# syn_classification_notebook

January 22, 2018

In [123]:
```
# Synapse Classification Challenge
# Introduction to Connectomics 2017
# W. Gray Roncal (your name here)

your_name = 'ramsden_devin'
```

In [124]:
```
# Load data

import numpy as np

data = np.load('./synchallenge2017_training.npz')

imtrain = data['imtrain']
annotrain = data['annotrain']
ytrain = data['ytrain']

data = np.load('./synchallenge2017_validation.npz')

imvalid = data['imvalid']
annovalid = data['annovalid']
yvalid = data['yvalid']
```

In [125]:
```
# Define feature extraction code

import skimage.feature as skif

def extract_features(imdata):
    xtrain = []
    for im in imdata:
        fvector = []
        # 50th percentile based on intensity
        fvector.append(np.percentile(im,50))

        #corner_kitchen_rosenfeld
        ckr = skif.corner_kitchen_rosenfeld(im)
        ckr = np.ravel(ckr)
        for i in ckr:
```

```
                    fvector.append(i)

                #corner_shi_tomasi
                csh = skif.corner_shi_tomasi(im)
                csh = np.ravel(csh)
                for i in csh:
                    fvector.append(i)

                #structure_tensor
                st = skif.structure_tensor(im)
                st = np.ravel(st)
                for i in st:
                    fvector.append(i)

                # add a contrast feature
                g = skif.greycomatrix(im, [1, 2], [0, np.pi/2],normed=True, symmetric=True)
                homogeneity = skif.greycoprops(g, 'homogeneity')

                # explict way to add feature elements one at a time
                homogeneity = np.ravel(homogeneity)
                for i in homogeneity:
                    fvector.append(i)

                fvector = np.asarray(fvector)
                xtrain.append(fvector)

            return np.asarray(xtrain)
```

In [126]:
```
# Extract Features from training

xtrain = extract_features(imtrain)
```

In [127]:
```
# Train Classifier

from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=200)
clf = clf.fit(xtrain, ytrain)
```

In [128]:
```
# Extract features from validation set
xvalid = extract_features(imvalid)
```

In [129]:
```
# Run Classifier on validation set
scoresvalid = clf.predict_proba(xvalid)
```

In [130]:
```
# Best f1 score report on validation set

from sklearn.metrics import f1_score
```

```
        # Can add post-processing here if desired

        prob_syn = scoresvalid[:,1]

        # default threshold
        print('default f1 score: {}'.format(np.round(f1_score(yvalid, prob_syn >=0.5),2)))

        f1_out = 0
        thresh = 0
        for i in np.arange(0.0, 1, 0.05):
            f1_test =  f1_score(yvalid, prob_syn > i)
            if f1_test > f1_out:
                f1_out = f1_test
                thresh = i

        print('My best validation f1-score is: {} at {} threshold.'.format(np.round(f1_out,2),
```

```
default f1 score: 0.84
My best validation f1-score is: 0.85 at 0.45 threshold.
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1113: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
```

In [131]:
```
        # Changing a parameter
        # Train Classifier

        from sklearn.ensemble import RandomForestClassifier
        clf = RandomForestClassifier(n_estimators=200)
        clf = clf.fit(xtrain, ytrain)

        # Run Classifier on validation set
        scoresvalid = clf.predict_proba(xvalid)

        # Best f1 score report on validation set

        from sklearn.metrics import f1_score

        # Can add post-processing here if desired

        prob_syn = scoresvalid[:,1]

        # default threshold
        print('default f1 score: {}'.format(np.round(f1_score(yvalid, prob_syn >=0.5),2)))

        f1_out = 0
        thresh = 0
```

```
        for i in np.arange(0.0, 1, 0.05):
            f1_test =  f1_score(yvalid, prob_syn > i)
            if f1_test > f1_out:
                f1_out = f1_test
                thresh = i

        print('My best validation f1-score is: {} at {} threshold.'.format(np.round(f1_out,2),
```

```
default f1 score: 0.84
My best validation f1-score is: 0.84 at 0.5 threshold.
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1113: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
```

In [132]:
```
# here we can inspect results

        valid_labels = np.asarray(prob_syn > thresh,dtype='int')
        # find images we did well on
        idx_correct_syn = np.where((valid_labels == yvalid) & (yvalid == 1))[0]
        idx_correct_nosyn = np.where((valid_labels == yvalid) & (yvalid == 0))[0]
        # find images we did poorly on

        idx_wrong_syn = np.where((valid_labels != yvalid) & (yvalid == 1))[0]
        idx_wrong_nosyn = np.where((valid_labels != yvalid) & (yvalid == 0))[0]

        import ndparse as ndp

        print('synapse present - true positive')
        ndp.plot(imvalid[idx_correct_syn[3]])

        print('no synapse present - true negative')
        ndp.plot(imvalid[idx_correct_nosyn[3]])

        print('synapse present - false negative')
        ndp.plot(imvalid[idx_wrong_syn[3]])

        print('no synapse present - false positive')
        ndp.plot(imvalid[idx_wrong_nosyn[3]])
```

```
-------------------------------------------------------------

ModuleNotFoundError        Traceback (most recent call last)

<ipython-input-132-3ab3cf437d5e> in <module>()
 10 idx_wrong_nosyn = np.where((valid_labels != yvalid) & (yvalid == 0))[0]
 11
```

```
---> 12 import ndparse as ndp
     13
     14 print('synapse present - true positive')


ModuleNotFoundError: No module named 'ndparse'
```

In [ ]:
```
# Validate performance on test set (should only run/score once!)

data = np.load('./synchallenge2017_test_notruth.npz')

imtest = data['imtest']
annotest = data['annotest']

# Extract features from test set
xtest = extract_features(imtest)

# Run classifier on test set
scoretest = clf.predict_proba(xvalid)

# Post-processing
prob_syntest = scoretest[:,1]
syntest_predict = prob_syntest > thresh
syntest_predict = np.asarray(syntest_predict,dtype = 'uint8')

# save file and upload to google docs with label vector
np.save(your_name+'_synchallenge_testdata.npy',syntest_predict)
```