# Unpacking the Dataset

The Kaggle Seizure Prediction Dataset is distributed as a set of seven gzip compressed files with a total compressed size of about 59.6GB and an uncompressed size of about 124GB. The program 7-zip can be used to decompress these files into subdirectories of a single directory as illustrated in the table below. The base directory can have any name, but the patient subdirectories should retain the specified names. Following decompression, each of the dog and patient subdirectories should contain on the order of one thousand data files in Matlab .mat format. Each of these files represents a single instance of Electroencephalographic (EEG) sensor data collected from the epileptic patient (dog or human) over a 10 minute period. The number of sensors used varies by patient over the range from 15 to 24. For dogs, the sensors were sampled at 400Hz while for humans the sensors were sampled at 5000Hz. Each filename contains one of three text strings: “preictal”, “interictal”, or “test”. The precital and interictal files are labeled training data. For preictal files the patient had a seizure within one hour following collection of the data, while for interictal files the patient had no seizure in the same time frame. The test files are unlabeled contest data for which competitors must submit a probability-like classification score in which lower numerical values represent interictal-like data and higher values-represent preictial-like data. The contest scoring algorithm computes an area under the curve (AUC) for the receiver operating characteristic (ROC) curve drawn using the submitted scores and the actual class targets for the test data. In order to prevent contestants from training using the contest scoring system, the scoring system partitions the test data into subsets and provides scores for one subset while deciding the contest on the other.

|  |  |
| --- | --- |
| Filename | Unzipped Location |
| Dog\_1.gz | Base/Dog\_1/\*.mat |
| Dog\_2.gz | Base/Dog\_2/\*.mat |
| Dog\_3.gz | Base/Dog\_3/\*.mat |
| Dog\_4.gz | Base/Dog\_4/\*.mat |
| Dog\_5.gz | Base/Dog\_5/\*.mat |
| Patient\_1.gz | Base/Patient\_2/\*.mat |
| Patient\_2.gz | Base/Patient\_2/\*.mat |

# Preparing the Computer

The software requires 64 bit python 3.3 or newer with the numpy, scipy, and optionally the pandas packages installed. I have run the software in two different environments: Windows 8.1 with 16GB of RAM using the CPU only, and Ubuntu Linux with 1GB RAM and GPU card. The following sections summarize my approach to configuring these environments. Working with this dataset is extremely memory intensive and 1GB is completely inadequate for processing more than small test cases. Working under Windows with 16GB was not entirely adequate either, and a computer well-matched to this task should have a fast CPU, a high end NVIDIA GPU with 4GB or more of memory, and 32-64GB of main memory in order to load and process complete patient datasets in memory.

## Windows CPU Operation

Anaconda Python from Continuum Analytics is free 64 bit python 3.3 installation for Windows with numpy, scipy, and pandas packages included. It provides a windows installer and is very simple to use.

<https://store.continuum.io/cshop/anaconda/>

This is the best source of 64 bit python for the Windows environment that I was able to find, and it installs all of the packages need to run the seizure prediction software in a CPU only configuration.

## Linux GPU Operation

To run the software in a GPU configuration, the following prerequisites must be installed:

* 64 bit python 3.3 with numpy, scipy, and pandas packages
* NVIDIA CUDA development tools and GPU driver
* Cudamat

The NVIDIA driver is not compatible with all Linux distributions and versions, so it is wise to choose a Linux version that is supported. I used Ubuntu 14.04 as it was the most recent Ubuntu version supported by the most recent NVIDIA tools.

### Installing Linux

Download Ubuntu 14.04 ISO and burn the image to DVD

Reboot the computer with the DVD in the drive and select boot from CD.

Allow the installation to erase all other installations.

### Installing Python

Install useful packages for python3:

* sudo apt-get install python3-numpy
* sudo apt-get install python3-scipy
* sudo apt-get install pandas

### Enabling SSH (Optional)

Install ssh-server to allow remote login:

* sudo apt-get install openssh-server
* sudo /etc/init.d/ssh restart

### Installing CUDA

Download the CUDA 6.5 .deb file:

* cuda-repo-ubuntu1404\_6.5-14\_amd64.deb

Install CUDA:

* sudo dpkg -i cuda-repo-<distro>\_<version>\_<architecture>.deb
* sudo apt-get update
* sudo apt-get install cuda

Edit .bashrc to update environment variables (note that .bash\_profile might actually be the correct place to put these):

* export PATH=/usr/local/cuda-6.5/bin:$PATH
* export LD\_LIBRARY\_PATH=/usr/local/cuda-6.5/lib64:$LD\_LIBRARY\_PATH

### Installing cudamat

Make ~/ML directory

Get and make cudamat:

* cd ~/ML
* git clone <https://github.com/cudamat/cudamat.git>
* cd cudamat
* make
* python3 test\_cudamat.py
* python3 test\_learn.py

Edit .bashrc to update environment variables:

* export PYTHONPATH=$PYTHONPATH:/home/mark/ML/cudamat

# Preparing a Contest Submission

For CPU operation, only three files are required:

|  |  |
| --- | --- |
| Filename | Unzipped Location |
| Python/PrepareContestSubmission.py | Performs the overall training and testing procedure. |
| Python/RbmStack.py | Implements the Restricted Boltzmann Machine class used to initialize network weights to sensible values. |
| Python/SequenceDecimatingNetwork.py | Implements the sequence decimating network. |

The software is presently configured to perform the necessary steps prepare a contest submission starting with the raw contest data. From a python command prompt with the three files in the python path, the process of preparing a submission for the Kaggle contest can be initiated in the following manner:

>>> import PrepareContestSubmission as pcs

>>> pcs.Go(sDatasetPath=’Base’, rSampleFrequency=’20’, tlGeometry= [(16,128),(2,128),(2,128),(2,128),(2,1)])

The software will iterate through each patient dataset invoking the necessary operations to train a classifier and classify the test data for contest submission. When all patients have been processed, the script will consolidate test data predictions into a single “Upload.csv” file in the sDatasetPath directory for upload to the contest scoring web page located here.

<http://www.kaggle.com/c/seizure-prediction/submissions/attach>

Unfortunately, the contest is presently closed and the scoring page seems to only be available to those who registered to participate during the enterable phase of the competition. The following sections provide a synopsis of the processing performed by the Go script to prepare the contest submission.

## Shuffling

If the patient directory does not contain a shuffle.csv file, create one and write to it a randomly shuffled list of all .mat filenames in the patient directory. This shuffling of the filenames can be used to select training / validation splits consistently over multiple training runs and various holdout fractions.

## Preprocessing

Preprocessing creates a subdirectory of the patient directory corresponding to the specified sample frequency. For each .mat file in the patient directory, it reads the .mat file, and for each sensor in the data it decimates the samples to the specified sample frequency, normalizes them to a constant standard deviation, a mean of .5, and values between 0 and 1. The preprocessor stores the preprocessed data in the sample frequency subdirectory as a python pickle file. Preprocessing of all patients can take several hours. If the preprocessor output files already exist for the specified sample frequency, the preprocessor skips them rather than recreating them.

The preprocessor can optionally perform linear de-trending of the data, but this option is currently disabled.

## Unsupervised Pre-training

Unsupervised pre-training establishes initial parameter values for the network parameters of the sequence decimating network by training a sequence of restricted Boltzmann machines in the following way:

Start with an empty list of sequence decimating network layer parameters. For each layer in the sequence decimating network:

* Instantiate a temporary sequence decimating network using pre-trained layer information for all subordinate layers (initially none).
* Generate input layer training samples drawn from random patient training files at random time offsets.
* Transform these input layer samples into training samples for the current layer to be trained by passing them through the temporary sequence decimating network.
* Instantiate a single layer restricted Boltzmann machine.
* Train the restricted Boltzmann machine using the transformed input samples. The training algorithm implements weight decay, stochastic sampling of the hidden layer, and dropout of both the visible and hidden layers. Stochastic sampling and dropout are currently disabled, and some experiments indicated that they may not be working correctly.
* Extract the network parameters from the trained restricted Boltzmann machine and add them to the sequence decimating network pre-trained network layer parameters (layer stack).
* Test a sequence decimating network using the current layer stack in an auto-encoder configuration to determine whether the training is performing reasonably.

This procedure builds up an initial weight set for the sequence decimating network that is pre-trained for use as a dimensionality reducing auto-encoder. Both the network parameters (layer stack) and an instance of the sequence decimating network are currently stored as pickle files for use during the supervised training. These files have a names which encode the network geometry with which they were created. The pre-trained model files are stored in the sample frequency directory.

By default, the pre-training procedure does not proceed with pre-training when the pre-trained model files already exist. The bRetrain flag can be used to force regeneration of the pre-training data.

## Supervised Training

Supervised training begins with a pre-trained sequence decimating network created during the pre-training phase. It uses batch stochastic gradient descent with an analytically determined gradient based on back-propagation. The training algorithm also implements momentum and weight decay. The algorithm measures root mean squared error with respect to the class target during training. Constant values near 0.5 indicate that the network is not effectively predicting the class. Unfortunately the network generally converges to this ineffective state. The desired result is something consistently less than 0.5.

## Assessment

Assessment applies the trained sequence decimating network to the training and validation data to plot an ROC curve and measure the area under the curve.

## Test

Test applies the trained sequence decimating network to the test data to generate scores for contest submission. The test scores are stored in the sample frequency subdirectory with the name Upload.csv.