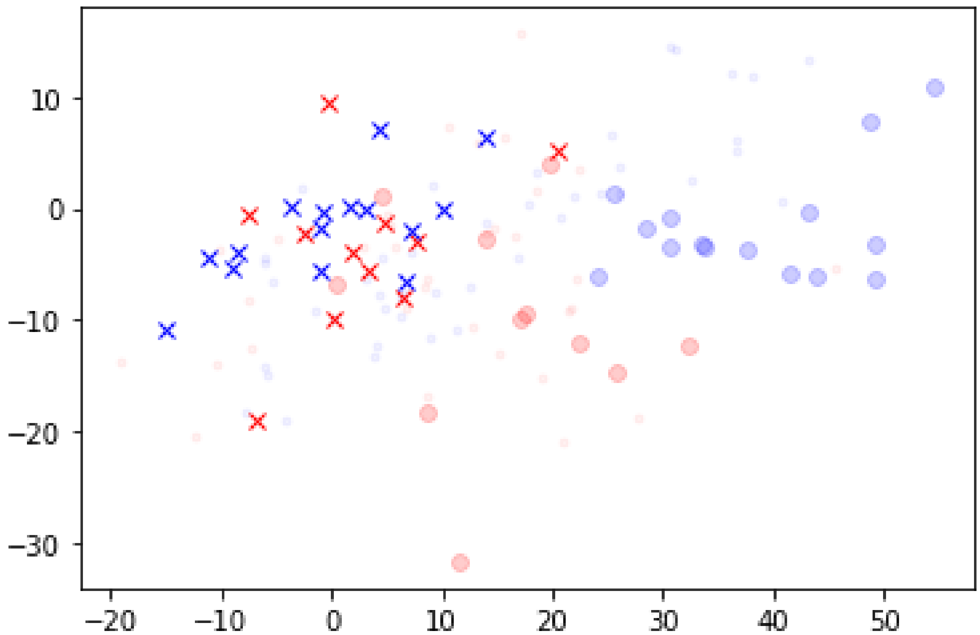
```
<Figure size 432x288 with 0 Axes>
```


15
10
5
0
−5
0
10
20
30
40

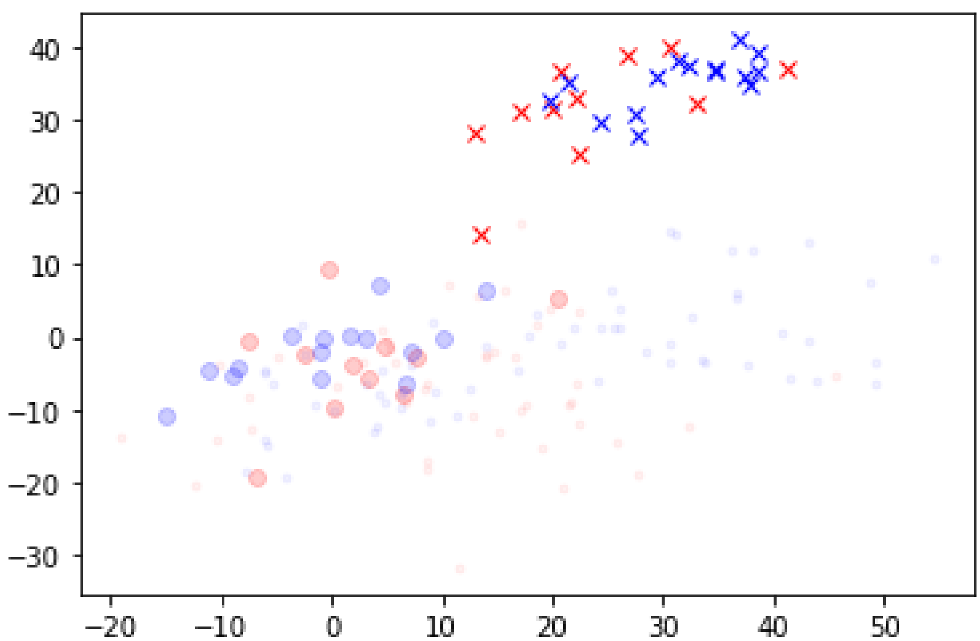

15
10
5
0
−5
−10
−15
−20
−10
0
10
20
30
40


15
10
5
0
−5
−10
−15
−20
−20
−10
0
10
20
30
40

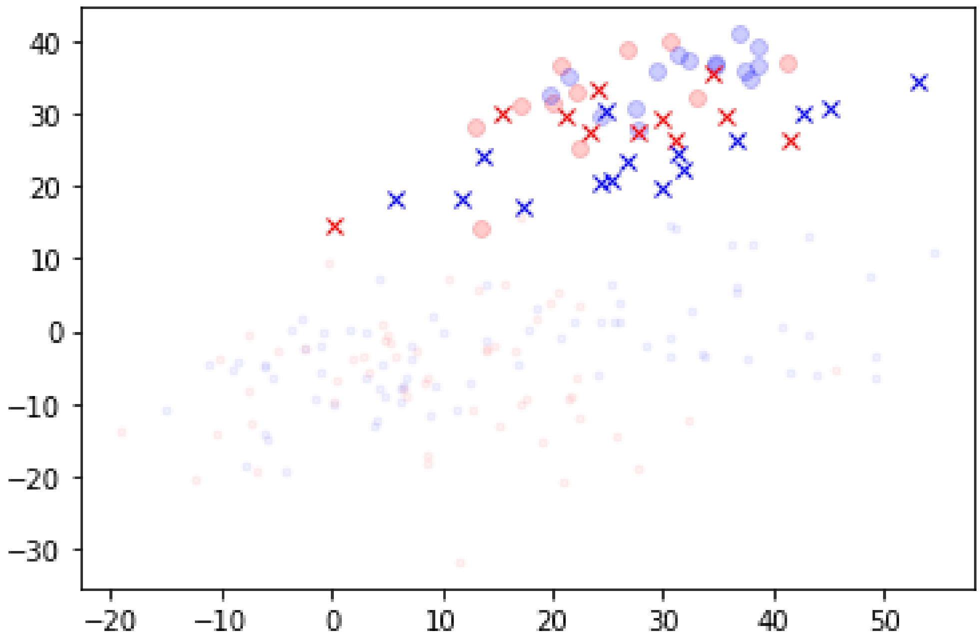

10
0
−10
−20
−30
−20
−10
0
10
20
30
40
50



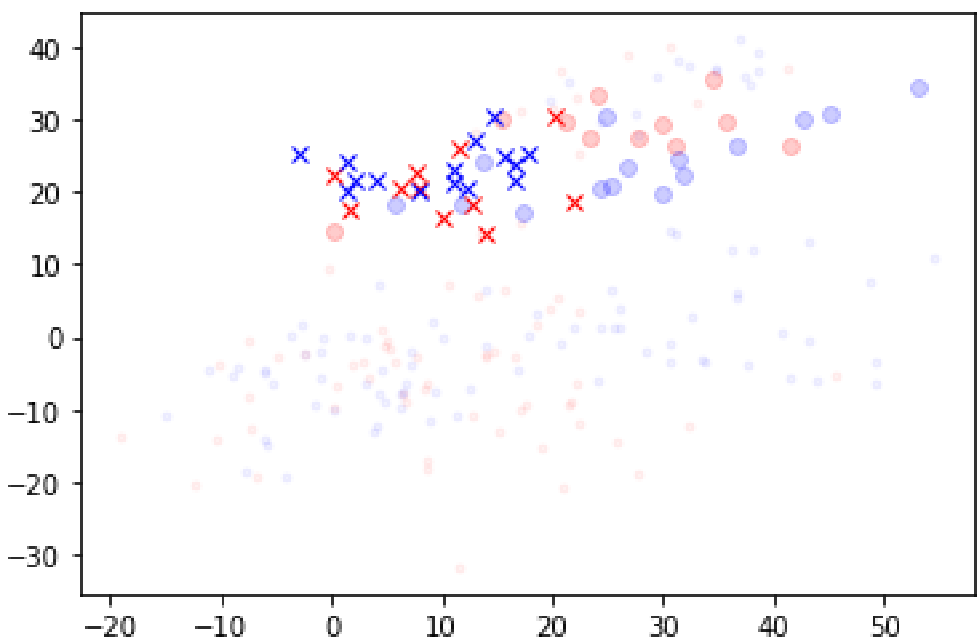
10
0
−10
−20
−30
−20
−10
0
10
20
30
40
50



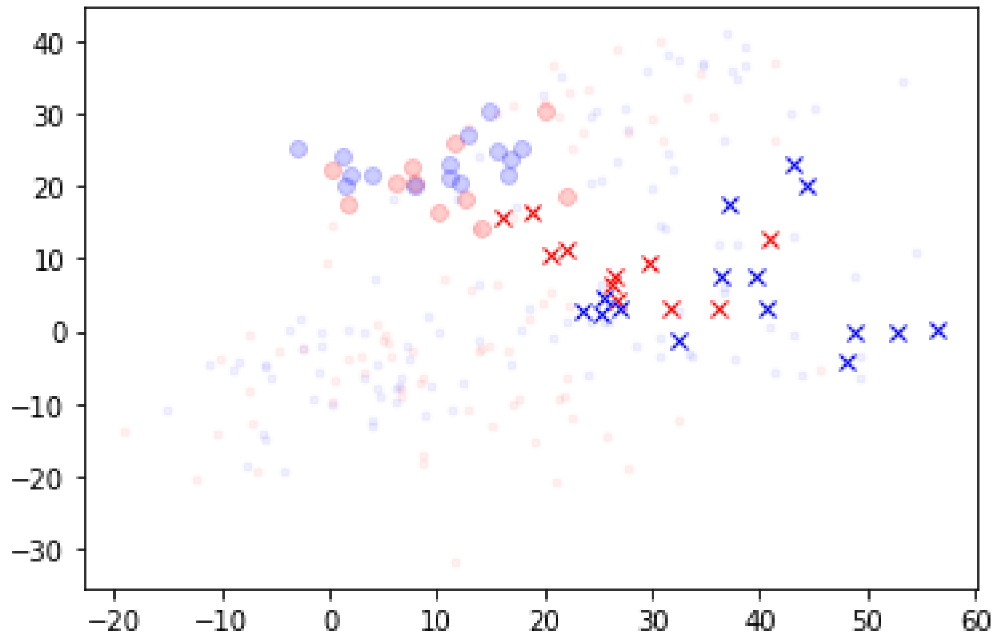
40
30
20
10
0
−10
−20
−30
−20
−10
0
10
20
30
40
50



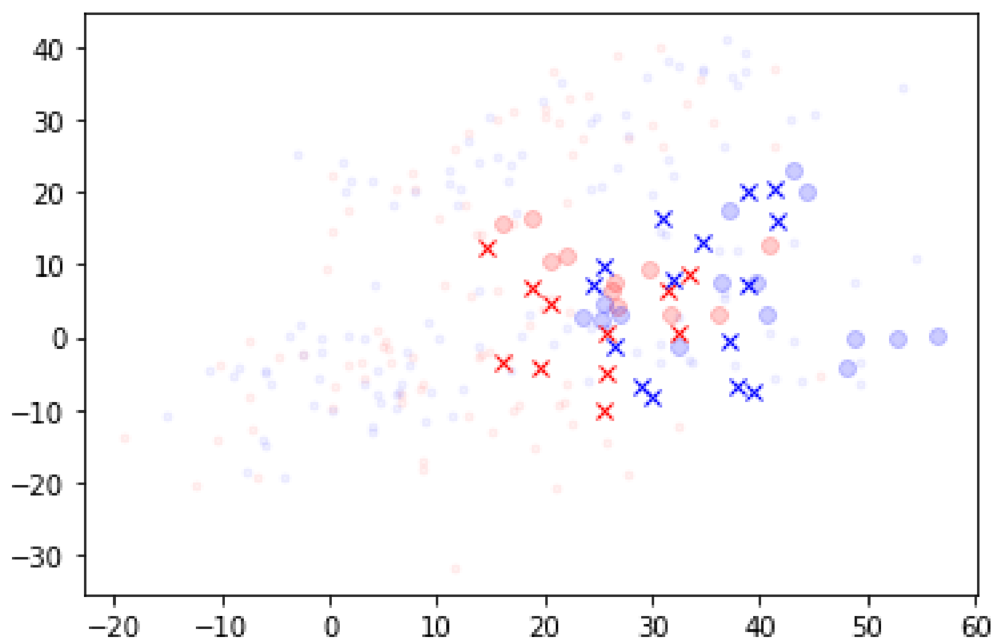
40
30
20
10
0
−10
−20
−30
−20
−10
0
10
20
30
40
50



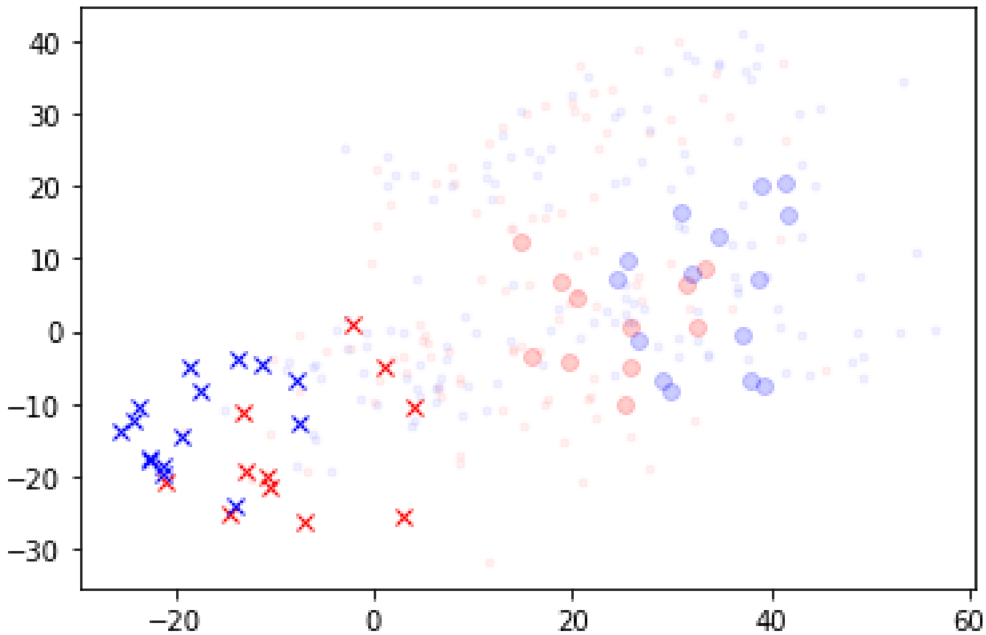
40
30
20
10
0
−10
−20
−30
−20
−10
0
10
20
30
40
50

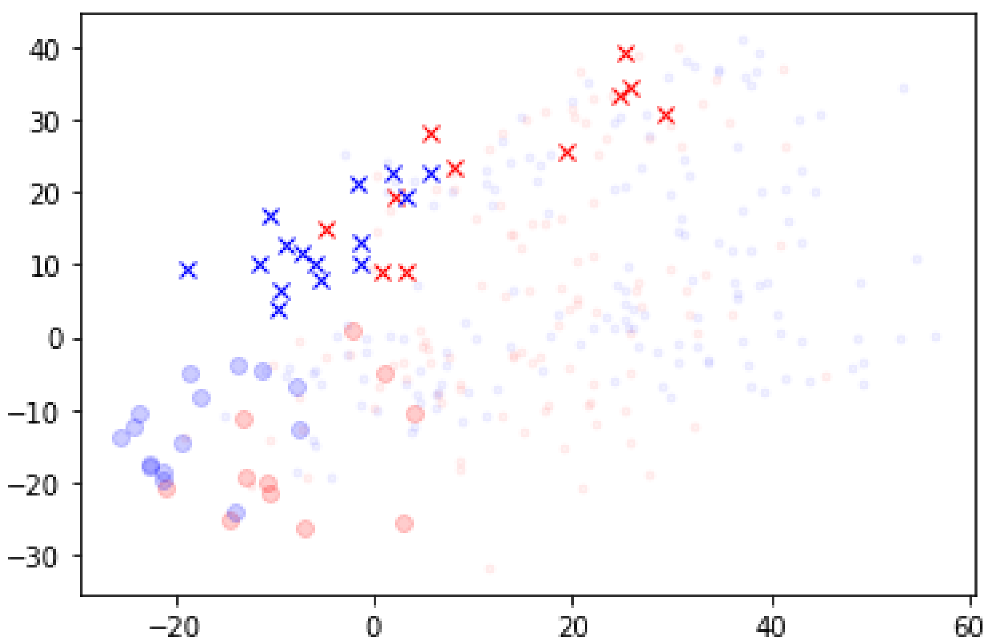


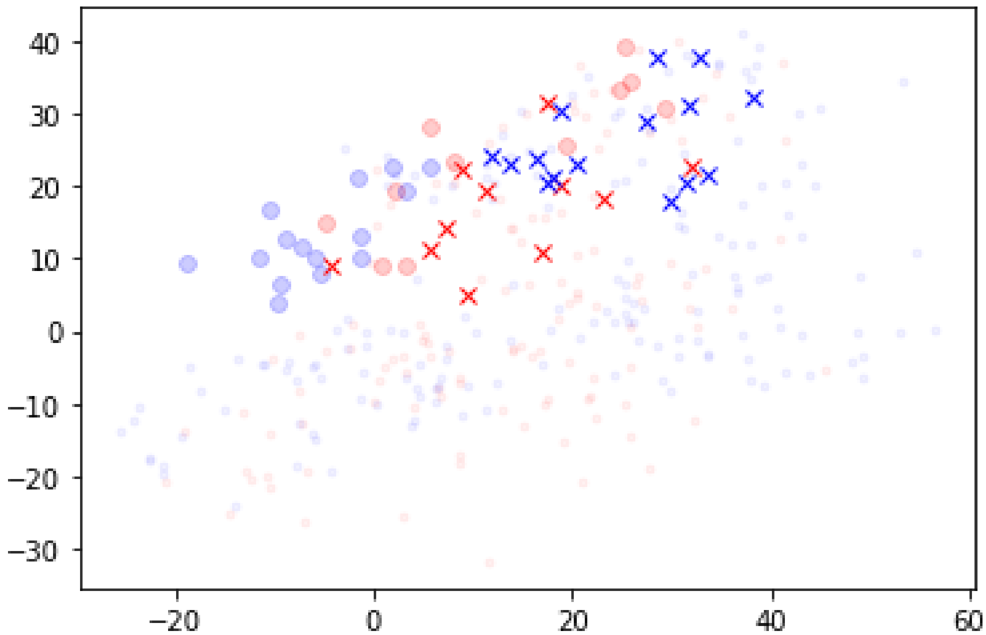
40
30
20
10
0
−10
−20
−30
−20
−10
0
10
20
30
40
50
60



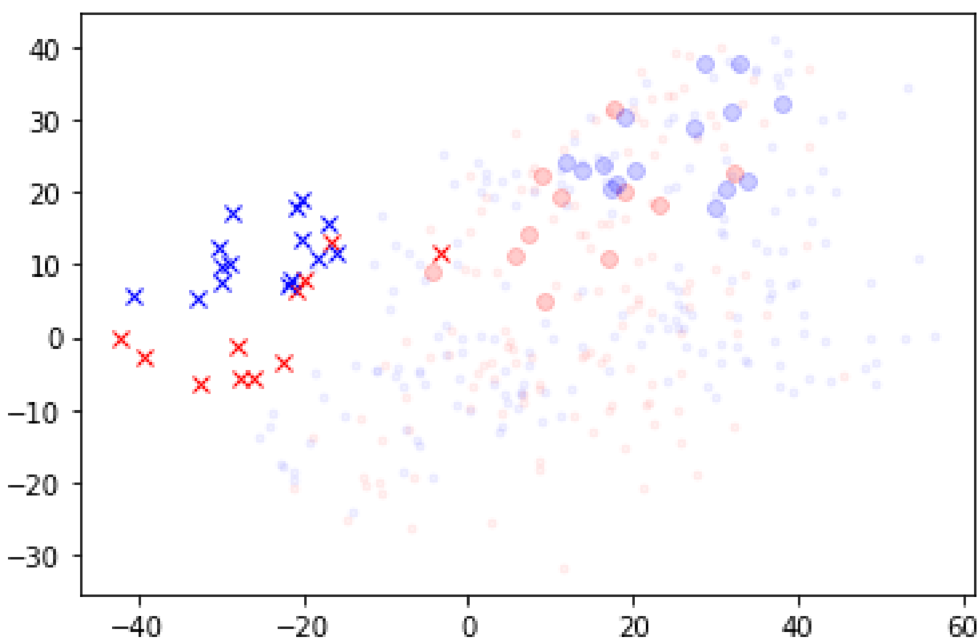
40
30
20
10
0
−10
−20
−30
−20
−10
0
10
20
30
40
50
60


40
30
20
10
0
−10
−20
−30
−20
0
20
40
60

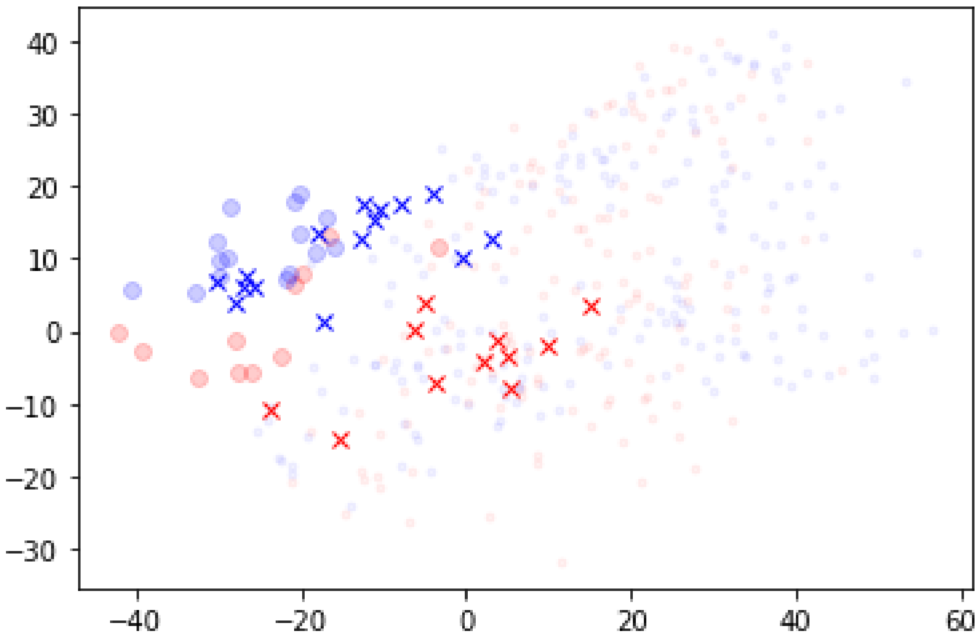

40
30
20
10
0
−10
−20
−30
−20
0
20
40
60

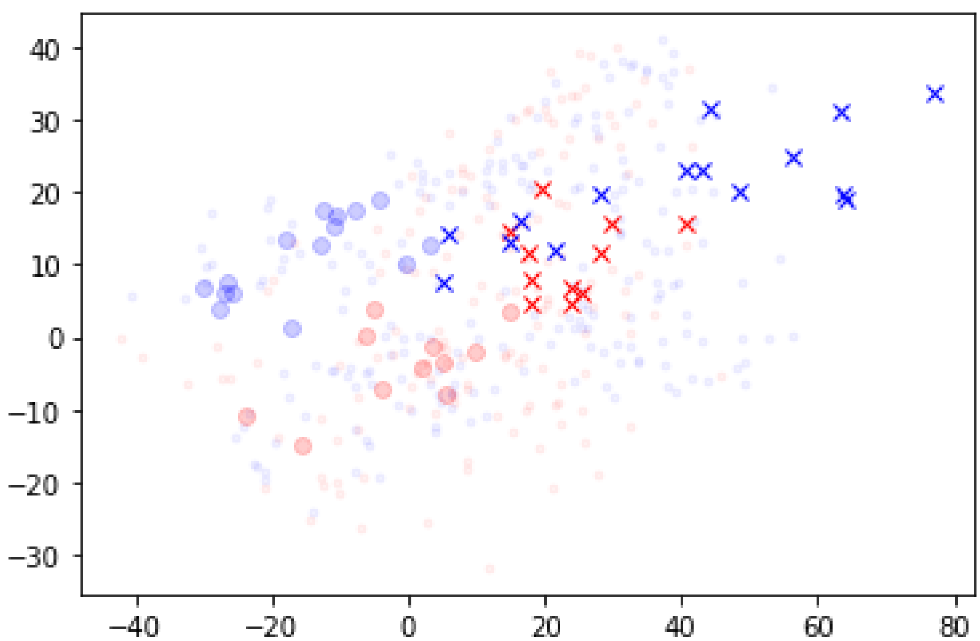


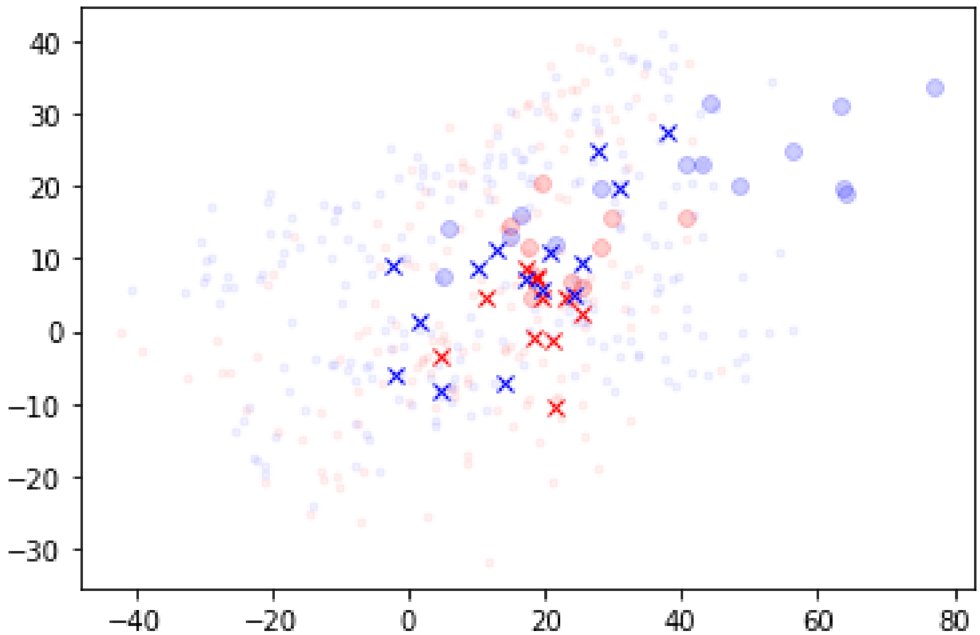
40
30
20
10
0
−10
−20
−30
−20
0
20
40
60



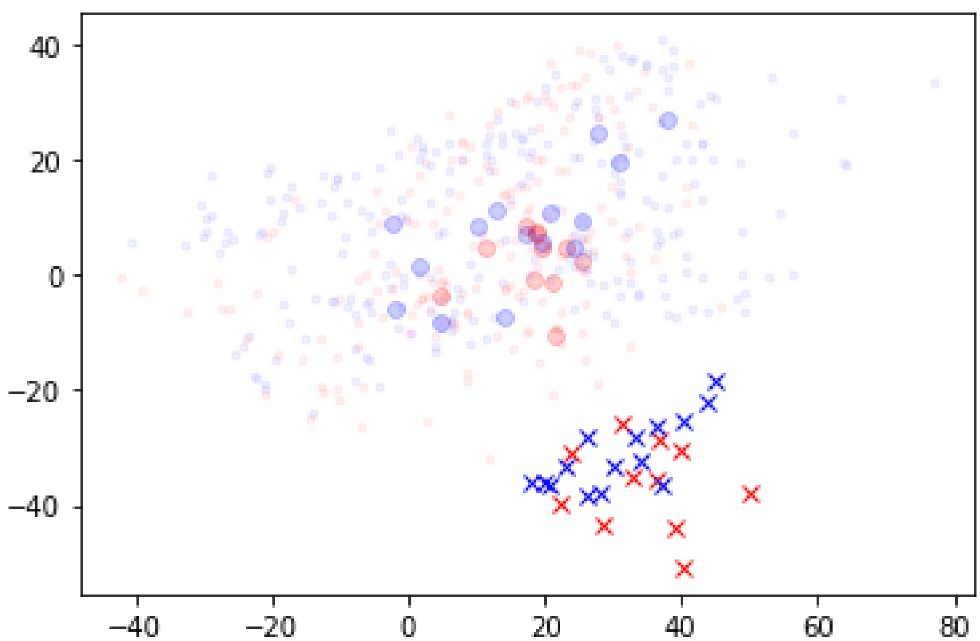
40
30
20
10
0
−10
−20
−30
−40
−20
0
20
40
60


40
30
20
10
0
−10
−20
−30
−40
−20
0
20
40
60

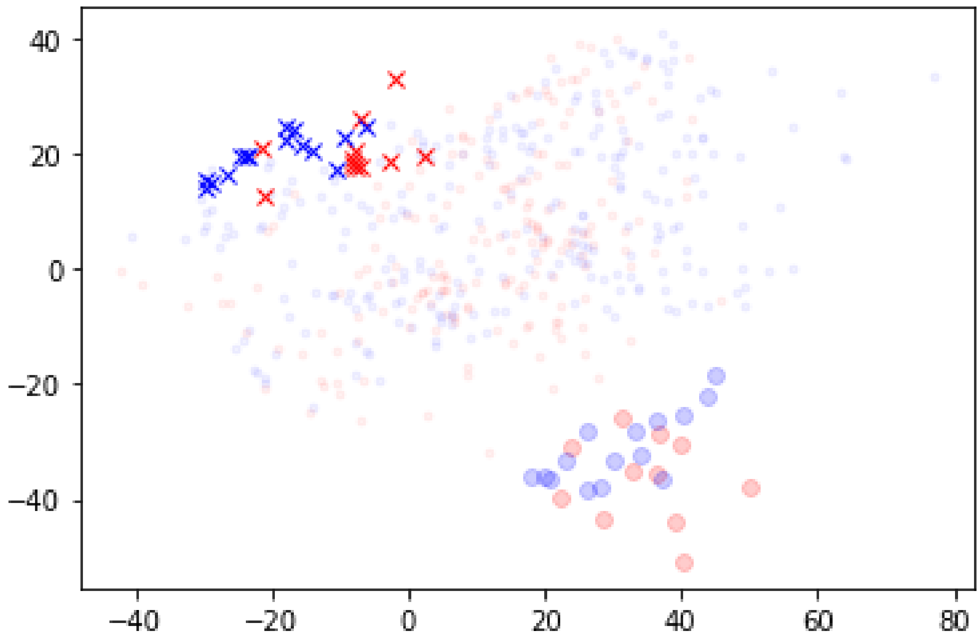

40
30
20
10
0
−10
−20
−30
−40
−20
0
20
40
60
80



40
30
20
10
0
−10
−20
−30
−40
−20
0
20
40
60
80



40
20
0
−20
−40
−40
−20
0
20
40
60
80



Data seems to move "together in a cloud", though consistently slightly separate per-channel-type. Similar results as use of TSNE.