# PyNexradML - Main Scripts

**datastore.py**

The datastore script is used to manage the nexrad data available for training. This script parses nexrad radar files (.Z or .gz) and adds them to an hdf5 file called *datastore.h5*. The sweeps in this file contain the reflectivity, radial velocity and spectrum width moments for the 0.5 degree elevation angle sweep. Data is stored in polar format.

Sweeps are further organized into named datasets that can be selected or combined for use in classifier training. Initial construction of the datastore requires the existence of a configuration file named *manifest.cfg*. This file will most likely be written by hand and defines the datasets as well as provides expert classifications for the sweeps being added. A sample manifest.cfg is shown below.

[rd1]

6500KILX20010907\_000028.Z = -1

6500KILX20010908\_000251.Z = 1

6500KILX20010919\_000545.Z = -1

6500KILX20010924\_221013.Z = 1

6500KILX20010925\_221419.Z = 1

6500KILX20010926\_221409.Z = 1

6500KILX20010927\_220927.Z = 1

[rd2]

6500KLIX20020925\_031530.Z = -1

6500KLIX20020926\_021310.Z = -1

6500KLIX20021026\_024415.Z = -1

6500KLIX20050829\_134926.Z = -1

6500KLIX20031013\_025737.Z = 1

6500KLIX20031015\_014940.Z = 1

6500KCRP20050403\_035853.Z = 1

6500KLCH20050420\_040528.Z = 1

This sample file defines two datasets, *rd1* and *rd2,* each containing the sweeps listed below them. The manifest.cfg file should be located in the same directory as the data being imported.

**Building:**

Initially building or rebuilding the datastore can be accomplished by running the datastore script with the -r argument as shown below, where D:\Data would be replaced by the location of your data.

>python datastore.py -r -d D:\Data

**Importing:**

Importing new data is equally straight forward. Simply put the new data into its own directory and create a new manifest.cfg file containing the classifications and datasets for the new data, placing it in the same directory. Importing is performed with the *-i* argument:

>python datastore.py -d D:\Data -i D:\new\_data

Other notable command line options include *--show\_datasets* and *--desc\_dataset* which are used to display the current datasets in the datastore or sweeps in a dataset respectively.

**trainer.py**

The trainer script is used to train and validate neural network classifers. The trainer script supports the following arguments.

-a, --arch This option is used to specify the network architecture for the neural network. Architecture is specified as a comma separated list of layer sizes. The first value should always be the number of inputs and last value should always be 1. Middle values are used to specify the number of nodes in the hidden layers. If unspecified, the default architecture is x,x/2,1 where x is the number of inputs.

--cache The cache command tells the trainer script to use a disk cache for the training and validation data. Caching the data to disk will generally cause the trainer script to run more slowly, but is useful if the amount of data after feature creation and filtering is greater than the amount that can fit into main memory. In this situation, using a cache allows the script to run without raising an out of memory exception.

-d, --data\_dir The -d/--data\_dir option is used to specify the location of the datastore (datastore.h5)

--epochs Specify the number of epochs (presentations of data) to use when training the network. A larger number may lead to better training, but will require more computation time.

-f. --config\_file Specify a configuration file other than the default (pynexrad.cfg). Most command line options can also be specified in a config file. This can reduce the number of commands that must be specified if there are commands that rarely change. An example config file is shown at the end of this section.

--features Specify the features to train on (defined in features.py). Individual features are described further down.

--filters Specify the filters to apply to the data. Filters can either filter out data or they can modify data. Individual filters are described below.

--norm Specify any normalization to perform on the selected features. Different normalizers are described below.

-t Specify the dataset(s) to use for classifier training.

-o Specify the output name for the classifier. For example -o myclassifier will result in myclassifier.net and myclassifier.proc files being created. These files can be subsequently used to classify new data.

**Features:**

Reflectivity() The reflectivity moment for the base NEXRAD data in dBz

Velocity() The radial velocity moment for the base NEXRAD data in m/s

SpectrumWidth() The spectrum width moment for the base NEXRAD data

Range() The range of a pulse volume in meters

Variance(BaseFeature,Window=3) This feature calculates the variance for the specified base feature using a [*window* x *window*] neighborhood. The default window size is [3x3].

Kurtosis(BaseFeature,Window=3) This feature calculates the kurtosis for the specified base feature using a [*window x window*] neighborhood. The default window size is [3x3].

Skew(BaseFeature,Window=3)

This feature calculates the skew for the specified base feature using a [*window x window*] neighborhood. The default window size is [3x3].

**Filters:**

RangeConstraints(min=20000,max=145000)

The RangeConstraints() filter removes all pulse volumes with range below the specified min or above the specified max. Ranges are specified in meters. The default min and max are 20km and 145km respectively.

RemoveBadValues(BaseFeature,BadValues=True,RangeFolded=True)

This filter removes pulse volumes in which the value of the specified base feature is corrupt. By default all bad or range folded values are considered corrupt, where 'bad' indicates the raw value was less than the signal to noise ratio.

SmootheBadValues(BaseFeature,BadValues=True,RangeFolded=True)

This filter is very similar to the RemoveBadValues filter except that pulse volumes containing offending values are modified rather than removed. Corrupt values are replaced with the average value of the base feature for the current datasets.

SubSample(Percent) The SubSample filter randomly removes pulse volumes until the resulting dataset is the specified percent of the original dataset.

For example SubSample(0.05) indicates that a user only wants to consider 5% of the original data. This is useful for reducing computation time on large datasets.

**Normalizers :**

SymmetricNormalizer(\*columns) Adjusted specified columns so they have a mean of zero and unit variance. Columns are zero base indexed. For example, to normalize the first three features, specify the following on the command line:

--norm SymmetricNormalizer(0,1,2)

**Examples:**

>python trainer.py --cache -d D:\Data --epochs 100 --features Reflectivity() Velocity() Range() Variance(Velocity) --filters RangeConstraints() RemoveBadValues(Reflectivity) SmootheBadValues(Velocity) SubSample(0.1) --norm SymmetricNormalizer(0,1,2,3) -t dataset1 dataset2 -o myclassifier

>python trainer.py --arch 3,6,1 -d D:\Data --epochs 10 --features Reflectivity()Variance(Reflectivity,25)Skew(Reflectivity,25) --filters RangeConstraints(10000,50000) RemoveBadValues(Reflectivity,True,False) --norm SymmetricNormalizer(0,1,2) -t dataset1 -o myclassifier

**Reasonable Baseline**

python trainer.py -d D:\Data --epochs 100 --features Reflectivity()Velocity() SpectrumWidth() Range() --filters RangeConstraints() RemoveBadValues(Reflectivity) SmootheBadValues(Velocity) SmootheBadValues(SpectrumWidth) --norm SymmetricNormalizer(0,1,2,3) -t rd2 -o baseline

**\*NOTE:** All parameters must be specified without spaces, e.g. RangeConstraints(10000,50000) **NOT** RangeContstraints(10000, 50000).

**Configuration Files:**

Configuration files can be used to specify command line options that rarely change. This reduces the amount of information that must be entered at the command line. Configuration settings can also be used as defaults, being overridden when the same option is specified on the command line. Most command line options can also be specified in a configuration file. By default, the trainer script looks for the existence of the pynexrad.cfg file and uses it if it exists. Configuration lines beginning with semi colons are considered comments. An example configuration file is shown below:

[trainer]

;Data directory containing the datastore.h5 file

data\_dir = D:/Data/Data

epochs = 100

features = Reflectivity() Velocity() Range()

filters = RangeConstraints() RemoveBadValues(Reflectivity)

;Normalize all feature columns to have a zero

;mean and unit variance

norm = SymmetricNormalizer(0,1,2)

training\_data = dataset1

**screen.py**

The screen script can be used to screen new data for the presence of biological data. A classifier, that has previously been constructed using the trainer script, is applied to the new data and a message is printed indicating whether the sweep matches or does not match the criteria specified.

-d, --data\_dir The -d/--data\_dir option is used to specify the location of the new data to screen. New data is expected to be in the original level II data format (\*.Z/\*.gz)

-t Specifies a threshold for classification which serves as the classification criteria. This threshold is a target ratio or biological to nonbiological pulse volumes.

-i Specify the classifier to use (e.g. myclassifier).

**Examples:**

>python screen.py -i myclassifier -d D:\new\_data -t 0.7

## PyNexradMl on GitHub

This project can be downloaded via git by cloning the repository at https://github.com/reggiemead/py-nexrad-ml.git. This can be accomplished from the command line with the following command:

git clone https://github.com/reggiemead/py-nexrad-ml.git pynexradml