Predicting region specific atrophy levels in the brain due to Huntington's disease through gene expression data

Alexandra Browne
BSc Computer Science
Submission Date: 26th April 2013
Supervisor: Dr Kevin Bryson

Abstract

Huntington's disease (HD) is a late-onset, inherited neurodegenerative disorder characterized through cognitive decline affecting movement, mood and behaviour. MRI imaging reveals region specific atrophy before onset of the clinical manifestations associated with HD. The mechanism by which cell death occurs is not fully understood. Nor why some regions are more sensitive than others. In this project we attempt to determine whether atrophy level is related to the genes being expressed by the different neural tissues. To test this hypothesis we will try to predict the spatial and temporal pattern of atrophy using only the spatial gene expression features.

MRI imaging data sets for R6/2 and control mice allowed us to determine and analyse the spatial occurrence of brain tissue atrophy in HD over time. Support Vector Machines (SVMs) were trained using gene expression data for the adult mouse brain from the Allen Brain Atlas (ABA). These generated models were then used to assign unseen sets of expression data with a predicted atrophy level.

The results are exciting and indicate a strong relationship between atrophy and gene expression levels with all of our selected time points agreeing on four out of five top genes used to predict atrophy.

With our machine learning techniques validated through excellent accuracies on both cross validation and subsequent testing on unseen data, more weight is given to our hypothesis that gene expression levels are significant in determining atrophy attributed to HD.

Future work would involve delving further into the biology of these genes and conducting a detailed literature review to determine if there is more evidence suggesting that the genes our machine learning models have detected, and their expression levels, are in fact relevant in atrophication of the different neural tissues.

Table of Figures

Contents

1 Introduction

Huntington's disease (HD) is a late-onset, inherited neurodegenerative disorder characterized through cognitive decline affecting movement and cognition. Those suffering will also exhibit psychiatric as well as behavioural problems. The disease is progressive and non-reversible with symptoms worsening over a period of ten to twenty years (NHS 2012). Death is often the result of a secondary cause.

There exists a mouse model of this disease called the R6/2 mouse model – which mimics the neurodegeneration of the human disease within the mouse brain. MRI imaging of those suffering from HD reveals region specific atrophy before onset of the clinical manifestations associated with HD (Aggarwal et al. 2012).

MRI data sets exist for control and R6/2 mice (Aggarwal et al. 2012; Zhang et al. 2010) revealing the disease causes atrophy within specific regions of the brain, increasing over time, with the striatum the most affected in the early stages of the disease. Thus this MRI data allows one to determine and analyse the spatial occurrence of brain tissue atrophy in HD over time.

Although the death of brain tissue is attributed to the presence of a mutated form of the huntingtin gene, the mechanism by which cell death occurs is not fully understood. Nor why some regions are more sensitive than others. This said, three key models of brain atrophy within neurodegenerative diseases can be postulated:

1. The atrophy is entirely dependent on the tissue, with each region of the brain being independent of other regions.

2. The atrophy starts in one region of the brain and then propagates to other regions similarly to some amyloid diseases such as prion diseases. Hence the temporal model would include this spatial spreading.

3. The atrophy starts in one region of the brain and then propagates along fibres of neurons to other regions of the brain.

In this project we attempt to determine whether atrophy level is related to the genes being expressed by the different neural tissues. Establishing such a relationship could uncover genes which were not previously identified to contribute to tissue atrophy in HD. Looking much further into the future, identifying said genes could to lead to improved characterization of HD phenotypes.

In testing this hypothesis we utilize two valuable but disconnected data sets. The first are MRI imaging sets for eight control wild-type and seven R6/2 mice (Aggarwal et al. 2012; Zhang et al. 2010). The second is the Allen Brain Atlas (ABA) map of gene expression within the adult mouse brain (Lein et al. 2007; Jones et al. 2009). This dataset is based on profiling around 20,000 genes in slices of the brain and then rendering them in 3D.

Difficulties with research into such a hypothesis arise from the non-uniform representation of relevant information. Data sets, particularly the ABA are vast; the magnitude of information highlights the limitations of various programming languages in processing the data and transforming it into a more workable form. Thus in this project we also intend to deliver a software package which allows the ABA to be read and processed. In addition to allow further extension of our research, we also aim to provide appropriate machine learning models in the form of SVM classifiers which can predict the time-course atrophy of particular regions of the brain, based entirely on the gene expression levels within that region.

This project report is presented as a progression of our research, difficulties encountered along the way and design decisions.

Chapter 2 delves further into the underpinnings of the scientific concepts presented in this chapter and highlights the tools and sources we drew upon.

Chapter 3 looks to uncover differences between appropriate machine learning techniques and their application in terms of our project, focusing on SVMs.

Chapter 4 discusses our first approaches to tackling our hypothesis along with original deliverables and an overview of changes to these during the project's duration to arrive at our final methodology.

Chapter 5 summarises our results and evaluates our techniques, as well as restrictions and constraints which limited the scope of our project.

.

2 Background Information

2.1 Scientific Concepts

2.1.1 The Huntingtin Gene

Huntington's disease, first described in 1872 by Dr George Huntington, is known as a tri-nucleotide repeat disorder. The Huntingtin (Htt) gene contains a section of a repeated nucleotide sequence, CAG. Cytosine-adenine-guanine or CAG is responsible for the amino acid glutamine. Genomes in individuals not suffering from HD show between seven and thirty-five repeats of this nucleotide sequence. Beyond thirty-five, the resulting poly-CAG expansion becomes unstable giving rise to a mutant form (mHtt) of the protein. Normally symptoms develop between the ages of 30-55, with the length of the poly-glutamine stretch inversely proportional to the age of onset (Landles & Bates, 2004).

2.1.2 Neurodegeneration & Associated Symptoms

Although the function of Htt remains unclear, its absence in the mouse genome results in death early on in the embryonic stage, highlighting its importance in cell survival and normal brain development (Bates & Murphy, 2002). In the early stages of HD pathogenesis MRI scans indicate atrophy within the striatum due to the toxic function of mHtt. The striatum's cortical connections provide input to the basal ganglia system, atrophication of the striatal tissue upsets the balance of inhibitory signals with hyperkinetic involuntary movements or chorea as a result.

2.1.3 The R6/2 Mouse Model

Several animal models exist to characterize the pathology of HD and mimic the neurodegeneration of the human disease. The R6/2 transgenic mice strain is one such model. With a CAG repeat of ~125, within 6-8 weeks of age the mouse begins to develop visible degradation in cognitive function

(Charles River Laboratories, 2013). The mouse genome contains its own huntingtin gene homologous to the human variant with a genetic similarity of 81% (HOPES, 2010). Fewer repeats of the CAG nucleotide sequence in the mouse Htt gene explain the absence of natural models of HD within wild mice. This necessitates an artificial model where the mouse genome has been complemented with the human mHtt.

Due to the nature of transgenic techniques the mouse will express both its normal Htt gene and the newly introduced mHtt gene. The R6/2 transgenic mouse is one of the most commonly used animal models for both short and longitudinal studies. In using such a model to elucidate the mechanisms of HD induced atrophy, of great importance is to recognise the differences in the structure between the mouse and human brain, as well as variances in gene expression. The study by Miler et al. (2010) highlights the appropriateness of this model stating "brain transcriptome are highly preserved between species" with "all modules of highly co-expressed genes identified in mouse also identified in humans".

The availability of data sets and the vast amount of research based upon this model of HD in addition to these gene expression similarities made the R6/2 mouse the model of choice for our study.

2.1.4 Comparing Human & Mouse Brain Structures

In MRI imaging data sets, atrophy of the striatal region is detectable before onset of clinical manifestations, more specifically as "early as 15 years before disease onset (for the human disease) and continues throughout the period of manifest illness" (Roze, E. et al., 2011). The atrophication of the striatal region in the early stages is the case in both the diseased human and R6/2 mouse brain. **Figure 1, Figure 2, Figure 3** and **Figure 4** compare the structure of the striatum for a healthy human and healthy C57BL/6 mouse brain as viewed in The Allen Brain Atlas[1] (ABA) Brain Explorer® 2 software[2].

[1] See section 2.2.1.
[2] http://mouse.brain-map.org/static/brainexplorer

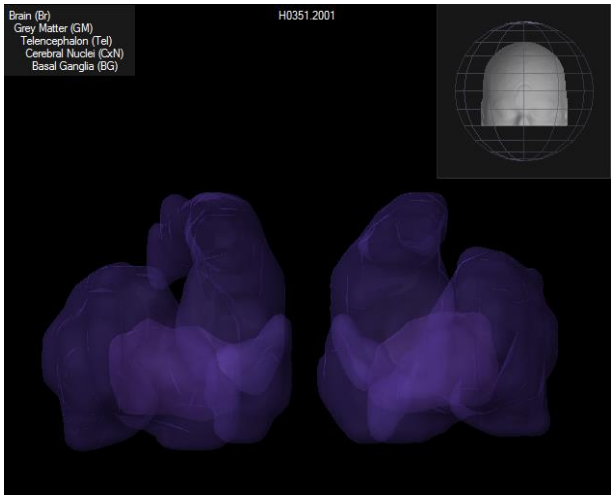


Figure 1 Anterior transparent view of human striatal region composed of the caudate nucleus, nucleus accumbens and the putaman as viewed within the basal ganglia along with the globus pallidus

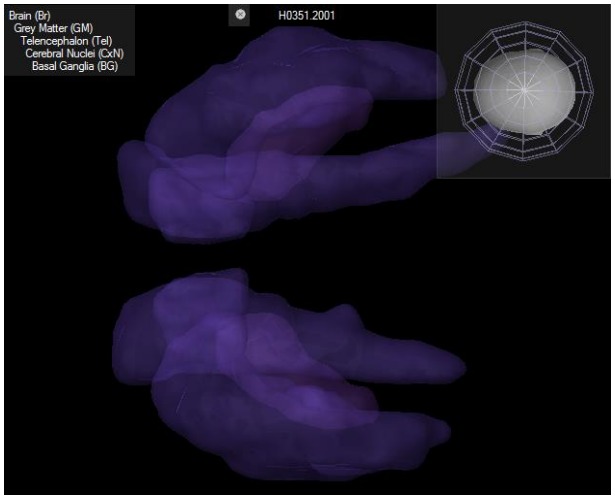


Figure 2 Superior transparent view of human striatal region composed of the caudate nucleus, nucleus accumbens and the putaman as viewed within the basal ganglia along with the globus pallidus

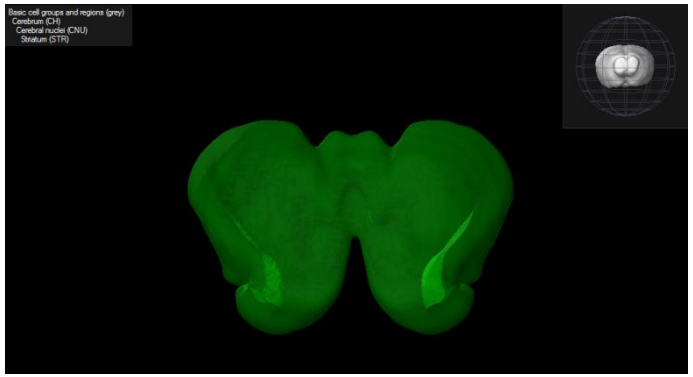


Figure 3 Anterior transparent view of mouse striatal region composed of the caudate nucleus, nucleus accumbens and the putaman as viewed within the basal ganglia along with the globus pallidus

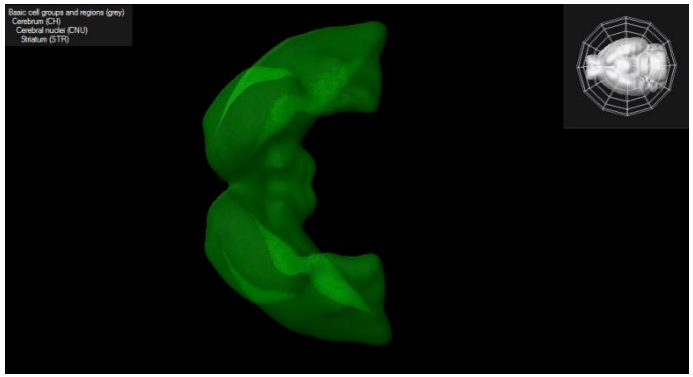


Figure 4 Superior transparent view of mouse striatal region composed of the caudate nucleus, nucleus accumbens and the putaman as viewed within the basal ganglia along with the globus pallidus

2.2 Tools & Data Sets

2.2.1 The Allen Brain Atlas & Brain Explorer® 2

The Allen Brain Atlas[3] (ABA) aims to promote further understanding of the structure and neural connections of the brain in terms of gene function. The ABA is a constituent of the Allen Institute for Brain Science; utilizing in-situ hybridization techniques[4] (ISH) 3D architectures of the brain of mice and humans as well as other neuroanatomical structures have been realized in a hope to construct a knowledge base sufficient to fuel research into several neurobiological disorders. Currently there are five brain atlases in addition to maps of connectivity in the mouse brain and gene expression data for the mouse spinal cord. Of relevance to us and this project is the mouse brain atlas. This atlas details a high resolution genome wide collection of expression data for the adult male C57BL/6 mouse brain.

Expression data for each of the ~20,000 uniquely identifiable genes in the mice genome is available to download through the ABA's application programming interface[5] (API) as an expression energy volume stored in a sparse volume format. To download the following prototype must be used:

http://www.brain-map.org/aba/api/expression/[ImageSeriesID].sva

[ImageSeriesID] represents a place marker for an integer corresponding to a single gene. The energy expression volume for each gene represents the area of the mouse brain as divided into cubic voxels of resolution 200 microns in each of the x, y and z dimensions. The brain itself occupies voxel dimensions of 67 x 41 x 58 in a posterior-inferior-right (PIR) orientation and has been segmented into 209 identifiable regions.

The structure of each sparse volume data file consists of a sequence of voxels with a corresponding energy expression value in floating point

[3] http://www.brain-map.org/
[4] http://help.brain-map.org/download/attachments/2818169/ABADataProductionProcesses.pdf?version=1&modificationDate=1319477154403
[5] http://community.brain-map.org/download/attachments/525267/TheABAAPI_Final.pdf?version=2

precision. Energy expression values are a product of the density and intensity of expression found within the volume the voxel occupies. Note however only voxels with an energy expression value of above 0.0 are included. The following is a sample taken from the image series with Id 556990:

Comment:Smoothed energy volume for imageseriesId 556990
Dimensions:67,41,58
64,5,9,1.78316e-05
64,9,9,5.6337e-06
64,10,9,2.45415e-06
65,10,9,3.65273e-05
63,11,9,2.1395e-07
64,11,9,1.31732e-05
63,15,9,0.000120203
60,16,9,0.00052681

In order to download each energy expression volume we required a list of all the image series id's, this is available through a gene to image series mapping (note voxel expression data for some genes/image series' is not available) also provided through the API in a folder of additional data sets[6]. This folder also provides an annotated atlas volume detailing the brain structure by means of an informatics id (numbered between 1 and 232, some numbers are skipped hence 209 regions) which each voxel corresponds to. This volume is supplied in three resolutions of 25, 100 and 200 microns. The third and final file in this folder provides a brain structure ontology, mapping each brain structure to its corresponding informatics id.

The gene series map, the atlas annotation at a resolution of 200 microns (gene expression data is only available at this resolution), and the brain structure ontology data file were the three key data sets we needed from the ABA in order to download and process the ABA's complete expression data into a more useful form for our project.

Brain Explorer®[7] 2 is a desktop application for both Mac and Windows for viewing 3D structures generated from the ABA image series data for the human, mouse, and developing mouse brain atlases. As well as viewing the

[6] http://community.brain-map.org/download/attachments/525267/data.zip?version=1&modificationDate=1206659449002
[7] http://www.biomedcentral.com/1471-2105/9/153

2D ISH images in 3D, gene expression data can also be overlaid and visualized in the form of an intensity map.

2.2.2 SPM & SPMMouse

Statistical parametric mapping (SPM) refers to the application of random field theory in the analysis of functional imaging data (fMRI, PET) in order to characterize and examine differences in brain activity. These concepts have been captured in the software SPM[8] developed by the Wellcome Trust Centre for Neuroimaging[9]. The SPM software supports analysis of images using voxel based morphology, allowing us to examine and measure disease related changes in brain tissue of our mouse MRI imaging data sets in order to obtain raw atrophy calculations. However SPM is aligned for use with the human brain and therefore is only compatible with image files of certain voxel dimensions. This provoked the need for the SPMMouse[10] software which extends the functionality of SPM and caters for differences in size between the human and mouse brain.

2.2.3 LIBSVM

LIBSVM[11] (Chih-Chung Chang and Chih-Jen Lin, 2011) is a library used for creating and training support vector machines (SVMs). It provides several interfaces allowing use in applications written in Java, Python and MATLAB. The software also provides windows executables as well as C source code for compilation on UNIX or other operating systems. LIBSVM and its use will be discussed further in chapter 4 when discussing our design decisions and resulting implementation.

[8] http://www.fil.ion.ucl.ac.uk/spm/
[9] http://www.fil.ion.ucl.ac.uk/
[10] http://www.spmmouse.org/
[11] http://www.csie.ntu.edu.tw/~cjlin/LIBSVM/

2.2.4 R

R[12] is a programming language and software for statistical computing and graphics. RStudio[13] is the integrated development environment (IDE) and graphical user interface used during our project for programming in R. R is open source and therefore has an extensive list of packages[14] written by developers in the open source community. R was chosen for the several packages for analysis of MRI and bioinformatics data sets.

2.2.5 MRI Imaging Data Sets

The MRI imaging data sets from the study by Zhang et al. (2010) provide an in-vivo view of HD atrophy over time for both R6/2 and control mice. MRI scans are provided for 7 transgenic mice (with CAG repeats of 103-112) and 8 control mice. The survival period for transgenic mice was 84 ± 3.5 days. Mice were imaged at 3 weeks of ages, weekly up until 6 weeks of ages, then biweekly up until 12 weeks of age. MRI images are provided in a NIfTI image[15] format as dual .hdr and .img files as opposed to single .nii files.

[12] http://www.r-project.org/
[13] http://www.rstudio.com/
[14] http://cran.r-project.org/
[15] http://nifti.nimh.nih.gov/nifti-1

3 Review of Machine Learning Techniques

3.1 Purpose & Primary Concepts

3.1.1 Purpose

Machine learning (ML) is an area of active research within the discipline of artificial intelligence. It aims to provide computers and similar programmable technologies with the ability to learn and act out operations not explicitly defined within their programming. It is not unusual that a computer or program doesn't perform as well as expected in its intended environment and especially not within a novel one. ML algorithms overcome this by providing the computer with the ability to generalize and adapt to new information whilst allowing self-improvement of the knowledge base of a system. Similarly some tasks are not well-defined or easy to capture as a set of logical operations, except by example. ML algorithms are able to adjust their structure when trained on a labelled (during supervised learning) input space and output predicted responses in reaction to novel or unknown data.

ML also has applications in data mining, the primary focus of our project, whereby the algorithm is able to detect statistical relationships between observations in a large input space which are difficult to distinguish otherwise.

3.1.2 Choosing a Machine Learning Algorithm

When choosing a ML algorithm we must consider the size of our set of observations and in what application the computational structure we aim to create will be used. In terms of our project we seek to determine a relationship between gene expression data for a region and its atrophy level attributed to HD.

Computational problems can be categorized into those which aim to separate data into classes (classification) and those which highlight correlations between observations in the input space (regression).

In determining an ML approach for our project there were several key considerations:

- Data type of the input space
- Dimensionality of the input space
- The desired output

ML algorithms which aim to solve classification problems label novel data with a predicted class. Those which aim to solve regression problems label novel data with a real value, the precision of which is dependent upon the problem at hand. It is clear that our project aims to conduct regression analysis.

Note however in attempting to determine a relationship between region gene expression levels and atrophy, there is no guarantee of success. Therefore it would be sensible to validate our approach and confirm our methodology is feasible.

With this in mind before working with atrophy calculations we first utilized only the gene expression data. We attempted to create a model which was able to predict the brain region from just the expression data of that region. Therefore in choosing a ML approach we look to be able to conduct both classification and regression.

For both problems our input space consists of vectors. Each region is composed of a varying number of voxels presented as X, Y, Z voxel co-ordinates. Each voxel for a region will represent a single positive training instance. Dimensionality corresponds to the number of features/attributes per instance. The number of instances in our training files is highly dependent upon the region we are aiming to make a classifier for, but will range in the hundreds. Each region classifier will use all ~20,000 genes. Note however we are only creating classifiers for those regions with 20 or more voxels to ensure we have a sufficient number of positive cases in our training and testing files.

For regression analysis our training sets will have instances into the thousands as they each will contain all voxels within the brain. Again each instance will use all ~20,000 genes.

Our desired output is binary classification for our region classifiers (i.e. does this set of expression data belong to this region, yes/no) and output as a real number (the predicted atrophy level) for our atrophy predictors.

In summary our ML algorithm must be able to handle both small and large numbers of vector instances in its training sets with thousands of attributes and perform both regression and classification tasks.

With these requirements in mind and also the availability of software[16] for implementation, Support Vector Machines (SVMs) were chosen as our ML technique.

[16] See section 2.2.3 for LIBSVM.

3.2 Support Vector Machines

3.2.1 Underlying Concepts

A Support Vector Machine (SVM) is a vector based classifier trained using supervised learning which aims to find an optimally separating hyperplane between classes of data points in the input space (a training set). It aims to solve the following optimization problem (Chih-Wei Hsu et al, 2010):

$$\min w, b, \xi: \quad \frac{1}{2} w^T w + C \sum_{i=1}^{i} \xi_i$$

subject to: $(y_i \, w^T \phi(x_i) + b) \geq 1 - \xi_i \, , \, \xi_i \geq 0$

Where (x_i, y_i) are our instance label pairs

Equation 1 The SVM Optimization Problem

This separating hyperplane is known as the decision boundary. In order to reduce noise sensitivity this boundary should be maximally distance from all points in the vector space. This allows the SVM to generalize and classify new examples.

Twice the distance from the closest training example to this decision boundary is known as the margin, thus we aim to minimise errors and maximise the margin of our training set.

Those vectors or data points which define our decision boundary are described as support vectors.

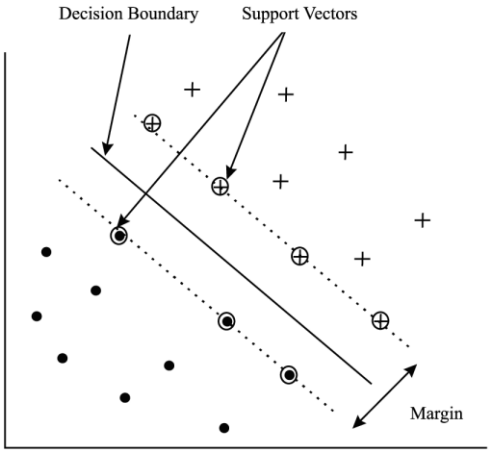


Figure 5 The optimal separating hyperplane (David Corney, 2002)

3.2.2 Feature Expansion & The Kernel Trick

Figure 5 details a linearly separable problem in the Cartesian plane which can be partitioned using a straight line. There are cases however where a separating hyperplane fails to partition the data.

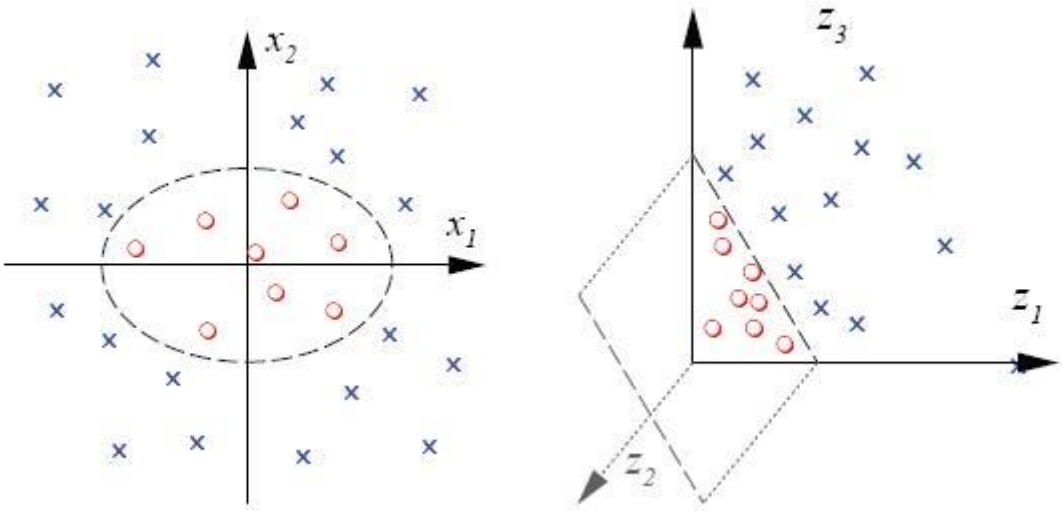


Figure 6 Mapping input features into a higher dimensional space (Chris Thornton)

A solution is to map the lower dimension input space to that of a higher dimension, (**Figure 6**) whereby it is hoped we can then find a separating hyperplane. This is what the kernel trick describes.

A kernel function allows us to perform this mapping by calculating the inner product (dot product) of pairs of data points onto which these pairs are mapped (the kernel trick). This gives a new dimension in our higher dimensional feature space for every inner product or pair of points in our lower dimensional input space. Optimisation is still efficient if the number of features grows as the function is dependent upon the dimensionality of the original space as opposed to our new feature space.

Several basic kernel functions can be defined (Chih-Wei Hsu et al, 2010):

- Linear: $K(x_i, x_j) = x_i^T x_j$
- Polynomial: $K(x_i, x_j) = (\gamma\, x_i^T x_j + r)^d,\ \gamma > 0$
- Radial Basis Function (RBF): $K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2),\ \gamma > 0$
- Sigmoid: $K(x_i, x_j) = \tanh(\gamma\, x_i^T x_j + r)$

Where γ, r and d are kernel parameters.

Note $K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j)$ where ϕ is the function that maps our input space to a higher dimensional space.

It is not always obvious which kernel and what parameters are appropriate for our application; it is very much dependent on what we are trying to model.

An RBF kernel allows features to define a hyper sphere for data partition, the linear kernel a hyperplane, and the polynomial models relationships in terms of feature conjunction up to the order of the polynomial (Roberto de Souza C, 2012). The sigmoid function is derived from another ML model known as neural networks. It is utilized as an activation function for the binary decision neurons of the net. Similarly choosing parameters for the kernel function is equally as difficult as well as tedious[17].

[17] See chapter 4 for a discussion of the approach we utilized for kernel and parameter selection.

3.2.3 Training & Cross Validation

SVMs are trained using supervised learning i.e. they are provided with a labelled training set. These training sets provide a series of examples/observations (instances) consisting of attributes/features. Each example is labelled with a class label for multiclass SVMs or a target value (any real number) for regression SVMs. It is from these labelled instances it is hoped the SVM will discover an underlying relationship between attributes/features and the target class or value.

Once a model has been generated it is common to then pass the model a validation set consisting of instances for which we know the target value or class label, however the data set will not have been used to train the SVM. This allows us to determine the predictive accuracy of the model and gives us a general idea of how it will perform on unseen data. Depending upon the accuracy on validation sets the model is then finely tuned. In our procedure we have replaced the use of a validation set with cross-validation on the training set.

Cross-validation allows us to reduce over-fitting, a common problem for SVMs. During the SVM training procedure k-fold cross validation aims to partition the data set into k distinct subsets of equal size (if possible), one subset is used for validation purposes (as a testing set), whilst the SVM is trained on the other k-1 subsets. This process is repeated k times, where each of the k subsets will be used once for the validation process.

Once a model has been generated from the training set and finely tuned using a validation set, or cross-validation, a final stage requires us to pass the SVM a final data set, the testing set. The testing set provides a more realistic estimate of the predictive accuracy of the SVM on completely unseen data. In generating our region classifiers the initial ABA data set was partitioned into separate directories for training and testing, providing us with two distinct data sets upon which to train our SVM classifiers then subsequently test.

Dependent upon software being used to train an SVM, training/testing sets can take different forms. LIBSVM[18] specifies the following format:

<label> <index1>:<value> <index2>:<value> <index3>:<value>……
<label> <index1>:<value> <index2>:<value> <index3>:<value>…… n attributes
<label> <index2>:<value> <index3>:<value> <index4>:<value>……
:

: x instances

Each of the x instances consists of attributes specified by index 1…n in ascending order. Note not every instance needs all indices (see instance 3 above).

[18] See section 2.2.3 for a brief introduction to LIBSVM.

4 Design & Implementation

4.1 Initial Objectives & Deliverables

Figure 7 presents a diagrammatic overview of the methodologies; 1, 2 and 3, considered during the course of the project. The next few sections highlight each of these methodologies and detail the issues which caused us to change our approach, allowing us to arrive at our final methodology.

4.1.1 MRI Analysis with SPM (Methodology 1)

During the first few weeks focus was placed largely on researching available software for the purposes of MRI analysis. This included the Statistical Parameter Map (SPM 8)[19] and SPMMouse software. Due to SPM and SPMMouse's compatibility with MATLAB we initially considered developing a MATLAB package which would allow SPM and SPMMouse to read and process the ABA database.

SPMMouse was built as an extension for a specific version of SPM (SPM 5), the recommended version of SPM at the time of writing is SPM 8. With this in mind as well as SPMMouse no longer being maintained we moved away from the SPM and SPMMouse software along with MATLAB, focusing our attention on the statistical programming language R[20].

The open source language R allows us to maintain our original goal of developing software and scripts all under an open source license. With contributions from the open source community, R also gives us greater variety in the choice of additional packages available for our project. Although choosing R meant help and support was limited, R felt the better choice over MATLAB for the additional reason that it is the language of choice for clinicians working in similar research. This change gave rise to methodology 2.

[19] See section 2.2.2 for information on SPM and SPMMouse.
[20] See section 2.2.4 for information on R

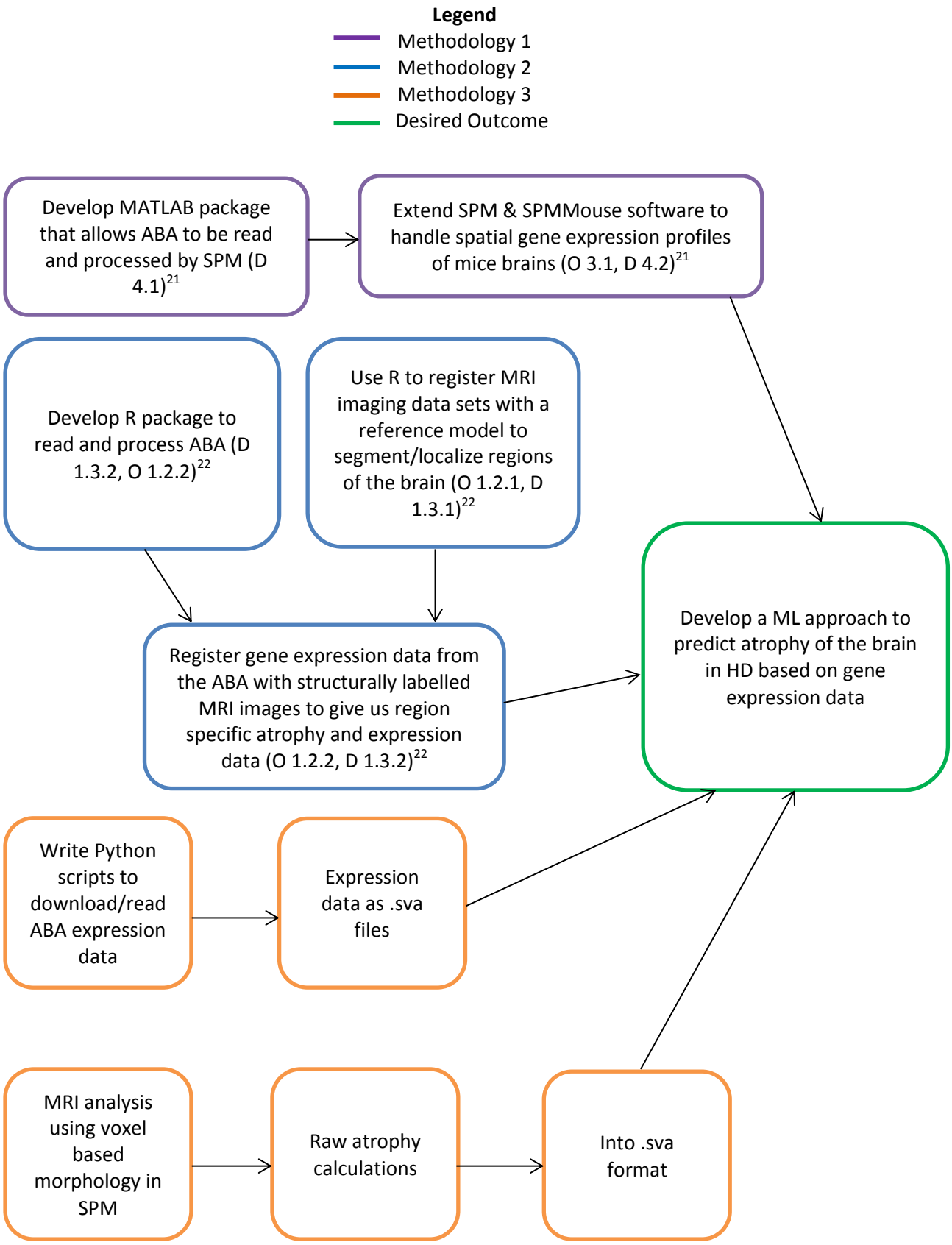


Figure 7 Overview of methodologies considered

[21] See the project plan (Appendix 1) for objectives O and deliverables D.
[22] See the interim report (Appendix 2) for objectives O and deliverables D.

4.1.2 MRI Analysis with R (Methodology 2)

Several packages are available for MRI analysis in R, three of which we experimented with; MRITC[23], FMRI[24] and DPMIXSIM[25].

MRITC utilizes variants of mixture models with voxels classed as belonging to one of three tissues; grey matter (GM), white matter (WM) and cerebral spinal fluid (CSF). MRITC allows the user to choose which method is used for MRI tissue classification by providing a range of mixture models whose parameters are specified by various algorithms.

Normal mixture models are probalistic models whereby we represent the sub populations; CSF, WM and GM tissues as part of an overall population, the brain.

A Markov random field model aims to address the problem of not knowing to which tissue type a voxel belongs. The Markov model represents a system undergoing transitions between states. In terms of brain segmentation, GM, WM and CSF represent our states. However we cannot directly observe which state or tissue a voxel belongs to, we assign voxels to tissues working on the basis that neighbouring voxels belong to the same tissue type. This leads to our model being referred to as the Hidden Markov model.

Combining the Markov random field model with a normal mixture model gives rise to the Hidden Markov Normal Mixture Model (HMNMM), the model we adopted in experimenting with tissue classification in MRITC.

MRITC also provides several algorithms for HMNMM parameter selection. These include the Iterated Condition Mode (ICM) algorithm, the Hidden Markov Random Field Expectation Maximisation (HMRFEM) algorithm or the Bayesian (MCMC) classification method.

[23] http://cran.r-project.org/web/packages/mritc/index.html
[24] http://cran.r-project.org/web/packages/fmri/index.html
[25] http://cran.r-project.org/web/packages/dpmixsim/index.html

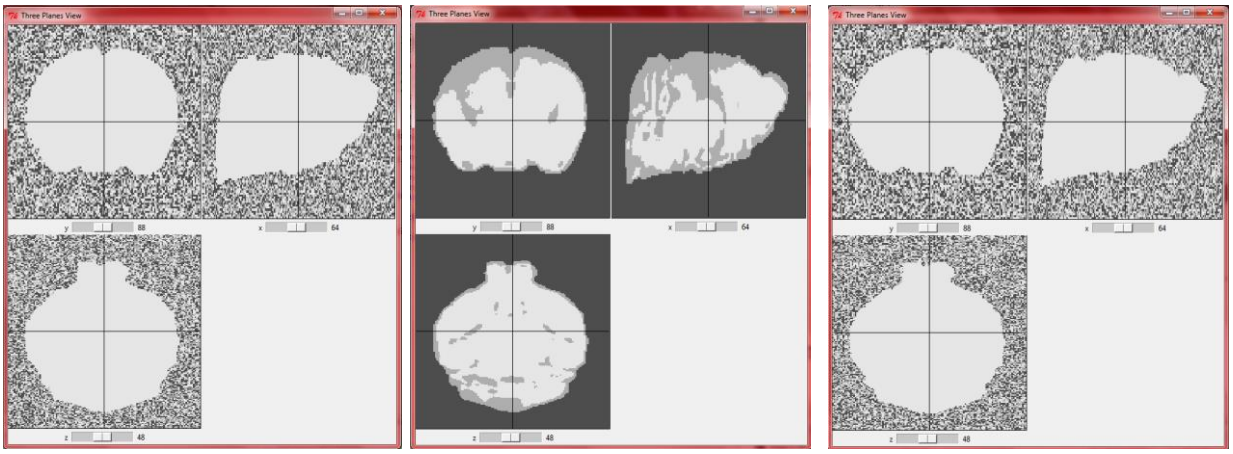


Figure 8 Visualization of MRI scan for transgenic mouse at 21 Days

After reading the NIfTI format image for the first transgenic mouse at 21 days we utilized MRITC's plot function to assign voxels to a one of the three tissue types, specifying the model and algorithm to use. Plot uses the function slices3d from the misc3d[26] R package to display the result. Shown in **Figure 8** is the result from left to right of plot using the ICM algorithm, the MCMC algorithm, and the HMRFEM algorithm.

As is visible in **Figure 8** it is hard to distinguish between the three tissues with the ICM and HMRFEM algorithm assigning all voxels to WM. There has been marginal success with the MCMC algorithm as we can distinguish some areas of GM and WM; however in all three images it is near impossible to identify any reduction in brain volume with the three tissues appearing as one single tissue.

With MRITC proving to be unsuccessful we looked at the DPMIXSIM and fMRI packages. Both however had similar results to MRITC, with little to no successful identification of tissue atrophication. This lack of success drove us to methodology 3.

[26] http://cran.r-project.org/web/packages/misc3d/index.html

4.2 Final Objectives & Deliverables (Methodology 3)

Looking closer into SPM and SPMMouse software we discovered they use a Voxel Based Morphology (VBM) approach to tissue classification. It was this level of complexity and lack of success in other approaches which caused us to re-evaluate our decision to move away from the SPM and SPMMouse software.

As opposed to using SPM and SPMMouse software to process the ABA database we considered using SPM and SPMMouse to transform the MRI imaging data sets into a format more compatible with the ABA.

Overcoming compatibility issues between SPM and SPMMouse versions we arrived at our final methodology. Using SPM and SPMMouse's VBM approach we aimed to generate raw atrophy calculations. We then looked to transform these atrophy calculations into a similar format to that of the ABA. With these MRI imaging and ABA data sets now compatible we would then use them in generating our atrophy predictors.

4.3 Final Methodology - A High Level Overview

In attempting to use gene expression data to predict region specific atrophy levels there is no guarantee of success. In designing our methodology we identified SVMs as the ML technique to use however before training an SVM for atrophy prediction it was necessary to validate our methodology and training procedure. With this in mind the project was split into two phases (**Figure 9**). The first phase would allow us to identify an SVM training procedure which could be utilized for generating each atrophy predictor in the second phase. In training an SVM for the purpose of region classification (**Figure 9** – Phase 1) strong predictive genes can be validated by examining the literature to see if they are known to be specific for that region. This would confirm our data pre-processing steps in region classification were sensible and provides a solid platform for the data pre-processing and training procedure we will use in atrophy prediction (**Figure 9** – Phase 2).

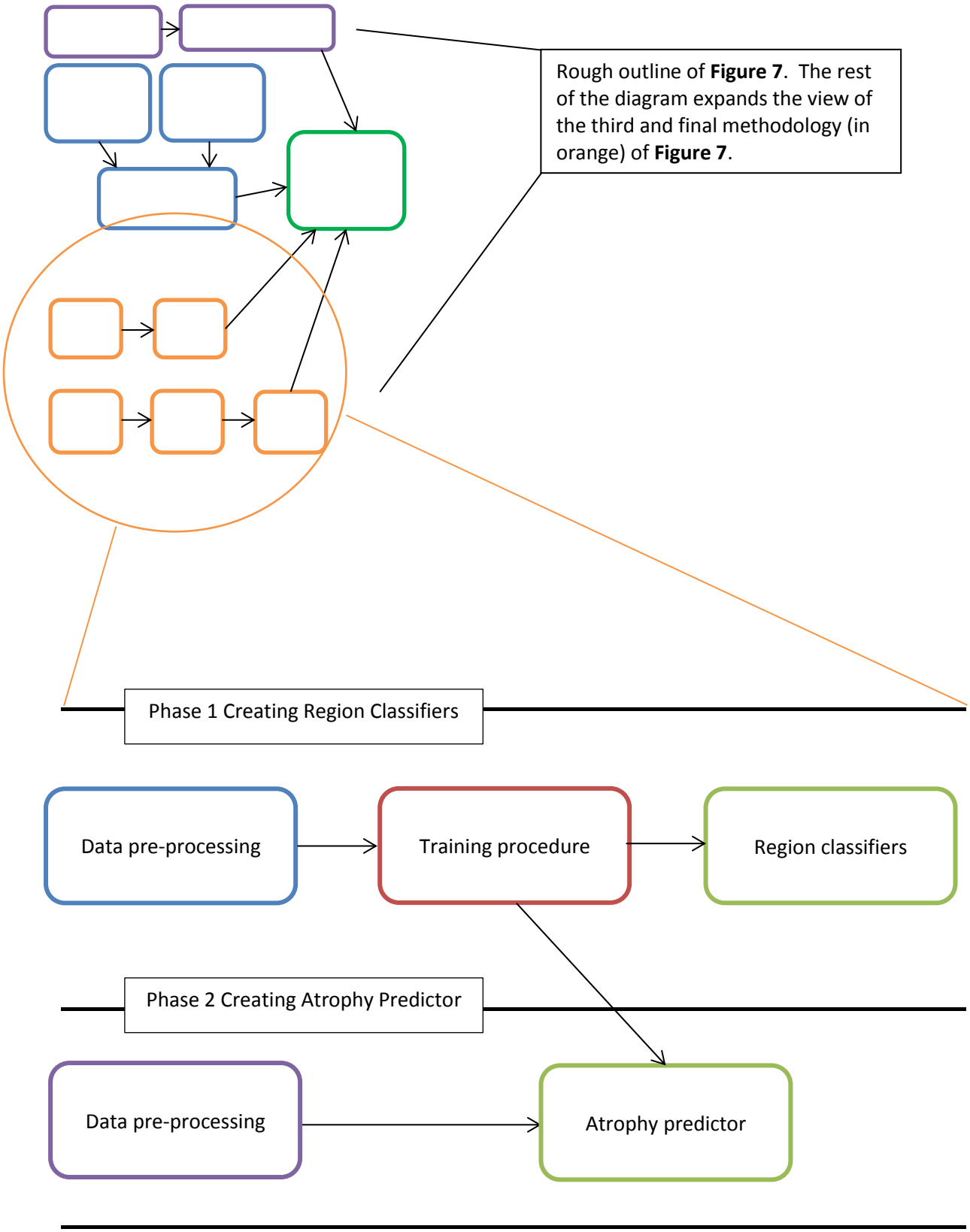


Figure 9 High level overview of design process (more detailed look at Figure 7)

4.3.1 Choosing Python

As touched upon in subsequent sections there are a number of necessary data pre-processing steps required to execute our final methodology. These included downloading vast amounts of data as well as organising and separating said data based on certain criteria. This required a language which was easy to learn and quick to program in as well as expressive. We also needed a language with libraries available for downloading and parsing HTML.

With these requirements in mind we chose Python[27] for a majority of the pre-processing steps. However this choice did highlight the issues of memory management in languages which offer a 'black-box' solution as opposed to complete control of memory allocation, forcing us to turn to C for later data pre-processing stages. Later chapters will discuss this further.

4.4 Creating Region Classifiers

4.4.1 Data Pre-Processing

In creating our region classifiers there are several necessary data pre-processing steps. These are presented in **Figure 10** as a further break down of the data pre-processing portion in **Figure 9** – Phase 1.

The next few paragraphs highlight each of the stages of **Figure 10** in more detail.

[27] http://www.python.org/

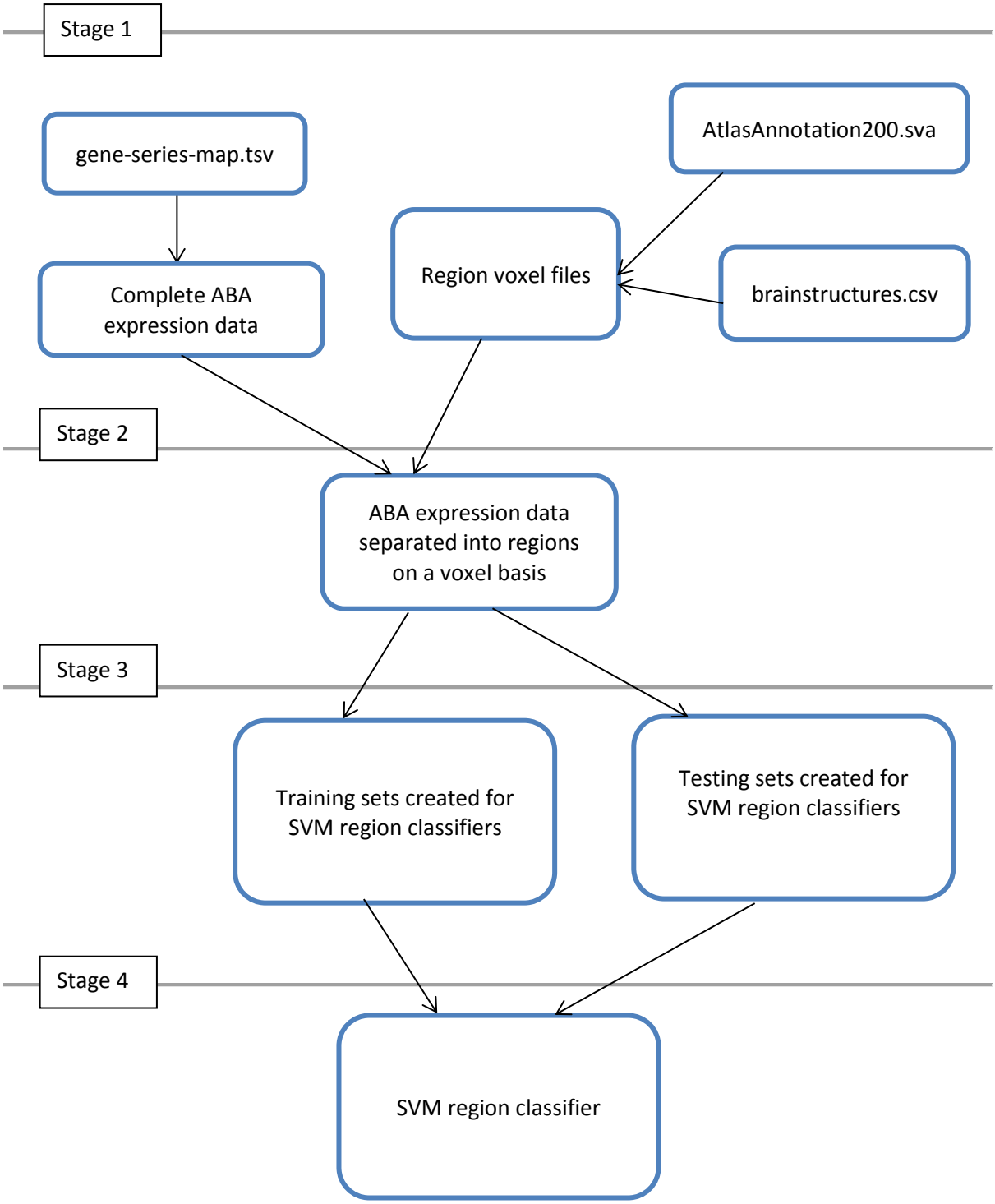


Figure 10 Overview of data pre-processing for Figure 9 – Phase 1

Stage 1

Stage 1 made use of the ABA's folder of additional data sets[28] available through their API. As discussed in section 2.2.1 this folder consisted of a mapping between an image series id and its gene (gene-series-map.tsv), the mapping between voxel (resolution of 200 microns) co-ordinates and an informatics id (AtlasAnnotation200.sva) and the mapping between an informatics id and its brain structure (brainstructures.csv). Stage 1 consisted of two parallel steps. The first step involved downloading the complete ABA expression data and required just the gene-series-map.tsv. The second step extracted information from the brain structure ontology (brainstructures.csv) and the AtlasAnnotation200.sva in order to collect the voxels for each region in separate files. This would allow further separation of the ABA expression data into their regions in order to create training and testing sets for our SVM.

Stage 1 - Step 1

Using a Python script to iterate through the gene-series-map.tsv (see **Table 1**) and looking at the third column, "imageseriesid", allowed us to access each webpage of expression data by substituting in the current "imageseriesid" into the following prototype:

http://www.brain-map.org/aba/api/expression/[ImageSeriesID].sva

Table 1 First few rows of gene-series-map.tsv

genesymbol	genename	imageseriesid	plane	entrezgeneid	ncbiaccessionnumber
Pzp	pregnancy zone protein	69015212	sagittal	11287	NM_007376
Aanat	arylalkylamine N-acetyltransferase	71495179	sagittal	11298	NM_009591
Aatk	apoptosis-associated tyrosine kinase	71213880	sagittal	11302	NM_007377

[28] See section 2.2.1 for more information on the folder of additional data sets available at http://community.brainmap.org/download/attachments/525267/data.zip?version=1&modificationDate=1206659449002

Once the webpage had been accessed we needed to read each line, saving it to a file. This approach is summarized as pseudo-code in **Pseudo-Code 1**[29].

```
1    For imageseriesid in gene-series-map.tsv
2        Lines = read lines from URL/imageseriesid.sva
3        Create file DirOnComp/imageseriesid.sva
4        For line in Lines
5            Save line to file DirOnComp/imageseriesid.sva
6        Close file DirOnComp/imageseriesid.sva
```

Pseudo-Code 1 Downloading gene expression data from ABA

Summary of results from Stage 1 – Step 1

We now have the complete ABA expression data, ~20,000 sparse volume files saved as [imageseriesid].sva where "imageseriesid" was taken from the gene-series-map.tsv file. Each file has the following structure:

```
Comment:Smoothed energy volume for imageseriesId
Dimensions:67,41,58
X0,Y0,Z0,Energy expression level
X1,Y1,Z1,Energy expression level
X2,Y2,Z2,Energy expression level
    :
    :   (For all X,Y,Z voxel co-ordinates the gene with id [imageseriesid] is
    :    expressed in)
```

[29] See Code Listing 1 for full Python script.

Table 2 First few rows of brainstructures.csv

StructureName	Abbreviation	ParentStruct	red	green	blue	informaticsId	StructureId
Basic cell groups and regions	Brain		176	255	255	1	0
Cerebrum	CH	Brain	176	240	255	2	70
Cerebral cortex	CTX	CH	176	255	184	3	85

Stage 1 – Step 2

To separate the gene expression data into their respective regions we require the voxels co-ordinates for each region.

Iterating through brainstructures.csv (see **Table 2**) and looking at the "informaticsid" column allowed us to reference each brain structure (the second last integer on each line) in the AtlasAnnotation200.sva.
This separated data will eventually be used to create our testing and training sets. With this in mind code was written to create two folders, a training folder and testing folder; each of which contained region files with voxels allocated alternately. This results in two different data sets for each brain region; one will be used to create our training sets for our region classifiers whilst the other will be used to create unseen testing sets.

```
1   informaticsIDS = get Informatics ids from brainstructures.csv
2   for id in informaticsIDS
3       trainingTurn = true
4       create and open training region file DirOnComp/Train/id.txt
5       create and open testing region file DirOnComp/Test/id.txt
6       open AtlasAnnotation200.sva
7       for line in AtlasAnnotation200.sva
8           if last integer of line == id
9               if trainingTurn
10                  save co-ordinates to file DirOnComp/Train/id.txt as line
11                  trainingTurn = false
12              else
13                  save co-ordinates to file DirOnComp/Test/id.txt as line
14                  trainingTurn = true
15      close file AtlasAnnotation200
16      close file DirOnComp/Train/id.txt
17      close file DirOnComp/Test/id.txt
```

Pseudo-Code 2 Separating co-ordinates based on their region

Summary of results from Stage 1 – Step 2

Pseudo-Code 2[30] creates 209 (for the 209 regions segmented by the ABA) [informaticsid].txt files where "informaticsid" was taken from the brainstructures.csv file. Each .txt has the following structure:

X0,Y0,Z0
X1,Y1,Z1
X2,Y2,Z2
:
: (For X,Y,Z voxel co-ordinates belonging to the region identified by
: "informaticsid")

Following the separation of the expression data, region files with less than 10 voxels in both the training and testing directories were removed to ensure we have a sufficient number of instances for training and testing purposes. This left us with 113 region files each of which are comprised of at least 20 voxels.

Stage 2

We now have a list of the voxels for each region and our complete ABA expression data, both acquired from the steps in stage 1. The next stage requires us to separate the voxels for each gene's expression data into their respective regions.

Pseudo-Code 3[31] shows our first naïve approach to separating the expression data on a region basis. Performing a time complexity analysis we see that the total number of executions is $\cong n \times r \times m \times g$ where n = no. of brain regions, r = average no. of lines in a brain region file, m = no. of genes and g = average no. of lines in a gene expression file. With both r and g as constants this gives us a time complexity of $O(n \times m)$. Clearly this isn't fast or efficient enough with I/O overheads for 113 region and ~20,000 gene files to process.

[30] See Code Listing 2 for full Python script.
[31] See Code Listing 3 for full Python script.

```
1       for each region in our 113 regions
2           for line in opened region file
3               extract voxel co-ords from line
4               for each gene in ~20,000 genes
5                   for line in opened gene file
6                       extract voxel co-ords from line
7                       if region co-ords == gene co-ords
8                           create file for output for region if it doesn't exist
9                           append gene co-ords, gene expression and gene id to file
10                          close output file
11                          continue
12                  close gene file
13          close region file
```

Pseudo-Code 3 First approach to separating expression data into regions

To decrease overheads with I/O operations we then looked to create a 3D matrix of expression data for each gene stored in a Python dictionary as a gene id: gene expression matrix pair.

```
1   geneMap = new empty dictionary
2    for each gene in ~20,000 genes
3       matrix = new empty matrix
4       for line in opened gene file
5           extract voxel co-ords from line
6           store expression level in matrix at voxel co-ords
7       close gene file
8       add gene:matrix pair to dictionary
9     return geneMap
10
11
12  for each region in 113 regions
13      create region output file
14      for line in opened region file
15          extract voxel co-ords from line
16           for gene, matrix in geneMap
17              get expression level at current co-ords in matrix
18              write co-ords, expression level and gene id to region output file
19      close region output file
20      close region file
```

Pseudo-Code 4 Second approach to separating expression data into regions

The algorithm presented in **Pseudo-Code 4**[32] proved to be a lot faster with look ups occurring to the expression matrix in memory as opposed to a look up with an I/O operation. However the large amount of memory we required (a matrix with dimensions 67 x 41 x 58 for each gene) coupled with Python's approach to memory management caused the OS to kill the Python process during execution. With limited control over memory allocation in Python we implemented the same algorithm in C[33] where we can allocate the exact amount of memory we needed dynamically. To simplify the creation of training and testing sets this C code was written to sort the data and write the output partly in LIBSVM's[34] format. This gives us the expression data for all genes, for a single voxel, appearing on a single line. In addition the C code writes an additional file that provides a mapping (an ExpressionMappingTrain.csv file) between a "feature id" indexed from 1-25551 and a gene's image series id. These feature ids will be used as opposed to image series ids within the training sets as LIBSVM insists on indexing attributes in ascending order. These ids will also allow us to carry out feature selection to determine each gene's predictive power in region classification.

In order to sort this data more efficiently the C program was compiled and executed on my supervisor's Linux machine due to its greater RAM (20Gb of RAM was used during sorting) and subsequent processing power.

Summary of results from Stage 2

We now have 113 [informaticsid].txt files in each of our training and testing directories for each region composed of 20 voxels or more. Each file has the following structure with a real example shown in **Figure 11**:

X0,Y0,Z0, FeatID:Expression FeatID:Expression ……(For all feature ids that map to
X1,Y1,Z1, FeatID:Expression FeatID:Expression an image series id)
X2,Y2,Z2, FeatID:Expression FeatID:Expression

: (For all X,Y,Z voxel coordinates that comprise the
: region identified by informatics id)

[32] See Code Listing 4 for full Python script.
[33] See Code Listing 5 for full C code.
[34] See section 3.2.3 for discussion of LIBSVM's format.

41, 33, 13, 1:5.91398 2:0.0531915 3:3.41902 .....
42, 32, 14, 1:5.44662 2:0.439908 3:1.28105 .....
42, 33, 14, 1:4.16373 2:0.108448 3:1.78403 .....
41, 34, 14, 1:5.90673 2:0.00626236 3:6.39334 .....
40, 36, 14, 1:1.09726 2:0.00041572 3:2.93271 .....

Figure 11 Example rows from region set 16.txt

Stage 3

With all the expression data now separated into our 113 regions we can create training and testing sets for our SVM region classifiers.

With no need at this stage for exact control of the allocation of memory, we again chose Python to further manipulate our data from Stage 2 into training and testing sets. Each training and testing set would need to take the LIBSVM form as discussed in section 3.2.3.

Stage 3 – Training Sets[35]

1 1:2.6192 2:0.692054 3:0.899749 5:0.959664 6:2.8656 7:0.0202167 .....
1 1:9.40227 2:0.109717 3:0.966273 4:0.518034 5:2.07318 6:9.63867 .....
1 1:4.09222 2:0.084193 3:0.646193 4:0.456955 5:1.61856 6:7.83364 .....
:
:
-1 1:13.0583 2:0.62457 3:2.36467 4:2.53573 5:0.0359436 6:9.65331 .....
-1 1:4.66941 2:1.9235 3:0.44831 4:0.0213208 5:0.128233 6:9.18999 .....
-1 1:11.6803 2:1.02983 3:2.0205 4:0.243166 5:0.131933 6:10.8633 .....
:
:

Figure 12 Example rows from training set 1.txt

Figure 12 provides a detailed look at the format of our training sets, showing several rows from the training set (1.txt) for the brain region with an informatics id of 1.

Each row in our training set represents a single voxel. Each row is comprised of <index>:<value> pairs where an 'index' denotes a gene

[35] See Code Listing 6 for full Python script which generates our training sets.

(identified by a feature id) expressed within that voxel. 'Value' denotes its corresponding energy expression level.

The first few hundred rows (labelled with a 1) in the training set are positive cases for region 1, i.e. those voxels which form and belong to that region and are within the training directory. Subsequent rows are negatives cases (labelled with a -1), i.e. voxels which do not form or belong to region 1. Negative cases were chosen by selecting two random voxels from each of the other 112 regions in our training directory.

Stage 3 – Testing Sets[36]

Testing sets took a similar form to that of training. Each one comprised rows representing voxels with feature id, energy expression level pairs. All voxels in the testing set that formed the region in question were selected as positive instances (labelled with a 1) with negative cases (labelled with a -1) comprised of two randomly selected voxels from each of the other 112 regions in the testing directory. Note that labels in the testing sets are only provided as a means for LIBSVM to calculate the accuracy of the SVMs classification and do not adjust the classifier model.

Summary of results from Stage 3

At this point we have 113 training and 113 testing sets, stored as [informaticsid].txt files. Each training set is comprised of all the voxels in the training data set within that region, as well as two random voxels from the other 112 regions. Our testing sets again take the same format with voxels taken from the data set in the testing directory.

[36] Code Listing 6 now executed on the testing directory was used to generate our testing sets.

4.4.2 The Training Procedure

Stage 4 comprised the training procedure and creation of region classifiers in **Figure 9** – Phase 1.

Before training an SVM there are several choices which need to be made. We need to select a kernel, the kernel parameters, and a value for the penalty parameter C of the error term in **Equation 1.**

To achieve rapid results and for guidance in training our SVM's we utilized the proposed training procedure as stated in the guide by Chih-Wei Hsu et al (2010). They propose the following steps:

- Transform data to the format of an SVM package
- Conduct simple scaling on the data
- Consider the RBF kernel $K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2),\ \gamma > 0$ Equation 1
- Use cross-validation to find the best parameter C and γ
- Use the best parameter C and γ to train the whole training set
- Test

LIBSVM provides several tools of which we can use to complete the above procedure. These are discussed as we address each step.

Transforming the data

Stage 3 provided us with training and testing sets in LIBSVM's format. To verify our sets comply with LIBSVM's format we can use the LIBSVM's "checkdata.py" script.

Conducting scaling

We chose to scale our attributes within the range of [0,1]. Scaling the data sets is useful in countering any dominating large valued attributes and minimises complexity in dot product calculations. It is imperative that both training and testing sets are scaled using the same factors. LIBSVM's "svm-scale.exe" allows us to scale our training sets to within our specified range

and save the parameters used. These can be restored later on when scaling the corresponding testing set.

Cross-validation and parameter selection

The RBF Kernel requires two parameters, C and γ. LIBSVM's "grid.py"[37] Python script allows us to conduct a grid search on pairs of C and γ parameters, commenting on the cross-validation rate/accuracy resulting from each pair. The best C and γ parameters are considered to be those which offer the best cross-validation rate. "grid.py" specifies the following defaults:

- $C = 2^{-5}, 2^{-3}, \ldots, 2^{15}$
- $\gamma = 2^{3}, 2^{1}, \ldots, 2^{-15}$
- 5 fold cross-validation

Chih-Wei Hsu et al (2010) recommend first performing a coarse grid search on pairs of C and γ followed by a finer grid search in the neighbourhood of the best C and γ. However given our large number of features Chih-Wei Hsu et al (2010) also state that "the nonlinear mapping does not improve the performance", indicating a linear kernel (this only uses C as a parameter) might be preferable over an RBF. Considering this we conducted a preliminary analysis into which kernel would be suitable for our given problem. In this analysis we created region classifies for four key regions known to be affected by HD with an additional control region. Using both an RBF and linear kernel and following the recommendation above we conducted a coarse grid search using the above default parameters, and a subsequent finer grid search to determine the best C, γ for the RBF kernel and the best C for the linear kernel.

Training the SVM

LIBSVM provides the tool "svm-train.exe", which when executed on our scaled training set, generates and saves our SVM region classifier as a

[37] "grid.py" requires the "gnuplot" software available at http://www.gnuplot.info/download.html

.model file. This tool requires we specify the kernel to use and dependent upon this, values for the kernel's parameters.

Testing the SVM

LIBSVM's "svm-predict.exe" allows us to specify the SVM model from the previous step (our region classifier) and a data set to classify. Execution with our testing data sets allows us to see how accurately each of our SVMs classifies the testing data into their respective regions.

4.4.3 Automating Training & Testing

LIBSVM does provide a Python script ("easy.py") which automates scaling of data, parameter selection using cross-validation, model creation and testing.

However "easy.py" is limited in terms of output and doesn't provide support for coarse and a subsequent finer grid search. With this in mind an additional Python script[38] was written to automate the procedure allowing for coarse and finer grid searches as well as the additional output which several of the tools provide. As discussed earlier our script was initially executed to create region classifiers for five regions (Cerebellum, Striatum, Substantia Nigra Reticular, Cortical Amygdalar, and the Pallidum Medial region) using both an RBF and linear kernel to determine which would be preferable when moving to creating classifiers for all regions. The script executed the following steps (these refer to the RBF kernel however these steps were repeated for the linear kernel with parameter selection for C only) with the final step conducting automatic testing of the generated model file:

For each of the five training sets [informaticsid].txt in our set of five regions:

- Check data format of training set using "checkdata.py", there is no need to check the format of the testing set as this mirrors the training.
- Scale training file and testing files using "svm-scale.exe" saving the scaled data to [informaticsid].scale and parameters to [informaticsid]Scale.params.

[38] See Code Listing 7 for the Python script which automates our procedure.

- Conduct a coarse grid search using "grid.py" with default parameters on [informaticsid].scale. Save pairs of C and γ tried with corresponding cross-validation rate to [informaticsid]_ScaleGrid.coarse.out. Save graph drawn with "gnuplot.exe" to [informaticsid]ScaleGrid.coarse.png
- Conduct a finer grid search using "grid.py" with best C and γ on [informaticsid].scale. Save pairs of C and γ tried with corresponding cross-validation rate to [informaticsid]_ScaleGrid.finer.out. Save graph drawn with "gnuplot.exe" to [informaticsid]ScaleGrid.finer.png
- Use "svm-train.exe" to create region classifier using best C and γ. Save model file to [informaticsid].model
- Use "svm-predict.exe" to test the region model file on an unseen testing set

A note about "grid.py" implementation & changes to script file

In the "grid.py" file during the update to the C and γ (for the RBF kernel) variables to reflect the current best C and γ parameters, if the current cross-validation rate is equivalent to the best rate and γ is the same but C differs, the best C is chosen to be the smallest. This highlights the importance of the trade-off between errors of the SVM (where we allow some misclassification through the error term) on training data and maximisation of the margin. If C were too large we would be concerned about over-fitting.

Additionally in allowing for coarse and finer grid searches in the automation of our procedure it is necessary to pass on the best γ and/or C (RBF/linear kernel) parameter found between each stage of the process. This required a means of which to store the best parameters when transitioning from a coarse to finer grid search. When running "grid.py" we can specify a path to an output file using the -out flag where "grid.py" will print output to both the command line:

[local] 5 -7 87.1972 (best c=32.0, g=0.0078125, rate=87.1972)
[local] -1 -7 87.1972 (best c=0.5, g=0.0078125, rate=87.1972)
:
:

[local] 13 -3 87.1972 (best c=8.0, g=0.0001220703125, rate=98.9619)
8.0 0.0001220703125 98.9619

and to the file specified by –out:

log2c=5 log2g=-7 rate=87.1972
log2c=-1 log2g=-7 rate=87.1972
:
:
log2c=13 log2g=-9 rate=93.0796
log2c=13 log2g=-3 rate=87.1972

Notice the command line output prints the best C and γ (this is for an RBF example) to the last line whereas the file specified by –out only contains pairs of $\log_2 C$ and $\log_2 \gamma$. It would be more useful to have output similar to the command line in our file from which we can extract the best C and γ for use in a finer grid search and when training our SVM using "svm-train.exe". This required a few changes to the "grid.py" file from lines 446 to 463[39]. With these changes implemented our output file now looks like this:

log2c=5 log2g=-7 rate=87.1972
log2c=-1 log2g=-7 rate=87.1972
:
:
log2c=13 log2g=-3 rate=87.1972
8.0 0.0001220703125 98.9619

The output to the command line remains the same.

[39] See Code Listing 8 for changes to "grid.py" script file.

4.4.4 Validating the Training Procedure

After the creation of the model files for the five regions, the "svm-predict.exe" tool was then used to determine the accuracy of the region classifiers (created using both the linear and RBF kernels), on our unseen testing sets.

Table 3 provides a comparison of the accuracies which resulted from models generated using the RBF and linear kernels through cross-validation during training and subsequent use of the classifiers in predicting unseen testing data.

Table 3 Comparison of RBF and Linear Kernel Accuracies

informaticsId	StructureName	RBF CV accuracy	RBF unseen accuracy	Linear CV accuracy	Linear unseen accuracy
16	Cortical amygdalar area	98.2578%	97.9094%	97.9094%	97.561%
35	Striatum	96.4567%	97.2441%	94.8819%	96.4567%
57	Pallidum_ medial region	98.893%	99.262%	98.155%	98.893%
138	Substantia nigra_ reticular part	99.6139%	98.4496%	99.6139%	98.4496%
224	Cerebellum	96.9492%	96.5986%	95.9322%	97.9592%

In averaging the accuracy determined by cross-validation (CV) and prediction on unseen testing sets for each of the five regions, the RBF kernel provides marginally better prediction. However the average time taken to create each of the region classifiers was much longer than the average time using a linear kernel. Additionally the .model file generated with an RBF kernel is much harder to analyse in terms of feature selection and would make ranking genes in terms of predictive power much more difficult.

With five .model files now generated the next step involved using LIBSVM's feature selection tool[40] to process each one in turn and rank each of the ~20,000 genes in terms of predictive power for that region. **Table 4** details the top ten genes as selected by this tool, in predicting the striatal region. The ExpressionMappingTrain.csv file produced during the execution of our

[40] Tool is available at http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/#primal_variable_w_of_linear_svm_and_feature_selection

Table 4 Striatum Predictor Gene Ranking

Rank	genesymbol	genename
1	Eif3s1	eukaryotic translation initiation factor 3, subunit 1 alpha
2	LOC383613	similar to Eukaryotic translation initiation factor 1 (eIF1) (Protein translation factor SUI1 homolog)
3	1700021K14Rik	RIKEN cDNA 1700021K14 gene
4	Ppp1r2	protein phosphatase 1, regulatory (inhibitor) subunit 2
5	Pla2g12a	phospholipase A2, group XIIA
6	Ppp1r1b	protein phosphatase 1, regulatory (inhibitor) subunit 1B
7	LOC432785	similar to mKIAA0686 protein
8	Ttc21a	tetratricopeptide repeat domain 21A
9	Zdhhc4	zinc finger, DHHC domain containing 4
10	B230114H05Rik*	RIKEN cDNA B230114H05 gene (non-RefSeq)

C code was used to translate the feature ids into image series ids, using the gene-series-map.tsv file from the ABA folder of additional data sets these were then translated to the corresponding gene symbols and gene names.

A literature review of genes known to be highly expressed and those which perform vital functions in the striatum would warrant further analysis outside the scope of this project and would be considered further work for extending the project in the future. Therefore in validating our training procedure and methodology we utilized only the cross validation and unseen accuracy values. The unseen accuracy values clearly indicate that the predictor is working as it is essentially correctly classifying unseen data to over 90% accuracy. These values agree with the cross validation accuracy indicating the machine learning process is more than adequate. If our SVM classifiers were being over-trained then the cross validation would have high accuracy but the unseen data accuracy would be low.

With at most a 1% difference in average accuracy for each of the five regions for the linear and RBF kernels, coupled with substantially less processing time and greater ease in feature selection, it was decided that the linear kernel would be used to generate the models for the rest of the regions and subsequent atrophy predictors. Furthermore the coarse grid search was favoured over a coarse and subsequent finer grid search in order to cut processing time as it was rarely the case the finer search provided better

parameters. These changes were incorporated into a new Python script[41] to automate the SVM creation process. This script was also run on my supervisor's machine, again for its superior processing power.

Table 5 Excerpt of Prediction Accuracy Results for All Region Classifiers

StructureName	Linear CV accuracy	Linear unseen accuracy
Accessory olfactory bulb	100%	99.5885%
Ammon's Horn	98.4772%	98.8136%
Anterior amygdalar area	98.374%	97.9675%
Anterior hypothalamic nucleus	100%	99.5868%
Anterior olfactory nucleus	97.6048%	98.8024%
Anterior pretectal nucleus	98.3607%	98.7705%
Anteromedial nucleus	97.9167%	98.7448%
Lateral septal nucleus	98.9726%	99.3151%
Lateral vestibular nucleus	99.5781%	98.7342%
Magnocellular nucleus	98.7234%	99.5726%
Magnocellular reticular nucleus	96.7611%	98.7805%
Main olfactory bulb	99.8578%	99.8575%
Medial amygdalar nucleus	99.2453%	97.7273%
Superior colliculus_ sensory related	99.3056%	100%
Superior olivary complex	98.8142%	99.2063%
Taenia tecta	98.8506%	99.6169%
Tegmental reticular nucleus	98.3539%	99.5868%
Thalamus	84.7909%	92.3664%

Table 5 provides the accuracy for cross validation and unseen testing sets for several of the 113 regions. See **Appendix 3** for predictive accuracy for all regions.

With these promising results the framework used for training our SVM classifiers will form the basis of the procedure used for training each atrophy predictor.

[41] See Code Listing 9 for full Python script.

4.5 Creating an Atrophy Predictor

4.5.1 Calculating Raw Atrophy Levels

After successfully validating the training procedure adopted during the creation of region classifiers we were then able to train an atrophy predictor (**Figure 9** - Phase 2) for several of the time points at which MRI scans were taken. However before we can create SVM training sets for each time point we first needed to obtain atrophy calculations from our MRI scans. Essentially we aimed to transform our MRI imaging data sets into a format similar to that of the ABA's expression data.

Determining spatial atrophy in the mouse brain due to HD was carried out jointly with my supervisor on his machine, as it required a powerful Windows-based system and resulted in the processing of 6.8 GB of data.

As discussed in section 4.2 it was SPM's Voxel Based Morphology (VBM) approach to tissue classification which would allow us to calculate atrophy levels for individual voxels. Utilizing SPM 5 with the SPMMouse toolkit we largely followed the SPMMouse VBM tutorial[42].

Our original John Hopkins MRI data provided a directory for each of the 8 wild-type and 7 R6/2 mutant mice. Each directory consisted of a set of 7 scans, named according to age of the mice in days; 21, 28, 35, 42, 56, 70, 84, at which the scan was taken. In determining atrophy it would be necessary to compare differences in volume of the various tissues between the wild-type and the mutant mice. With this in mind the data was rearranged into directories to allow comparison between MRI scans at any particular stage. This translates to 7 directories named 21, 28, 35, 42, 56, 70, and 84 each containing 15 scans (8 wild-types and 7 mutants).

Before any comparison in volume can be made the images must be aligned. SPMMouse provides a template of the mouse grey matter. This provided us within a single image to which we could align each of our images. Each of the MRI images were viewed in SPMMouse and compared against this grey

[42] www.spmmouse.org

matter template. A rigid-body translation of (0.0, -1.2, -3.4) was applied to each of the John Hopkins MRI images based on visual inspection of each against the grey matter template.

Each image was then segmented into grey matter (GM), white matter (WM) and cerebral spinal fluid (CSF) (see **Appendix 4**) using SPMMouse's segment functionality without registration, as manual registration had already been applied.

Only the grey matter segmented images were used for atrophy determination. These were smoothed using isotropic smoothing of 500 microns, with a parameter setting of [0.5, 0.5, 0.5], given in mm.

At each of the seven time points (21, 28,…,84) a 'Statistical Parameter Map' was generated by conducting a two-sample t-test model across every voxel of each MRI image. This conducted a comparison of the 8 smoothed grey matter images for the wild-type mice with the 7 smoothed grey matter images for the mutant mice and generated t-value scores for each voxel, a threshold was then applied at a 'False Discover Rate' (FDR) of 5%.

The result was 7 statistical parameter maps, one for each of the time points over the entire brains, providing t-values reflecting the significance of atrophy within each region.

Visually examining each of the 7 images, the first two time points, 21 and 28 days, showed no significant difference in tissue volume. At 35 days there was a weak overall atrophy level. 42 and 56 days showed similar but weak atrophy in specific regions. The final two time points, 70 and 84 days showed more significant and similar features of atrophy.

Hence in order to determine statistical significance and reveal stronger temporal features of atrophy, the image data from 21, 28 and 35 days were 'pooled' together. Similarly day 42 and day 44 were pooled. The final two time points, 70 and 84 days were also pooled together.

This resulted in three statistical parameter maps, see **Appendix 5** for an example of one of these maps, which allowed us to determine atrophy hot spots (**Figure 13**).

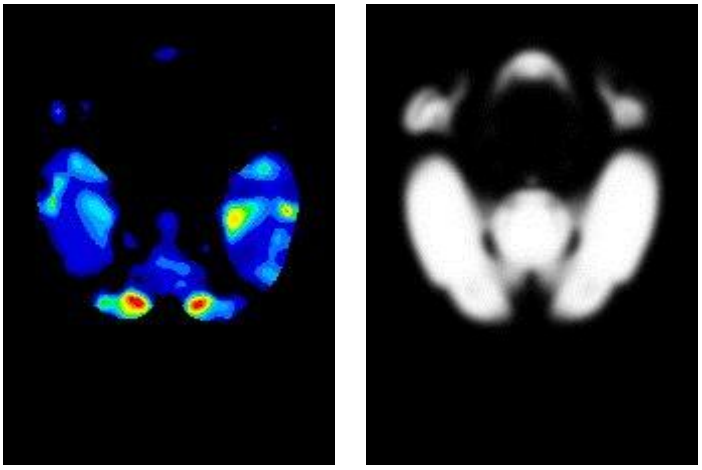


Figure 13 Viewing the atrophy map (SPMMouse co-ordinate system) of 70 day and 84 day (pooled) mice showing basal ganglia atrophy hotspot (left) together with the Grey Matter map for comparison (right)

However to work with these atrophy levels alongside expression data the images' co-ordinate system needs to be adjusted. The three SPMs are in SPMMouse's co-ordinate system of 135 slides in coronal orientation and must be transformed to the ABA's co-ordinate system of 58 slices in sagittal orientation.

To accomplish this we used the co-register facility of SPM. Co-registering the SPMMouse grey matter template against an ABA whole brain template (created from the AtlasAnnotation200.sva data file[43]) gave us the co-registration matrix and re-slicing transformation which we could then apply to each of three SPMs (**Figure 14**).

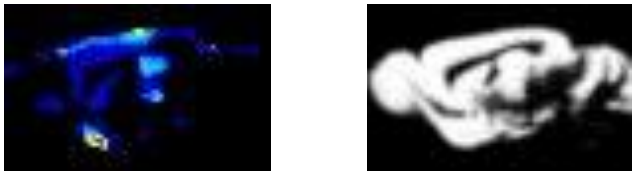


Figure 14 Viewing the atrophy map (ABA co-ordinate system) of 70 day and 84 day (pooled) mice showing basal ganglia atrophy hotspot (left) together with the Grey Matter map for comparison (right)

[43] See Code Listing 10 for full Python script.

With our three SPMs now compatible with ABA's co-ordinate system the final stage was to transform the maps into three .sva files. The 'Nibabel'[44] toolkit in Python allowed us to work with these NifTI format SPMs and was used to generate equivalent ABA .sva files[45]. The following is a sample taken from the one of the three .sva files comprising the pooled data for days 21, 28 and 35:

Comment: Atrophy output.
Dimensions:67,41,58
1,14,24,0.144789
1,14,25,0.494149
1,14,26,0.368675
1,14,27,0.046046
1,14,28,0.010040
1,14,32,0.019487

The format of the file is similar to that of our ABA expression data .sva files, except each voxel has an accompanying atrophy level as opposed to an expression energy level. Maintaining this format provides consistency and allows the data to be manipulated by the same scripts (with a few modifications) used during the creation of region classifiers.

4.5.2 Creating Training & Testing Sets

With atrophy levels now obtained the next phase is to generate our training sets on which to train our SVM atrophy predictors. Each of the three .sva files will be used to create three training files and three unseen testing data sets. This will allow us to create three SVM atrophy predictors in order to conduct spatial and temporal analysis over three stages (resulting from the pool atrophy data) in the imaged R6/2 mice's life span.

To create our training and testing sets in LIBSVM's format the atrophy data was partitioned into training and testing directories using a modified version of our earlier C code[46] (run and executed on my supervisor's machine). The basic principle remained the same but we no longer required an additional

[44] http://nipy.sourceforge.net/nibabel/
[45] See Code Listing 11 for full Python script.
[46] See Code Listing 12 for the full C code.

script to transform the data into LIBSVM's format. The C code generated the training and testing sets in the appropriate format when allocating voxels alternately. Each instance in these data files represents a single voxel as before, composed of feature id, expression value pairs; however each line is now prepended with the atrophy level of that voxel as opposed to a classification label of 1/-1. This allows us to conduct regression analysis. Note again the C code produces a mapping between feature ids and their corresponding image series id via an ExpressionMappingTrainAtrophy.csv file.

4.5.3 Generating the Atrophy Models

Running one of the training sets through LIBSVM highlighted further issues with processing such large quantities of data and the complex nature of regression analysis, requiring us to alter our approach.

Conducting a coarse grid search for linear kernel parameter selection proved unfeasible as did using LIBSVM's "svm-train.exe" tool. This resulted in us moving to using LIBLINEAR[47], a library developed by the same authors of LIBSVM specifically for using the linear kernel when working with millions of instances and features. We again modified our previous Python script[48] to automate the SVM training procedure and subsequent testing to account for the change in tool and removal of the coarse grid search.

[47] http://www.csie.ntu.edu.tw/~cjlin/liblinear/
[48] See Code Listing 13 for full Python script.

5 Results & Conclusions

5.1 LIBLINEAR's Output

After verifying they're in the correct format and performing scaling to [0,1] as per our previous training procedure the three training sets were then passed through LIBLINEAR. The first of these was P21_P28_P35.sva representing the pooled atrophy data for 21, 28 and 35 day old mice. The second, P42_P56.sva representing pooled atrophy data for 42 and 56 day old mice and the third P70_P84.sva for 70 and 84 day old mice.

5.2 Prediction Accuracy

As before cross validation was conducted during training. After generation of our three .model files automated testing was then conducted by using the model files to predict an unseen testing set (using LIBLINEAR's prediction tool as opposed to LIBSVM's svm-predict.exe). To judge the accuracy of the model for regression prediction we use the mean square error (MSE) and correlation squared (or coefficient of determination, R^2). These values are presented in **Table 6**.

Table 6 MSE & correlation^2 for each atrophy predictor

Mouse Age	Cross Validation (Mean Square Error)	Cross Validation (Correlation Squared)	Unseen Testing (Mean Square Error)	Unseen Testing (Correlation Squared)
P21_P28_P35	5.2819	0.626747	4.21201	0.693883
P42_P56	92.4219	0.728511	76.2485	0.772559
P70_P84	211.04	0.722832	176.808	0.764172

As the mice increase in the age the atrophy increases, increasing the values of the data being fitted, therefore we would expect, as is shown in the table, an increase in the MSE. The coefficient of determination R^2 provides us with a measure of the degree to which the predicted output, an atrophy level, can be determined or explained by the input, the gene expression data. Effectively it measures how well our linear model represents our input space. A R^2 value of over 70%, present in mice past 42 days indicates a

strong relationship. Younger than 42 days the R^2 value of above 60% indicates a moderate relationship. The model provides a better fit to the unseen data with the younger mice showing a near strong relationship between expression and atrophy whilst the relationship for the older mice approaches 80%. These accuracies during CV and testing provide sufficient computational validation allowing us to move to the next step of ranking each gene.

5.3 Gene Ranking

Feature selection was completed on the three produced .model files. The result for each .model file was a ranked gene list representing each of the ~20,000 genes predictive power in terms of atrophy attributed to HD.

Table 7 Top 5 genes for P21_P28_P35

genesymbol	ABS_Weight	genename
Sema4d	19.73579376	sema domain, immunoglobulin domain (Ig), transmembrane domain (TM) and short cytoplasmic domain, (semaphorin) 4D
Glra3	12.31229325	glycine receptor, alpha 3 subunit
Sp3	10.49656554	trans-acting transcription factor 3
Clca2	7.993665695	chloride channel calcium activated 2
Ikzf1	6.648397977	IKAROS family zinc finger 1

Table 8 Top 5 genes for P42_P56

genesymbol	ABS_Weight	genename
Glra3	49.36851812	glycine receptor, alpha 3 subunit
Ikzf1	37.14280794	IKAROS family zinc finger 1
Sp3	32.07063866	trans-acting transcription factor 3
Sema4d	31.70242245	sema domain, immunoglobulin domain (Ig), transmembrane domain (TM) and short cytoplasmic domain, (semaphorin) 4D
Clca2	25.31800166	chloride channel calcium activated 2

Table 9 Top 5 genes for P70_P84

genesymbol	ABS_Weight	genename
Glra3	67.34442496	glycine receptor, alpha 3 subunit
Sp3	55.29392822	trans-acting transcription factor 3
Grin3b	45.27067409	glutamate receptor, ionotropic, NMDA3B
Sema4d	37.2552434	sema domain, immunoglobulin domain (Ig), transmembrane domain (TM) and short cytoplasmic domain, (semaphorin) 4D
Clca2	36.77811468	chloride channel calcium activated 2

Table 7, **Table 8** and **Table 9** present the top five ranked genes for our three time points. The gene symbol and gene names were again determined using our mapping file, ExpressionMappingTrainAtrophy.csv along with the gene – series-map.tsv file from the ABA.

Sema4d is present in all three predictions; this is highly significant with the likelihood of this gene being picked from ~20,000, more precisely 25,551 other genes, minute. Similarly Glra3, Sp3 and Clca2 are all also present in all three predictions. With four of the top five genes identical between three different data sets our model is detecting a significant statistical relationship as opposed to something random. The significance of these biological findings; particularly a link between Sp3 and HD and Glra3 and HD are supported in the literature with a number of papers[49] addressing such a relationship.

5.4 Conclusion & Future Work

5.4.1 Difficulties

One of the greatest difficulties encountered in this project was the non-uniform representation of information. Most of the project's duration focused on transforming our three data sets, the ABA gene expression data, our MRI scans and resulting atrophy data into a more compatible format.

Additionally there was no guarantee of success with our machine learning approach, therefore it was necessary to validate the feasibility of our approach by first generating region classifiers.

However the high accuracy of the region classifiers in predicting unseen data resulted in a validated machine learning approach on which we could then base the training procedure for an atrophy predictor.

The next difficulty which placed great strain on our resources was the size of the data we were working with, hence the need to work jointly with my

[49] A paper addressing the link between Sp3 and HD is available at http://www.plosone.org/article/info:doi/10.1371/journal.pone.0014311
A paper addressing the link between Glra3 and HD is available at http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2776491/

supervisor using his machine. Region classifier model generation using an RBF and linear kernel for five regions took 10 hours on Dell Precision T5500 machine with 24Gb of RAM and four 6-core Xeon processors. This prompted the need for an adjustment in our training procedure, moving to the linear kernel with just a coarse grid search and resulted in 24Gb of training and testing data processed in 25 hours for creating classifiers for all regions.

Moving towards atrophy prediction required 6Gb of image analysis taking over 8 hours not including time required to align images and adjust the co-ordinate systems of SPMMouse's template to match that of the ABA's.

Working with millions of features and instances forced a change from LIBSVM to LIBLINEAR to conduct machine learning. The results however are computationally valid models expressing a strong relationship between gene expression and atrophication.

5.4.2 Closing Remarks

With computationally validity secured through high accuracy on region classification and atrophy prediction it appears in short, that yes, gene expression does appear to be related to atrophy level. The agreement in the three time stages of four out of the top five genes is highly significant and potentially very exciting and forms a basis for future work.

5.4.3 Future Work

With more time the project can be extended further with a more detailed analysis of both the machine learning results from region classification and atrophy prediction. A substantial literature review on the gene ranking for region classification would provide further validation of our techniques and make for an interesting discussion on the biology of each region dependent upon the expression levels of its constituent genes.

The most exciting aspect however would involve looking more closely at the top five genes, perhaps even top ten which have been ranked as a result of feature selection completed on our atrophy predictors. A strong relationship

indicated by the coefficient of determination between gene expression and atrophy prompts a more detailed look at the biology of these genes and could shed further light on the mechanisms of atrophy attributed to HD.

Appendices & Code Listing

Alexandra Browne
Dr Kevin Bryson

Predicting region specific atrophy levels in the brain due to Huntington's Disease through gene expression data

1 Aim

To determine whether the level of atrophy within Huntington's disease is related to the genes being expressed by the different neural tissues. To test this hypothesis we will try to predict the spatial and temporal pattern of atrophy using only the spatial gene expression features.

2 Overview

Huntington's disease is a late-onset inherited neurodegenerative disorder which causes specific atrophy over time within particular regions of the brain. There exists a mouse model of this disease called the R6/2 mouse model – which mimics the neurodegeneration of the human disease within the mouse brain. MRI imaging data-sets exist where MRI imaging have been done over time for control and R6/2 mice (Aggarwal et al. 2012; Zhang et al. 2010). This reveals the disease causes atrophy within specific regions of the brain, such as the striatum, which increases over time. This MRI data thus allows one to determine and analyze the spatial occurrence of brain tissue atrophy in Huntington's disease over time.

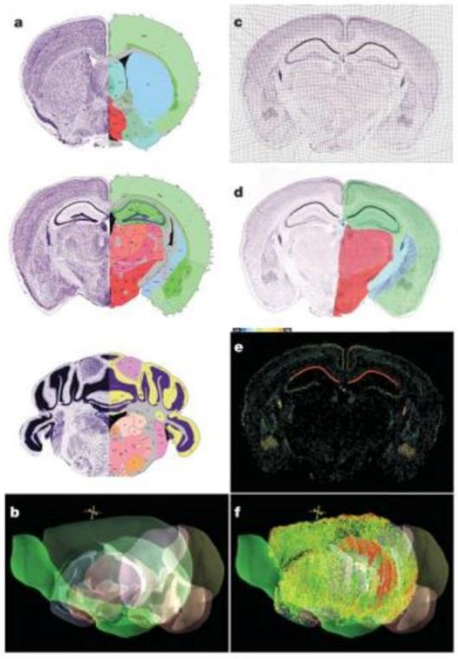


Figure 1

Another valuable but disconnected dataset is the Allen Brain Atlas map of gene expression within the mouse brain (Lein et al. 2007; Jones et al. 2009). This is based on profiling around 20,000 genes in slices of the brain and then rendering thee in 3D (Figure 1).

Three key models of brain atrophy within neurodegenerative diseases can be postulated:

1. The atrophy is entirely dependent on the tissue with each region of the brain being independent of other regions.
2. The atrophy starts in one region of the brain and then spreads to other regions similarly to some amyloid diseases such as prion diseases and ALS – hence the temporal model would include this spatial spreading.
3. The atrophy starts in one region of the brain and then spreads along fibres of neurons to other regions of the brain. This would require 'brain region connectivity' information to be included which is also available from the Allen Brain Atlas site.

We would hope to distinguish between these three different hypothesis for Huntington's Disease by developing the relevant machine learning predictors based on gene

expression, gene expression with spatial closeness information and gene expression with region connectivity information.

3 Objectives

3.1 Extend the Statistical Parameter Map (SPM 2012) and SPMMouse (SPMMouse 2012) software to handle spatial gene expression profiles of mice brains.

3.2 Using this extension, develop a machine learning approach to predict atrophy of the brain in Huntington's disease based on gene expression levels in different parts of the brain.

4 Deliverables

4.1 A MATLAB package which allows the Allen Brain Atlas to be read and processed by SPM.

4.2 An SPM extension that is a self –contained additional package (i.e. SPMExpression module) that can be published to the research community.

4.3 An appropriate machine learning model which can predict the time-course of atrophy in particular regions of the brain based entirely on the gene expression levels within that region.

4.4 A publishable research paper detailing the SPM extension and prediction results if successful.

5 Work plan

Project start to end November: Review background information on biology and Huntington's disease. Review background information on MRI imaging and the Statistical Parameter Mapping approach to its analysis. Review background information on Allen Brain Atlas. Download SPM version 5 and SPMMouse, together with the R6/2 mouse data. Do the SPMMouse tutorial to get familiar with prediction of levels of atrophy from MRI data.

Start of December to mid-December: Import the temporal R6/2 mouse dataset into SPM, will involve quite complex integration with SPMMouse since these are two distinct data-sets).

Milestone 1: Mid December, successfully import temporal R6/2 mouse dataset into SPM.

Mid December to end of December: Recapitulate the findings of Zhang et al. (2010) by analyzing the temporal R6/2 data according to the scheme outlined in the SPMMouse tutorial. Examine the Allen Brain Atlas Resource and the data formats it uses for the mouse transcriptomics data and brain region connectivity data.

January to end January: Develop a MATLAB package to read this data across the network into an appropriate spatial format that integrates with the SPM package (and possibly the bio informatics toolbox for the gene-related data). This may also require appropriate 'spatial normalization' to ensure the spatial gene expression data aligns with the MRI data. Initial prototyping of a simple linear models predicting atrophy against gene expression and time. These most likely would employ lasso or elastic net techniques due to the high number of features compared to dependent variables. Work on interim report.

Milestone 2: End of January, MATLAB package which allows the Allen Brain Atlas data to be read and processed by SPM. Prediction of the spatial and temporal pattern of atrophy based on the type of genes being expressed by the different neural tissues. Interim report.

End January to mid February: Testing of more complex machine learning approaches (SVM, ANN) and benchmarking these different approaches. (With appropriate cross-validation.)

Mid February to end of March: Publication of new SPMExpression toolkit. Development of research publication outlining new package and results. Work on Final Report.

Milestone 3: End of March, SPMExpression toolkit. Research publication. Final Report.

6 References

Aggarwal, M. et al., 2012. Spatiotemporal mapping of brain atrophy in mouse models Huntington's disease using longitudinal in vivo magnetic resonance imaging. *NeuroImage*, 60(4), pp.2086–2095.

Jones, A.R., Overly, C.C. & Sunkin, S.M., 2009. The Allen Brain Atlas: 5 years and beyond. *Nat Rev Neurosci*, 10(11), pp.821–828.

Lein, E.S. et al., 2007. Genome-wide atlas of gene expression in the adult mouse brain. *Nature*, 445(7124), pp.168–176.

SPM, 2012. SPM - Statistical Parametric Mapping. Available at: http://www.fil.ion.ucl.ac.uk/spm/ [Accessed October 13, 2012].

SPMMouse, 2012. SPM Mouse. Available at: http://www.spmmouse.org/ [Accessed November 10, 2012].

Zhang, J. et al., 2010. Longitudinal characterization of brain atrophy of a Huntington's disease mouse model by automated morphological analyses of magnetic resonance images. *NeuroImage*, 49(3), pp.2340–2351.

Alexandra Browne
Dr Kevin Bryson

Predicting region specific atrophy levels in the brain due to Huntington's disease through gene expression data

1 Changes to project plan

1.1 Change from SPM and MATLAB to R

Originally our objectives and deliverables included a self-contained SPM extension able to handle spatial gene expression profiles of mice brains. A machine learning approach would then be developed utilizing this extension with the aim of predicting region specific atrophy levels through gene expression data.

However due to SPMMouse no longer being developed as well as compatibility issues, we moved away from the SPM and SPMMouse software and the minefield that is using commercial software such as MATLAB and focused our attention on the statistical programming language R.

The open source language R allows us to maintain our original goal of developing software and scripts all under an open source license, and gives us greater variety in the choice of additional packages available for our project as these have been developed by members of the open source community. However choosing R meant help and support was limited. This aside, R felt the better choice for the additional reason that it is the language of choice for clinicians working in similar research.

1.2 New objectives

1.2.1 Utilize the statistical programming language R to register our MRI images with a reference model in order to localize specific regions of the brain.
1.2.2 Develop an R package able to read the Allen Brain Atlas and register the gene expression data with our structurally labelled MRI images.
1.2.3 Using this R package to develop a machine learning approach to predict atrophy of the brain in Huntington's disease based on gene expression levels in different parts of the brain.

1.3 New deliverables

1.3.1 An R script able to label specific regions of the brain from an MRI image.
1.3.2 An R package which allows the Allen Brain Atlas to be read and processed by R.
1.3.3 An appropriate machine learning model which can predict the time-course of atrophy particular regions of the brain based entirely on the gene expression levels within that region.
1.3.4 A publishable research paper detailing the R script and package as well as prediction results if successful.

2 Progress made to date

1.2.1 – Software from UCLA's lab of neuroimaging (LONI) has been found which includes an already segmented MRI image of the brain. This is achieved through a label index file and an associated label volume placed over the top of the MRI image of the mouse brain. The label index file contains indices for various co-ordinates of the image; with its associated label volume 127 segmented structures can be identified. It is hoped that by using R's affine registration packages, we can register our own MRI images with the MRI image of this software, thus our own images will be appropriately segmented. Having been unsuccessful with R's package NiftyReg for image affine registration, we are in the process of trying NiPy.

1.2.2 – Although R can be utilized to open and read webpages, python scripts were instead written for their faster performance and portability. Thus gene expression data for all available genes (~25,000) has been downloaded into text files.
Further scripts have been written to divide the data based on brain region, however to achieve this, the brain has been made into a cuboid shape to account for dis-connectivity between the voxel coordinates supplied in the gene expression files and the coordinates in the brain region files. This data is being used to test machine learning techniques at present. However, later on machine learning will take place using the expression data for each region matched by identical voxel coordinates, with any unmatchable data recorded.

1.2.3 - Machine learning techniques have been practiced with the R –interface to package libsvm with the iris data set. The easy.py has then been used as a standalone script with the same data set to gain familiarization with cross validation and scaling. A python script has been developed to take the gene expression data .txt files and produce it in the format required to create training, test and validation sets for libsvm and easy.py.

3 Remaining work

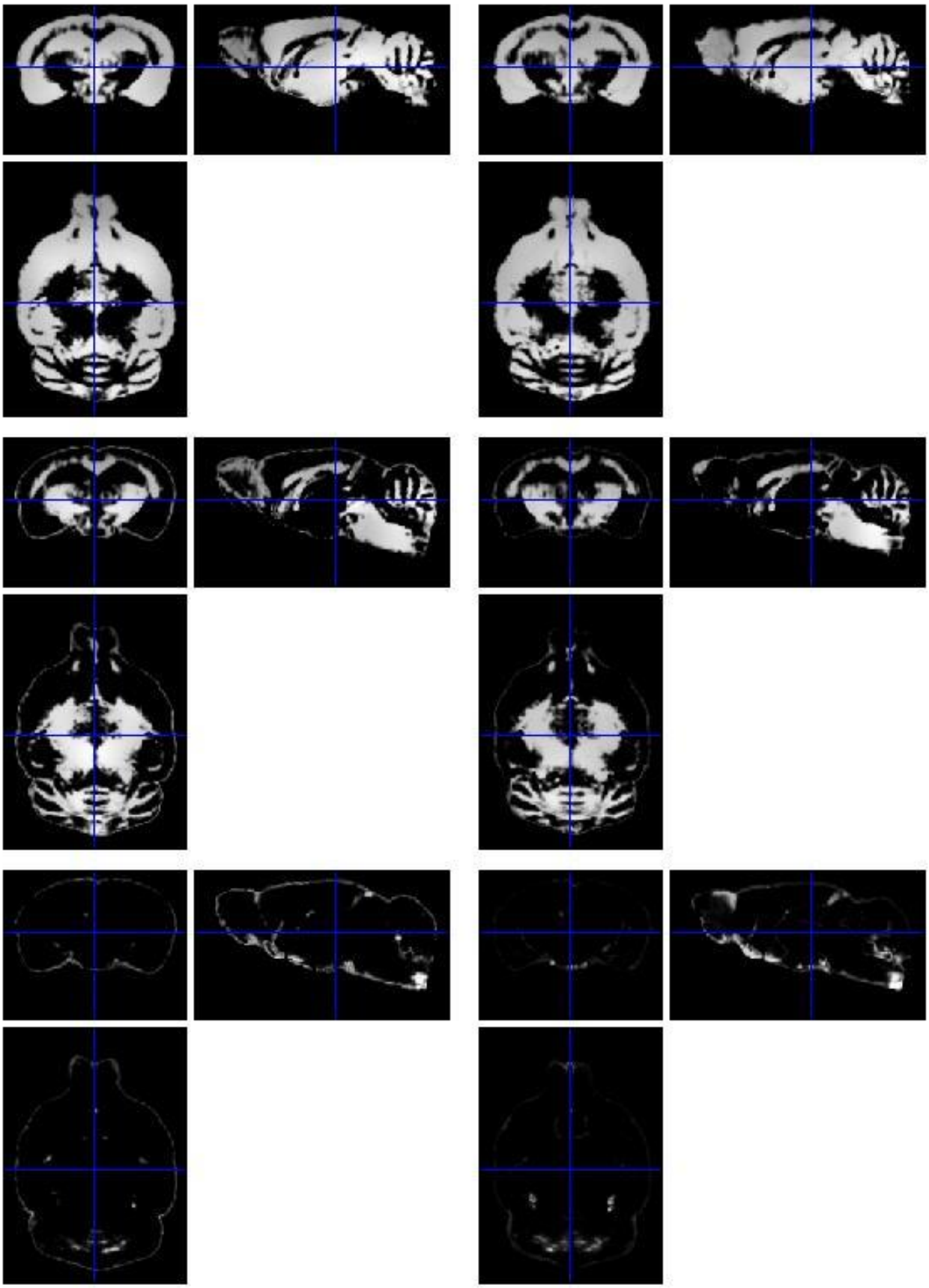
3.1 Work with the package NiPy and successfully segment our MRI images in order to identify the regions of the brain corresponding to those in the ABA data.
3.2 Write new python scripts in order to separate gene expression data into regions based on matching voxel co-ordinates.
3.3 Run python scripts on separated data to convert into format for training, test and validation sets.
3.4 Utilize libsvm and easy.py on training set to create appropriate SVM model for each region to validate end of project results.
3.5 Use appropriate scaling and cross validation in order to take unseen expression data and identify the region it is from.
3.6 Identify and measure atrophy, creating an appropriate R readable model from MRI images.
3.7 Create training, testing and validation sets on atrophy measurements with corresponding expression data and region in order to create new SVM model.
3.8 Use SVM model on unseen atrophy measurements with corresponding gene expression data in order to predict the region of the brain.

Appendix 3 Prediction Accuracy Results for All Region Classifiers

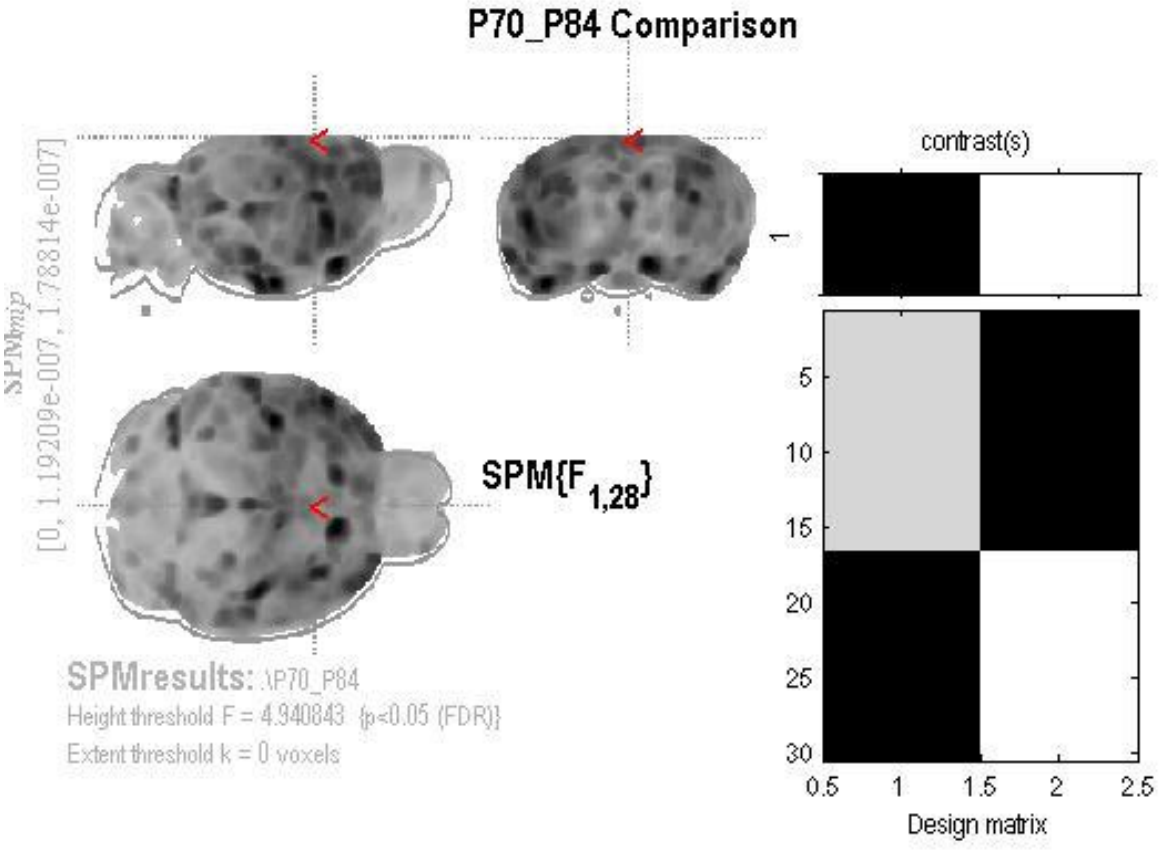
StructureName	Linear CV accuracy	Linear unseen accuracy
Accessory olfactory bulb	100%	99.5885%
Ammon's Horn	98.4772%	98.8136%
Anterior amygdalar area	98.374%	97.9675%
Anterior hypothalamic nucleus	100%	99.5868%
Anterior olfactory nucleus	97.6048%	98.8024%
Anterior pretectal nucleus	98.3607%	98.7705%
Anteromedial nucleus	97.9167%	98.7448%
Anteroventral nucleus of thalamus	99.1525%	98.7288%
Arcuate hypothalamic nucleus	99.1525%	98.7234%
Basic cell groups and regions	90.1454%	92.8918%
Bed nuclei of the stria terminalis	99.6198%	98.4791%
Brain stem	90.9836%	93.0328%
Caudoputamen	98.8208%	99.0566%
Central amygdalar nucleus	97.2656%	97.2656%
Cerebellar cortex	99.1661%	99.6149%
Cerebellum	95.9322%	97.9592%
Cerebral cortex	98.9845%	98.9843%
Cerebral nuclei	86.0902%	91.3534%
Cerebrum	90.625%	94.0972%
Cochlear nuclei	99.2424%	98.8636%
Cortical amygdalar area	97.9094%	97.561%
Cortical plate	97.0588%	97.4684%
Cuneiform nucleus	97.4576%	99.1489%
Dentate gyrus	97.5741%	98.1081%
Dorsal part of the lateral geniculate complex	100%	99.5798%
Dorsomedial nucleus of the hypothalamus	99.1525%	99.1525%
Facial motor nucleus	100%	100%
Fastigial nucleus	99.5745%	99.5726%
Fundus of striatum	97.0213%	99.5745%
Hindbrain	93.6975%	94.958%
Hippocampal formation	96.7078%	96.2963%
Hippocampal region	96.3563%	97.5709%
Hypoglossal nucleus	98.2979%	99.5726%
Hypothalamus	93.5123%	95.5257%
Inferior colliculus	98.1233%	97.8552%
Inferior olivary complex	98.374%	100%
Interbrain	95.339%	96.1864%
Interposed nucleus	98.7705%	98.7654%
Intralaminar nuclei of the dorsal thalamus	97.0213%	97.8723%
Lateral dorsal nucleus of thalamus	98.7903%	99.5968%
Lateral group of the dorsal thalamus	94.8207%	96.8%
Lateral habenula	98.2979%	99.1453%
Lateral posterior nucleus of the thalamus	96.4143%	97.2%
Lateral reticular nucleus	98.7603%	98.7603%
Lateral septal nucleus	98.9726%	99.3151%
Lateral vestibular nucleus	99.5781%	98.7342%
Magnocellular nucleus	98.7234%	99.5726%
Magnocellular reticular nucleus	96.7611%	98.7805%
Main olfactory bulb	99.8578%	99.8575%
Medial amygdalar nucleus	99.2453%	97.7273%
Medial geniculate complex	100%	99.5918%
Medial mammillary nucleus	100%	100%
Medial vestibular nucleus	98.8048%	98.008%
Mediodorsal nucleus of thalamus	98.4127%	99.2032%
Medulla	92.3967%	94.3709%
Medulla_ motor related	94.2623%	94.6721%
Medulla_ sensory related	96.2025%	97.0339%
Midbrain	91.4405%	92.6778%
Midbrain reticular nucleus_ retrorubral area	98.7448%	99.5798%
Midbrain_ motor related	94%	94.4%

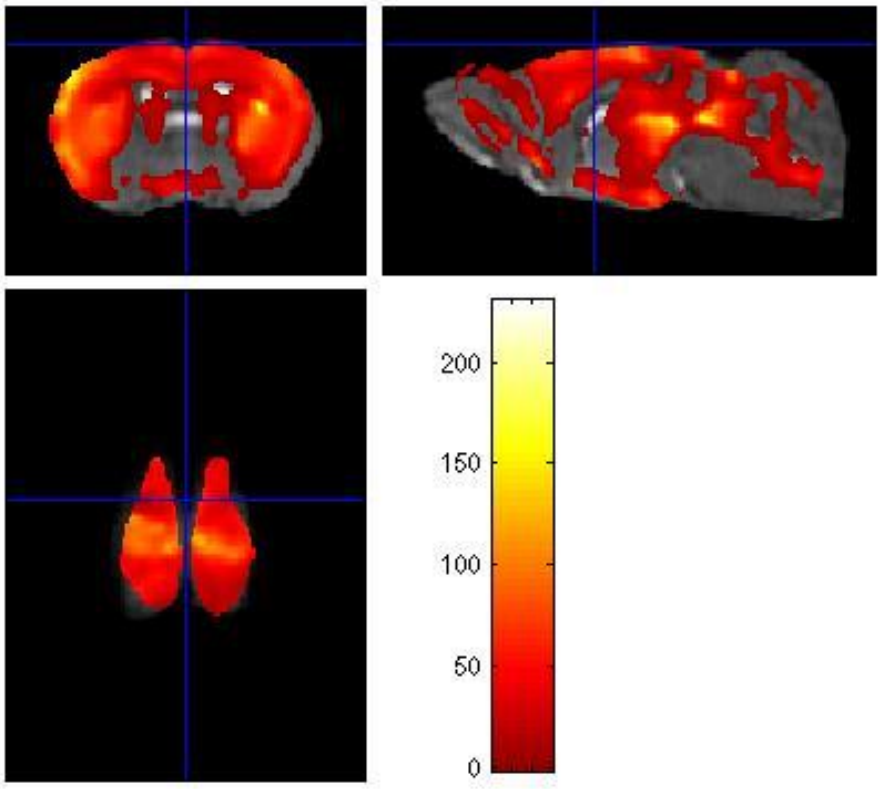
Motor nucleus of trigeminal	100%	100%
Nucleus accumbens	97.7208%	98.8571%
Nucleus of the lateral lemniscus	97.6285%	98.0159%
Nucleus of the lateral olfactory tract	97.8814%	100%
Nucleus of the solitary tract	98.3539%	97.9339%
Olfactory areas	89.4366%	90.8451%
Olfactory tubercle	99.0536%	99.0506%
Pallidum	95.8159%	96.2185%
Pallidum_ dorsal region	98.2206%	99.2857%
Pallidum_ medial region	98.155%	98.893%
Parabrachial nucleus	98.4064%	99.2032%
Parafascicular nucleus	98.3193%	99.1597%
Paragigantocellular reticular nucleus	98.7903%	98.3871%
Pedunculopontine nucleus	98.7342%	99.5781%
Periaqueductal gray	97.971%	98.8372%
Periventricular region	97.8992%	99.1561%
Piriform area	97.8261%	98.5507%
Piriform-amygdalar area	99.5951%	100%
Pons	92.8196%	94.0351%
Pontine central gray	98.7448%	99.5798%
Pontine gray	99.2032%	100%
Posterior hypothalamic nucleus	97.5309%	97.9424%
Postpiriform transition area	97.2%	97.1888%
Principal sensory nucleus of the trigeminal	99.2095%	99.2063%
Red Nucleus	99.5781%	99.5763%
Reticular nucleus of the thalamus	99.6255%	98.1273%
Retrohippocampal region	97.1619%	98.6644%
Spinal nucleus of the trigeminal_ caudal part	98.7705%	99.1803%
Spinal nucleus of the trigeminal_ interpolar part	99.2481%	99.2481%
Spinal nucleus of the trigeminal_ oral part	97.9253%	99.5833%
Striatum	94.8819%	96.4567%
Striatum-like amygdalar nuclei	96.2185%	98.3122%
Subiculum	98.8571%	98%
Substantia innominata	95.6667%	97.3333%
Substantia nigra_ compact part	98.3471%	98.7552%
Substantia nigra_ reticular part	99.6139%	98.4496%
Superior central nucleus raphÚ	99.5885%	99.177%
Superior colliculus_ motor related	98.4694%	98.2143%
Superior colliculus_ sensory related	99.3056%	100%
Superior olivary complex	98.8142%	99.2063%
Taenia tecta	98.8506%	99.6169%
Tegmental reticular nucleus	98.3539%	99.5868%
Thalamus	84.7909%	92.3664%
Thalamus_ polymodal association cortex related	90.8759%	93.4066%
Tuberal nucleus	98.7448%	97.4895%
Ventral group of the dorsal thalamus	96.1832%	96.5517%
Ventral medial nucleus of the thalamus	97.992%	97.1888%
Ventral part of the lateral geniculate complex	98.7395%	99.5798%
Ventral posterior complex of the thalamus	95.3571%	98.2143%
Ventral tegmental area	96.7347%	98.3673%
Ventromedial hypothalamic nucleus	99.1489%	99.1453%
Vestibular nuclei	96.8127%	97.6096%
Zona incerta	95.6364%	97.0803%

Appendix 4 GM (top), WM (middle), & CSF (bottom) Segmentation at 84 Days for Wild-type (left pane) & Mutant (right pane) Mouse



Appendix 5 Example SPM Atrophy Map When Comparing 70 and 84 Day (pooled) Wild-type Mice (n = 16) Against Mutant Mice (n = 14)





Code Listing 1 “Download Gene Exp Data From ABA.py”

```
#Helper method to split excel line from gene-series-map.tsv into individual cells
def split_line(line):
    return line.split()


#Write web page text to file
from BeautifulSoup import BeautifulSoup

def write_expression_info(expressionData, imageSeriesID):
    expressionFile = open("C:\\Users\\Alex\\Desktop\Huntingtons\\"
                          + "Data From Scripts\\Expression Data\\" + str(imageSeriesID) + ".sva", "w")

    for line in expressionData.readlines():
        soup = BeautifulSoup(line)
        body = soup.findAll(text=True)
        expressionFile.write(str(body[0]))


#Access Allen Brain Atlas to download .SVA files through
#http://www.brain-map.org/aba/api/expression/[imageseriesID].sva

import urllib2

def download_gene_expression_data(imageSeriesID):
    URL = "http://www.brain-map.org/aba/api/expression/" + imageSeriesID + ".sva"
    try:
        expressionData = urllib2.urlopen(URL)
        write_expression_info(expressionData, str(imageSeriesID))
    except:
        print("No expression data for " + imageSeriesID)


#Open gene series map to get imageSeriesID'S
f = open("C:\\Users\\Alex\\Desktop\\Huntingtons"
         + "\\Important Data\\gene-series-map.tsv", "r")


#Entry point of script
def start():

    for line in f:
        geneInfo = split_line(line)
        geneInfo.reverse()                  #Reverse list and index from end due to spaces in genename column
        download_gene_expression_data(geneInfo[3])


#Fetch gene expression data
start()
```

Code Listing 2 “Map Brain Regions To CoOrds.py

```
#Container for informatics ID's
informaticIDS = []

#Helper method to split excel line into individual cells
def split_line(line):
    return line.rstrip('\n').split(",")

#Open brainstructures.csv to get informaticsID'S
brainStructures = open("C:\\Users\\Alex\\Desktop\\Huntingtons"
        + "\\Important Data\\brainstructures.csv", "r")


#Open AtlasAnnotation to get co-ordinates and related informaticsID's
def open_atlas():
    atlasAnnotation = open("C:\\Users\\Alex\\Desktop\\Huntingtons"
        + "\\Important Data\\AtlasAnnotation200.sva", "r")
    next(atlasAnnotation)
    next(atlasAnnotation)
    return atlasAnnotation


#Write co-ordinates for each region to separate files
#Alternate between training and testing files.
def create_brain_region_files():
    for i in informaticIDS:

            voxelCount = 0;
            trainingTurn = True;

            brainRegionFileTrain = open("C:\\Users\\Alex\\Desktop\Huntingtons\\"
                + "Data From Scripts\\Brain Regions Train\\" + str(i) + ".txt", "w")
            brainRegionFileTest = open("C:\\Users\\Alex\\Desktop\Huntingtons\\"
                + "Data From Scripts\\Brain Regions Test\\" + str(i) + ".txt", "w")
            atlasAnnotation = open_atlas()
            for line in atlasAnnotation:
                annotationInfo = split_line(line)
                if str(annotationInfo[3]) == str(i):
                    voxelCount += 1
                    if (trainingTurn):
                        brainRegionFileTrain.write(annotationInfo[0] + ',' + ','.join(map(str, annotationInfo[1:3])) + '\n')
                        trainingTurn = False;
                    else:
                        brainRegionFileTest.write(annotationInfo[0] + ',' + ','.join(map(str, annotationInfo[1:3])) + '\n')
                        trainingTurn = True;

            atlasAnnotation.close()
            brainRegionFileTrain.close()
            brainRegionFileTest.close()


#Get informaticID's from brainstructures.csv
def get_informatic_IDS():
    next(brainStructures)
    for line in brainStructures:
        structureInfo = split_line(line)
        informaticIDS.append(structureInfo[6])


#Entry point of script
def start():
    get_informatic_IDS()
    create_brain_region_files()


#Start script
start()
```

Code Listing 3 “Sort Exp Data By Voxel By Region (Non Matrix).py”

```
#Directory for gene ID's to retrieve expression data for
def open_gene_listing():
    expressionLevelsDir = "C:\\Users\\Alex\\Desktop\\Huntingtons\\Data From Scripts\\"
                          + "Complete ABA Expression Data"
    return os.listdir(expressionLevelsDir)


#Get informaticID's which mark regions
def open_region_listing():
    brainRegionDir = "C:\\Users\\Alex\\Desktop\\Huntingtons\\Data From Scripts\\Brain Regions"
    return os.listdir(brainRegionDir)


#Open gene file to retrieve expression data
def open_gene_file(fileName):
    expressionLevelsFile = open("C:\\Users\\Alex\\Desktop\\Huntingtons\\"
                               + "Data From Scripts\\Complete ABA Expression Data\\" + fileName, "r")
    next(expressionLevelsFile) #Skip comment line
    next(expressionLevelsFile) #Skip dimension line
    return expressionLevelsFile


#Helper method to split line into individual coOrds
def split_into_array(coOrdLine):
    return coOrdLine.rstrip('\n').split(",")


#Check if file is empty i.e. there was no data on server
#when file was written to by checking if co-ord can be
#cast to an int
def expression_file_empty(fileCoOrds):
    return fileCoOrds[0].isdigit()


#Write the gene expression levels for the region to file
def write_expression_data(region, imageSeriesID, expressionData):
    expressionDataForRegion =  open("C:\\Users\\Alex\\Desktop\\Huntingtons\\Data From Scripts\\"
                          + "Sorted By Region Output\\" + str(region), "a")
    expressionDataForRegion.write(str(expressionData[0]) + ','
    + ','.join(map(str, expressionData[1:4])) + ',' + str(imageSeriesID[:-4]) + '\n')
    expressionDataForRegion.close()

#Iterate through the gene files finding the voxel co-ordinate that
#matches the co-ordinate passed in and write the expression level to file
import os
def iterate_through_gene_files(region, coOrd):
    listing = open_gene_listing()
    for geneName in listing:
        expressionFile = open_gene_file(geneName)
        for line in expressionFile:
            geneCoOrds = split_into_array(line)
            if expression_file_not_empty(geneCoOrds):
                if int(coOrd[0]) == int(geneCoOrds[0]):
                    if int(coOrd[1]) == int(geneCoOrds[1]):
                        if int(coOrd[2]) == int(geneCoOrds[2]):
                            write_expression_data(region, geneName, geneCoOrds)
                            continue #Found a match so looking at file for next gene
        expressionFile.close()
```

```
#Iterate through the current region file to get co-ordinates to check against
#each gene expression file
def iterate_through_region_file(region, regionFile):
    for coOrdLine in regionFile:
        coOrds = split_into_array(coOrdLine)
        iterate_through_gene_files(region, coOrds)


#Iterate through informaticsIDS (region ID) and open appropriate file
def get_expression_data(regionListing):
    for region in regionListing:
        print("Getting expression data for: " + str(region))
        regionFile = open("C:\\Users\\Alex\\Desktop\\Huntingtons\\"
                + "Data From Scripts\\Brain Regions\\"
                + str(region), "r")
        iterate_through_region_file(region, regionFile)
        regionFile.close()


#Entry point of script
def start():
    regionListing = open_region_listing()
    get_expression_data(regionListing)


#Start script
start()
```

Code Listing 4 “Sort Exp Data By Voxel By Region.py”

```
#This section is concerned with bringing the expression data into
#a matrix in memory for all genes
########################################################################

import os
from numpy import *
import numpy

#Directory for gene ID's to retrieve expression data for
def open_gene_listing():
    expressionLevelsDir = "C:\\Users\\Alex\\Desktop\\Huntingtons\\Data From Scripts"
                          + "\\Complete ABA Expression Data"
    return os.listdir(expressionLevelsDir)


#Open gene file to retrieve expression data
def open_gene_file(fileName):
    expressionLevelsFile = open("C:\\Users\\Alex\\Desktop\\Huntingtons\\"
                               + "Data From Scripts\\Complete ABA Expression Data\\" + fileName, "r")
    next(expressionLevelsFile) #Skip comment line
    next(expressionLevelsFile) #Skip dimension line
    return expressionLevelsFile


#Helper method to split line into individual coOrds
def split_into_array(coOrdLine):
    return coOrdLine.rstrip('\n').split(',')


#Each region is split into voxel dimensions of 67 x 41 x 58
#so create appropriate matrix with values defaulted to 1.0
def create_gene_matrix():
  return ones((67,41,58), float)


#Check if file is empty i.e. there was no data on server
#when file was written to by checking if there is a co-ord
#that can be cast to an int
def exp_file_not_empty(fileCoOrds):
    return fileCoOrds[0].isdigit()


#Iterate through each line in the gene file, extracting the coOrds
#and writing them into a corresponding index in a 3D matrix
def store_exp_data_in_memory():
    geneMap = {}
    geneIDS = open_gene_listing()
    for geneID in geneIDS:
        print("Writing expression data for gene " + geneID + " into memory")
        geneExpFile = open_gene_file(geneID)
        geneMatrix = create_gene_matrix()
        for line in geneExpFile:
            voxelCoOrds = split_into_array(line)
            geneMatrix = set_exp_data(geneMatrix, voxelCoOrds)
        geneMap[geneID] = geneMatrix
        geneExpFile.close()
    return geneMap


def set_exp_data(geneMatrix, voxelCoOrds):
    matrix = geneMatrix
    if exp_file_not_empty(voxelCoOrds):
        x =  int(voxelCoOrds[0])
        y =  int(voxelCoOrds[1])
        z =  int(voxelCoOrds[2])
        geneExp = float(voxelCoOrds[3])
        matrix[x][y][z] = geneExp
    return matrix

########################################################################
#This section is concerned with iterating through the brain region
#files line by line, extracting the co-ordinates on each line and
#for each gene matrix in the previously created map, extracting the
#expression data at those indices.
########################################################################

#Get informaticID's which mark regions
def open_region_listing():
    brainRegionDir = "C:\\Users\\Alex\\Desktop\\Huntingtons\\Data From Scripts\\Brain Regions"
    return os.listdir(brainRegionDir)


#Iterate through informaticsIDS (region ID) and open appropriate file
def get_expression_data_from_memory():
    regionIDS = open_region_listing()
    for region in regionIDS:
        print("Getting expression data for genes for region: " + str(region))
        regionFile = open("C:\\Users\\Alex\\Desktop\\Huntingtons\\"
                + "Data From Scripts\\Brain Regions\\" + str(region), "r")
        iterate_through_region_file(region, regionFile)
        regionFile.close()


#Iterate through the region files to get co-ordinates to check against
#each gene expression file
def iterate_through_region_file(region, regionFile):
    for coOrd in regionFile:
        regionCoOrds = split_into_array(coOrd)
        iterate_through_gene_matrix(region, regionCoOrds)


def iterate_through_gene_matrix(region, regionCoOrds):
    x = int(regionCoOrds[0])
    y = int(regionCoOrds[1])
    z = int(regionCoOrds[2])
    for geneID, geneMatrix in geneIDExpData.items():
        geneExp = geneMatrix[x][y][z]
        write_exp_data(region, geneID, geneExp, regionCoOrds)


def write_exp_data(region, geneID, geneExp, regionCoOrds):
    expressionDataForRegion =  open("C:\\Users\\Alex\\Desktop\\Huntingtons\\Data From Scripts\\"
                    + "Sorted By Region Output\\" + str(region), "a")
    expressionDataForRegion.write(str(regionCoOrds[0]) + ',' +
                                  ','.join(map(str, regionCoOrds[1:])) + ',' +
                                  str(geneExp) + ',' + str(geneID) + '\n')
    expressionDataForRegion.close()

########################################################################
print("********************BEGINNING WRITING GENE EXP DATA INTO MATRICES**********")
geneIDExpData = store_exp_data_in_memory()
print("********************ALL GENES WRITTEN INTO MEMORY**********\n")
print("********************BEGINNING EXTRACTING EXP DATA FOR ALL REGIONS**********")
get_expression_data_from_memory()
print("********************ALL EXP DATA WRITTEN TO FILE FOR EACH REGION**********")
########################################################################

```

Code Listing 5 “SortByVoxel.c”

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <dirent.h>


#define GENE_DIRECTORY "/home/ucackxb/Projects/Project_Students/2012_2013/Alexandra_Browne/BRAIN_EXPRESSION_DATA/RAW_EXPRESSION_DATA_DOWNLOAD"

#define REGION_DIRECTORY "/home/ucackxb/Projects/Project_Students/2012_2013/Alexandra_Browne/BRAIN_EXPRESSION_DATA/Brain Regions Train"

#define MAP_OUTPUT "/home/ucackxb/Projects/Project_Students/2012_2013/Alexandra_Browne/BRAIN_EXPRESSION_DATA/ExpressionMappingTrain.csv"

#define OUTPUT_DIRECTORY "/home/ucackxb/Projects/Project_Students/2012_2013/Alexandra_Browne/BRAIN_EXPRESSION_DATA/Sorted By Region Output Train"

#define PATH_SEPARATOR "/"



typedef struct {
    int matrixIndex;
    char *gene;
}idIndexMap;


int geneNumber = -1;
int idx = -1;
int i, j, k;
int nslices = 67;
int nrows = 41;
int ncols = 58;
int ngenes = 25551;
idIndexMap mapping[25551]; //Hold the mapping between gene and index in matrix
float**** matrix;


DIR *get_geneIDS() {
    DIR *dir = opendir(GENE_DIRECTORY);
    return dir;
}


DIR *get_regionIDS() {
    DIR *dir = opendir(REGION_DIRECTORY);
    return dir;
}

```

```
47 char *get_gene_file_path(char *geneID) {
48     char *path = (char*)malloc(strlen(geneID) + strlen(GENE_DIRECTORY) + 3);
49     strcpy(path, GENE_DIRECTORY);
50     strcat(path, PATH_SEPARATOR);
51     strcat(path, geneID);
52     return path;
53 }
54
55
56 char *get_region_file_path(char *regionID) {
57     char *path = (char*)malloc(strlen(regionID) + strlen(REGION_DIRECTORY) + 3);
58     strcpy(path, REGION_DIRECTORY);
59     strcat(path, PATH_SEPARATOR);
60     strcat(path, regionID);
61     return path;
62 }
63
64
65 char *get_output_file_path(char *regionID) {
66     char *path = (char*)malloc(strlen(regionID) + strlen(OUTPUT_DIRECTORY) + 3);
67     strcpy(path, OUTPUT_DIRECTORY );
68     strcat(path, PATH_SEPARATOR);
69     strcat(path, regionID);
70     return path;
71 }
72
73
```

```
74  void read_gene_file(char *geneID, FILE *fp_map_out) {
75
76      char *id = (char*)malloc(strlen(geneID) + 1);
77      strcpy(id, geneID);
78
79      char *geneFilePath = get_gene_file_path(id);
80      FILE *fp = fopen (geneFilePath, "r");
81      char line[81];
82      fgets(line, 80, fp); //Skip comment line
83      fgets(line, 80, fp); //Skip dimensions line
84
85      char *imageID = (char*)malloc(strlen(id) + 1);
86      strcpy(imageID, id);
87      imageID[strlen(id) - 4] = '\0';
88
89      idIndexMap map = {++geneNumber, imageID}; //Increase the matrix gene number counter and map gene id to matrix index
90      mapping[++idx] = map; //Store this mapping
91      printf("Storing expression data in matrix for gene %s\n", id);
92
93      fprintf(fp_map_out, "%d, %s\n", idx+1, imageID);
94
95      while (fgets(line, 80, fp) != NULL) { //Iterate through every line of the gene file, storing the exp data into the matrix
96        int x, y, z;
97        float exp;
98        x = atoi(strtok(line, ","));
99        y = atoi(strtok(NULL, ","));
100        z = atoi(strtok(NULL, ","));
101        exp = atof(strtok(NULL, ","));
102        matrix[geneNumber][x][y][z] = exp;
103     }
104     fclose(fp);
105
106 }
107
```

```
108
109
110 int write_expression_values(int x, int y, int z, char *id, FILE *fp_out) {
111     int i, matrixIndex;
112     float exp;
113     char *gene;
114
115     fprintf(fp_out, "%d, %d, %d, ", x, y, z);
116
117     for (i=0; i<ngenes; i++) {  //Iterate through mapping, finding index for each gene
118         gene = mapping[i].gene; //Found gene ID
119         matrixIndex = mapping[i].matrixIndex; //Found matrix index for that gene ID
120         exp = matrix[matrixIndex][x][y][z];   //Read expression data for that gene at voxels x, y, z
121         if(exp != 0) {
122             fprintf(fp_out, "%d:%g ", i+1, exp);
123         }
124     }
125
126     fprintf(fp_out, "\n");
127
128     return 0;
129 }
130
131
132
133 void read_region_file(char *regionID) {
134
135     char *id = (char*)malloc(strlen(regionID) + 1);
136     strcpy(id, regionID);
137     char *regionFilePath = get_region_file_path(id);
138     FILE *fp = fopen (regionFilePath, "r");
139
140
141     char *region = (char*)malloc(strlen(regionID) + 1);
142     strcpy(region, regionID);
143     char *outputFilePath = get_output_file_path(region);
144     FILE *fp_out = fopen (outputFilePath, "w");
145
146     char line[81];
147     printf("Reading expression data for region %s\n", id);
148     while (fgets(line, 80, fp) != NULL) {
149       int x, y, z;
150       x = atoi(strtok(line, ","));
151       y = atoi(strtok(NULL, ","));
152       z = atoi(strtok(NULL, ","));
153       write_expression_values(x, y, z, id, fp_out);
154     }
155     printf("Expression data written to file for region %s\n", id);
156     fclose(fp_out);
157     fclose(fp);
158
159     free(id);
160     free(regionFilePath);
161     free(region);
162     free(outputFilePath);
163 }
164
165
```

```
166
167 void create_matrix() {
168     matrix = (float****) malloc(sizeof(float***)*ngenes);
169     for (i=0; i<ngenes; i++) {
170         matrix[i] = (float***) malloc(sizeof(float**)*nslices);
171         for (j=0; j<nslices; j++) {
172             matrix[i][j] = (float**) malloc(sizeof(float*)*nrows);
173             for (k=0; k<nrows; k++) {
174                 matrix[i][j][k] = (float*) malloc(sizeof(float)*ncols);
175             }
176         }
177     }
178 }
179
180
181 void write_exp_data_to_matrix() {
182     struct dirent *geneID;
183     DIR *geneDir = get_geneIDS();
184
185     FILE *fp_map_out = fopen (MAP_OUTPUT, "w");
186
187     while ((geneID = readdir(geneDir)) != NULL) {
188
189         if (strcmp(geneID->d_name, ".") != 0 &&  strcmp(geneID->d_name, "..")) {
190             read_gene_file(geneID->d_name, fp_map_out);
191         }
192     }
193
194     fclose(fp_map_out);
195 }
196
197
198 void write_exp_data_to_file() {
199     struct dirent *regionID;
200     DIR *regionDir = get_regionIDS();
201
202     while ((regionID = readdir(regionDir)) != NULL) {
203
204         if (strcmp(regionID->d_name, ".") != 0 &&  strcmp(regionID->d_name, "..")) {
205             read_region_file(regionID->d_name);
206         }
207     }
208 }
209
210 main() {
211     create_matrix();
212     write_exp_data_to_matrix();
213     write_exp_data_to_file();
214 }
215
```

Code Listing 6 “Create LIBSVM Files.py”

```
1
2  #Set paths
3  sortedExpDataPerVoxelDir = "C:\\Users\\Alex\\Desktop\\Huntingtons\\Data From Scripts"
4                           + "\\Sorted By Region Output Train"
5  trainingFileOutputDir = "C:\\Users\\Alex\\Desktop\\Huntingtons\\Data From Scripts"
6                           + "\\LIBSVM Files Train"
7
8
9  def write_positive_cases_for_region(postiveRegion, trainingSetFile):
10     sortedExpDataPerVoxelPath = sortedExpDataPerVoxelDir + "/" + postiveRegion
11     sortedExpDataPerVoxelFile = open(sortedExpDataPerVoxelPath, "r")
12     read_block_for_positive_voxels(sortedExpDataPerVoxelFile, postiveRegion, trainingSetFile)
13     sortedExpDataPerVoxelFile.close();
14
15
16 def get_info_as_list(infoForVoxel):
17     return infoForVoxel.rstrip('\n').split(",")
18
19
20 def read_block_for_positive_voxels(sortedExpFile, positiveRegion, trainingSetFile):
21
22     for line in sortedExpFile:
23         infoForVoxelAsList = get_info_as_list(line)
24
25         trainingSetFile.write("1 ")
26         trainingSetFile.write(infoForVoxelAsList[3])
27         trainingSetFile.write("\n")
28
29
30 def write_negative_cases_for_region(positiveRegion, trainingSetFile):
31     import os
32     sortedExpDataRegionListing = os.listdir(sortedExpDataPerVoxelDir)
33     for negativeRegion in sortedExpDataRegionListing:
34         if negativeRegion != positiveRegion:
35             print("Adding Negative voxels added to ", positiveRegion, " for ", negativeRegion)
36             read_block_for_negative_voxels(negativeRegion, positiveRegion, trainingSetFile)
37
38
39
40 def read_block_for_negative_voxels(negativeRegion, regionFileName, trainingSetFile):
41
42     negativeCaseFile = open(sortedExpDataPerVoxelDir + "/" +  negativeRegion, "r")
43     negativeRegionLines = negativeCaseFile.readlines()
44     negativeCaseFile.close()
45
46     import random
47     twoRandomLines = random.sample(negativeRegionLines, 2)
48
49     infoForVoxelAsList1 = get_info_as_list(twoRandomLines[0]);
50     infoForVoxelAsList2 = get_info_as_list(twoRandomLines[1]);
51
52     trainingSetFile.write("-1 ")
53     trainingSetFile.write(infoForVoxelAsList1[3])
54     trainingSetFile.write("\n")
55
56     trainingSetFile.write("-1 ")
57     trainingSetFile.write(infoForVoxelAsList2[3])
58     trainingSetFile.write("\n")
59
```

```
60
61 def start():
62     import os
63     sortedExpDataRegionListing = os.listdir(sortedExpDataPerVoxelDir)
64     completedTrainingFiles = os.listdir(trainingFileOutputDir)
65     for region in sortedExpDataRegionListing:
66         if region not in completedTrainingFiles:
67
68             trainingSetFile = open(trainingFileOutputDir + "/" +  region, "w")
69
70             print("Writing positive cases for region: ", region)
71             write_positive_cases_for_region(region, trainingSetFile);
72             print("Positive written")
73             print("Writing negative cases for region: ", region)
74             write_negative_cases_for_region(region, trainingSetFile);
75             print("Negative written")
76
77             trainingSetFile.close()
78
79 start()
80
81
82
```

Code Listing 7 “Automate SVM Creation Stage 1.py”

```
#Set paths for python scripts and executables needed
import os
svmRegionTrainDir = "/home/ucackxb/Projects/Project_Students/2012_2013/Alexandra_Browne/BRAIN_EXPRESSION_DATA/LIBSVMTrain_Stage1"
svmRegionTestDir = "/home/ucackxb/Projects/Project_Students/2012_2013/Alexandra_Browne/BRAIN_EXPRESSION_DATA/LIBSVMTest_Stage1"
regionName = ""
trainingFileDir = ""
svmCreationErrors = ""
trainingFile = ""
gridSearchOut = ""
gridSearchPng = ""
finerSearchOut = ""
finerSearchPng = ""
coarseBestC = float(0)
coarseBestG = float(0)
coarseBestRate = float(0)
finerBestC = float(0)
finerBestG = float(0)
finerBestRate = float(0)

checkData = "/home/ucackxb/Software/libsvm-3.17/tools/checkdata.py"
gridSearch = "/home/ucackxb/Software/libsvm-3.17/tools/grid.py"
scaleData = "/home/ucackxb/Software/libsvm-3.17/svm-scale"
trainData = "/home/ucackxb/Software/libsvm-3.17/svm-train"
classifyNewData = "/home/ucackxb/Software/libsvm-3.17/svm-predict"


```

```
27 #Entry point of script
28 def start():
29     regionListing = os.listdir(svmRegionTrainDir)
30     for listing in regionListing:
31         global regionName, kernel, kernelName
32         regionName = listing
33         print("[INFO] Current region is " + regionName)
34 
35         ####### RBF Kernel.
36         kernel = "2"
37         kernelName = "RBF"
38         set_variables()
39 
40         check_training_file_format()
41         scale_training_file()
42         scale_testing_file()
43 
44 
45         conduct_coarse_grid_search_RBF()
46         get_coarse_best_parameters_RBF()
47         conduct_finer_grid_search_RBF()
48         get_finer_best_parameters_RBF()
49         create_svm_CV_classifier()
50         create_svm_classifier()
51         create_svm_test_classifier()
52 
53         ####### Linear Kernel.
54         kernel = "0"
55         kernelName = "Linear"
56         set_variables()
57 
58         conduct_coarse_grid_search_linear()
59         get_coarse_best_parameters_linear()
60         conduct_finer_grid_search_linear()
61         get_finer_best_parameters_linear()
62         create_svm_CV_classifier()
63         create_svm_classifier()
64         create_svm_test_classifier()
65 
66 
67 
68 def set_variables():
69     global svmCreationErrors
70     global trainingFileDir, testingFileDir, trainingFile, gridSearchOut, gridSearchPng
71     global finerSearchOut, finerSearchPng
72     trainingFileDir = svmRegionTrainDir + "//" + regionName
73     testingFileDir = svmRegionTestDir + "//" + regionName
74     svmCreationErrors = trainingFileDir + "//SVMCreationErrors.txt"
75     trainingFile = trainingFileDir + "//" + regionName + ".txt"
76 
77     gridSearchOut = trainingFileDir + "//" + regionName + "_" + kernelName + "_ScaleGrid.coarse.out"
78     gridSearchPng = trainingFileDir + "//" + regionName + "_" + kernelName + "_ScaleGrid.coarse.png"
79     finerSearchOut = trainingFileDir + "//" + regionName + "_" + kernelName + "_ScaleGrid.finer.out"
80     finerSearchPng = trainingFileDir + "//" + regionName + "_" + kernelName + "_ScaleGrid.finer.png"
81 
82 
83 #Check training file is in correct format using checkdata.py from libsvm
84 def check_training_file_format():
85     print("[INFO] Checking " + trainingFile + " training file " +
86           "is in correct format")
87     os.system("python " + checkData + " " + trainingFile +
88               " > " + svmCreationErrors)
89 
```

```
90
91  #Scale the data in the range [0,1] using svm-scale, save the parameters
92  #these will be used later on to scale testing data
93  def scale_training_file():
94      print("[INFO] Scaling " + regionName + " training file, " +
95          "saving result & paramaters used")
96      os.system(scaleData + " -l 0 -u 1 -s " + trainingFileDir +
97              "//" + regionName + "Scale.params " + trainingFile +  " > " +
98              trainingFileDir + "//" + regionName + ".scale")
99
100
101
102 #Scale the data in the range [0,1] using the saved parameters.
103 def scale_testing_file():
104     print("[INFO] Scaling " + regionName + " testing file.")
105
106     os.system(scaleData + " -r " + trainingFileDir + "//" + regionName + "Scale.params " +
107             testingFileDir + "//" + regionName + ".txt > " +
108             testingFileDir + "//" + regionName + ".scale")
109
110
111
112 #Conduct a coarse grid search, use default c & g parameters
113 #for the RBF Kernel, (cb,ce,cs = -5,15,2 & gb,ge,gs = 3,-15,2)
114 #The grid.py python file has been edited to write the information
115 #sent to the command line (current c & g rate, along with best c, g, & rate)
116 #to the output file specified by -out instead of just current c, g and rate
117 def conduct_coarse_grid_search_RBF():
118     print("[INFO] Conducting course grid search, " + kernelName + " kernel")
119     try:
120         os.system("python " + gridSearch + " -t 2 -svmtrain " + trainData
121         + " -gnuplot null " + trainingFileDir + "//"
122         + regionName + ".scale > " + gridSearchOut)
123
124     except:
125         print("[ERROR] Attempting to handle error")
126         errorFile = open(svmCreationErrors, "a")
127         errorFile.write(RuntimeError.message)
128         errorFile.close()
129
130 def conduct_coarse_grid_search_linear():
131     print("[INFO] Conducting course grid search, " + kernelName + " kernel")
132     try:
133         os.system("python " + gridSearch + "  -log2c  -1,2,1  -log2g  null -t 0 -svmtrain " + trainData
134         + " -gnuplot null " + trainingFileDir + "//"
135         + regionName + ".scale > " + gridSearchOut)
136
137     except:
138         print("[ERROR] Attempting to handle error")
139         errorFile = open(svmCreationErrors, "a")
140         errorFile.write(RuntimeError.message)
141         errorFile.close()
142
143
144 #Read the file specified by -out containing and obtain best c and g
145 #where C = 2**(c) & G = 2**(g)
146 def get_coarse_best_parameters_RBF():
147     gridSearchFile = open(gridSearchOut, "r")
148
149     for line in gridSearchFile:
150         pass
151     last = line
152     bestParameters = last.split(" ")
153     global coarseBestC, coarseBestG, coarseBestRate
154
155     coarseBestC = float(bestParameters[0])
156     coarseBestG = float(bestParameters[1])
157     coarseBestRate = float(bestParameters[2])
158
159
```

```
160 def get_coarse_best_parameters_linear():
161     gridSearchFile = open(gridSearchOut, "r")
162 
163     for line in gridSearchFile:
164         pass
165     last = line
166     bestParameters = last.split(" ")
167     global coarseBestC, coarseBestG, coarseBestRate
168 
169     coarseBestC = float(bestParameters[0])
170     coarseBestG = float(1.0)
171     coarseBestRate = float(bestParameters[1])
172 
173 
174 
175 #Compute log2 of best c and g to get parameters for finer grid search
176 #Set parameters for c and g start, c and g end and the increment for
177 #the finer grid search
178 #Conduct a finer grid search in the neighbourhood of the best c and g
179 def conduct_finer_grid_search_RBF():
180     import math
181     c = math.log(coarseBestC, 2)
182     g = math.log(coarseBestG, 2)
183 
184     cb = str(c-2)
185     ce = str(c+2)
186     cs = str(0.25)
187 
188     gb = str(g-2)
189     ge = str(g+2)
190     gs = str(0.25)
191 
192     try:
193         print("[INFO] Conducting finer grid search, " + kernelName + " kernel")
194         os.system("python " + gridSearch + " -log2c " + cb + "," + ce + "," + cs +
195                 " -log2g " + gb + "," + ge + "," + gs +
196                 " -t 2 -svmtrain " + trainData +
197                 " -gnuplot null " + trainingFileDir + "//" +
198                 regionName + ".scale > " + finerSearchOut )
199     except:
200         print("[ERROR] Attempting to handle error")
201         errorFile = open(svmCreationErrors, "a")
202         errorFile.write(RuntimeError.with_traceback)
203         errorFile.close()
204 
205 
206 def conduct_finer_grid_search_linear():
207     import math
208     c = math.log(coarseBestC, 2)
209 
210     cb = str(c-2)
211     ce = str(c+2)
212     cs = str(0.25)
213 
214     try:
215         print("[INFO] Conducting finer grid search, " + kernelName + " kernel")
216 
217         os.system("python " + gridSearch + " -log2c " + cb + "," + ce + "," + cs +
218                 " -log2g null " +
219                 " -t 0 -svmtrain " + trainData +
220                 " -gnuplot null " + trainingFileDir + "//" +
221                 regionName + ".scale > " + finerSearchOut )
222     except:
223         print("[ERROR] Attempting to handle error")
224         errorFile = open(svmCreationErrors, "a")
225         errorFile.write(RuntimeError.with_traceback)
226         errorFile.close()
227 
228 
```

```
229
230 #Read the file specified by -out and obtain best c and g
231 #where C = 2**(c) & G = 2**(g)
232 def get_finer_best_parameters_RBF():
233     gridSearchFile = open(finerSearchOut, "r")
234     for line in gridSearchFile:
235         pass
236     last = line
237     bestParameters = last.split(" ")
238     global finerBestC, finerBestG, finerBestRate
239
240     finerBestC = float(bestParameters[0])
241     finerBestG = float(bestParameters[1])
242     finerBestRate = float(bestParameters[2])
243
244
245 def get_finer_best_parameters_linear():
246     gridSearchFile = open(finerSearchOut, "r")
247     for line in gridSearchFile:
248         pass
249     last = line
250     bestParameters = last.split(" ")
251     global finerBestC, finerBestG, finerBestRate
252
253     finerBestC = float(bestParameters[0])
254     finerBestG = float(1.0)
255     finerBestRate = float(bestParameters[1])
256
257
258
259 def create_svm_CV_classifier():
260     if coarseBestRate == finerBestRate:
261         paramC = str(coarseBestC)
262         paramG = str(coarseBestG)
263         print("[INFO] Finer grid search has not revealed better parameters")
264     elif coarseBestRate < finerBestRate:
265         paramC = str(finerBestC)
266         paramG = str(finerBestG)
267         print("[INFO] Finer grid search has provided better C & G parameters")
268
269     print("[INFO] Creating the SVM cross-validation classifier for region " + regionName)
270     os.system(trainData + " -c " + paramC + " -g " + paramG +
271             " -t " + kernel + " -v 5 " + trainingFileDir + "//" +
272             regionName + ".scale > " + trainingFileDir + "//" +
273             regionName + "_" + kernelName + ".CV_out")
274
275
276 #Compare CV rates from coarse and finer grid search to obtain c and g
277 #parameters for creating the SVM region classifier
278 #Create the SVM using the best c & g parameters obtained from coarse and
279 #finer grid searches
280 def create_svm_classifier():
281     if coarseBestRate == finerBestRate:
282         paramC = str(coarseBestC)
283         paramG = str(coarseBestG)
284         print("[INFO] Finer grid search has not revealed better parameters")
285     elif coarseBestRate < finerBestRate:
286         paramC = str(finerBestC)
287         paramG = str(finerBestG)
288         print("[INFO] Finer grid search has provided better C & G parameters")
289
290     print("[INFO] Creating the SVM classifier for region " + regionName)
291     os.system(trainData + " -c " + paramC + " -g " + paramG +
292             " -t " + kernel + " " + trainingFileDir + "//" +
293             regionName + ".scale " + trainingFileDir + "//" + regionName + "_" + kernelName + ".model > " +
294             trainingFileDir + "//" + regionName + "_" + kernelName + ".FULL_out")
295
296
```

```
297
298
299 def create_svm_test_classifier():
300
301     print("[INFO] UNSEEN prediction for region " + regionName)
302     os.system(classifyNewData + " " +
303             testingFileDir + "//" + regionName + ".scale " +
304             trainingFileDir +  "//" + regionName + "_" + kernelName + ".model " +
305             testingFileDir +  "//" + regionName + "_" + kernelName + ".predicted > " +
306             trainingFileDir + "//" + regionName + "_" + kernelName + ".UNSEEN_out ")
307
308
309 ########################Begin#########################
310 start()
311
312
```

Code Listing 8 Side By Side Comparison of Changes to “grid.py”
Left column specifies original lines, right column shows changes

```
447  if options.out_pathname:
448      result_file.close()
449  job_queue.put((WorkerStopToken,None))
450  best_param, best_cg  = {}, []
451  if best_c != None:
452      best_param['c'] = 2.0**best_c
453      best_cg += [2.0**best_c]
454  if best_g != None:
455      best_param['g'] = 2.0**best_g
456      best_cg += [2.0**best_g]
457  print('{0} {1}'.format(' '.join(map(str,best_cg)), best_rate))
458
459  return best_rate, best_param
```

```
446  #Don't close file yet, we want to write last line of best c, g and rate
447  #if options.out_pathname:
448  #result_file.close()
449  job_queue.put((WorkerStopToken,None))
450  best_param, best_cg  = {}, []
451  if best_c != None:
452      best_param['c'] = 2.0**best_c
453      best_cg += [2.0**best_c]
454  if best_g != None:
455      best_param['g'] = 2.0**best_g
456      best_cg += [2.0**best_g]
457  #This line prints the best c and g to CLI, we want it in the result file too
458  print('{0} {1}'.format(' '.join(map(str,best_cg)), best_rate))
459  ###########These lines have been added
460  if options.out_pathname:
461      result_file.write('{0} {1}'.format(' '.join(map(str,best_cg)), best_rate))
462      result_file.close()
463  ########################################
464  return best_rate, best_param
```

Code Listing 9 “Automate SVM Creation Stage 2.py”

```
#Set paths for python scripts and executables needed
import os
svmRegionTrainDir = "/home/ucackxb/Projects/Project_Students/2012_2013/Alexandra_Browne/BRAIN_EXPRESSION_DATA/LIBSVMTrain"
svmRegionTestDir = "/home/ucackxb/Projects/Project_Students/2012_2013/Alexandra_Browne/BRAIN_EXPRESSION_DATA/LIBSVMTest"
regionName = ""
trainingFileDir = ""
svmCreationErrors = ""
trainingFile = ""
gridSearchOut = ""
gridSearchPng = ""
finerSearchOut = ""
finerSearchPng = ""
coarseBestC = float(0)
coarseBestG = float(0)
coarseBestRate = float(0)
finerBestC = float(0)
finerBestG = float(0)
finerBestRate = float(0)

checkData = "/home/ucackxb/Software/libsvm-3.17/tools/checkdata.py"
gridSearch = "/home/ucackxb/Software/libsvm-3.17/tools/grid.py"
scaleData = "/home/ucackxb/Software/libsvm-3.17/svm-scale"
trainData = "/home/ucackxb/Software/libsvm-3.17/svm-train"
classifyNewData = "/home/ucackxb/Software/libsvm-3.17/svm-predict"


#Entry point of script
def start():
    regionListing = os.listdir(svmRegionTrainDir)
    for listing in regionListing:
        global regionName
        regionName = listing
        print("[INFO] Current region is " + regionName)

        set_variables()

        check_training_file_format()
        scale_training_file()
        scale_testing_file()


        conduct_coarse_grid_search()
        get_coarse_best_parameters()
        create_svm_CV_classifier()
        create_svm_classifier()
        create_svm_test_classifier()


```

```
49
50 def set_variables():
51     global svmCreationErrors
52     global trainingFileDir, testingFileDir, trainingFile, gridSearchOut, gridSearchPng
53     global finerSearchOut, finerSearchPng
54     trainingFileDir = svmRegionTrainDir + "//" + regionName
55     testingFileDir = svmRegionTestDir + "//" + regionName
56     svmCreationErrors = trainingFileDir + "//SVMCreationErrors.txt"
57     trainingFile = trainingFileDir + "//" + regionName + ".txt"
58
59     gridSearchOut = trainingFileDir + "//" + regionName + "_ScaleGrid.coarse.out"
60     gridSearchPng = trainingFileDir + "//" + regionName + "_ScaleGrid.coarse.png"
61
62
63 #Check training file is in correct format using checkdata.py from libsvm
64 def check_training_file_format():
65     print("[INFO] Checking " + trainingFile + " training file " +
66         "is in correct format")
67     os.system("python " + checkData + " " + trainingFile +
68                 " > " + svmCreationErrors)
69
70
71 #Scale the data in the range [0,1] using svm-scale, save the parameters
72 #these will be used later on to scale testing data
73 def scale_training_file():
74     print("[INFO] Scaling " + regionName + " training file, " +
75         "saving result & paramaters used")
76     os.system(scaleData + " -l 0 -u 1 -s " + trainingFileDir +
77             "//" + regionName + "Scale.params " + trainingFile +  " > " +
78             trainingFileDir + "//" + regionName + ".scale")
79
80
81
82 #Scale the data in the range [0,1] using the saved parameters.
83 def scale_testing_file():
84     print("[INFO] Scaling " + regionName + " testing file.")
85
86     os.system(scaleData + " -r " + trainingFileDir + "//" + regionName + "Scale.params " +
87             testingFileDir + "//" + regionName + ".txt > " +
88             testingFileDir + "//" + regionName + ".scale")
89
90
91
92 def conduct_coarse_grid_search():
93         print("[INFO] Conducting course grid search.")
94     try:
95         os.system("python " + gridSearch + "  -log2c  -1,2,1  -log2g  null -gnuplot null -t 0 -svmtrain " + trainData
96         + " " + trainingFileDir + "//"
97         + regionName + ".scale > " + gridSearchOut)
98
99     except:
100         print("[ERROR] Attempting to handle error")
101         errorFile = open(svmCreationErrors, "a")
102         errorFile.write(RuntimeError.message)
103         errorFile.close()
104
105
106 #Read the file specified by -out containing and obtain best c and g
107 #where C = 2**(c) & G = 2**(g)
108 def get_coarse_best_parameters():
109     gridSearchFile = open(gridSearchOut, "r")
110
111     for line in gridSearchFile:
112         pass
113     last = line
114     bestParameters = last.split(" ")
115     global coarseBestC, coarseBestG, coarseBestRate
116
117     coarseBestC = float(bestParameters[0])
118     coarseBestRate = float(bestParameters[1])
119
```

```
120
121 def create_svm_CV_classifier():
122
123     paramC = str(coarseBestC)
124
125     print("[INFO] Creating the SVM cross-validation classifier for region " + regionName)
126     os.system(trainData + " -c " + paramC +
127             " -t 0 -v 5 " + trainingFileDir + "//" +
128             regionName + ".scale > " + trainingFileDir + "//" +
129             regionName + ".CV_out")
130
131
132 #Compare CV rates from coarse and finer grid search to obtain c and g
133 #parameters for creating the SVM region classifier
134 #Create the SVM using the best c & g parameters obtained from coarse and
135 #finer grid searches
136 def create_svm_classifier():
137
138     paramC = str(coarseBestC)
139
140     print("[INFO] Creating the SVM classifier for region " + regionName)
141     os.system(trainData + " -c " + paramC +
142             " -t 0 " + trainingFileDir + "//" +
143             regionName + ".scale " + trainingFileDir + "//" + regionName + ".model > " +
144             trainingFileDir + "//" + regionName + ".FULL_out")
145
146
147
148
149 def create_svm_test_classifier():
150
151     print("[INFO] UNSEEN prediction for region " + regionName)
152     os.system(classifyNewData + " " +
153             testingFileDir + "//" + regionName + ".scale " +
154             trainingFileDir +  "//" + regionName + ".model " +
155             testingFileDir +  "//" + regionName + ".predicted > " +
156             trainingFileDir + "//" + regionName + ".UNSEEN_out ")
157
158
159 ########################Begin#########################
160 start()
161
162
```

Code Listing 10 “Translate SVA to NiFTI.py”

```
1  #!/usr/bin/python
2
3  #
4  # Translate Allen Brain Atlas SVA files to NifTI formatted files for SPM5 processing.
5  #
6
7  import os
8  import glob
9
10 import re
11 import numpy as np
12 import nibabel as nib
13
14
15 # BRAIN_SYMMETRY=True required for annotation files which only provide half the brain.
16 BRAIN_SYMMETRY = True
17
18 # BINARY_THRESHOLD=True used to binary threshold the values into a 'mask'.
19 BINARY_THRESHOLD = True
20
21 # OUTPUT_FLOAT_32=True used to produce 32-bit floats (instead of 16-bit integers).
22 OUTPUT_FLOAT_32 = True
23
24 # Path to SVA files to convert.
25 path = "C:\\Users\\Alex\\Desktop\\Huntingtons\\SVA_DATA"
26
27
28 print 'Translating Allen Brain Atlas SVA files to NifTI format.\n'
29
30
```

```
for infile in glob.glob(os.path.join(path, '*.sva')):

    print 'Processing File: ', infile, '\n'

    fo = open(infile, "r+")
    header_text = fo.readline();

    image_dimensions = fo.readline();

    dimensions = re.match(r'Dimensions:(\d+),(\d+),(\d+)', image_dimensions, 0)

    print "Image Dimensions", dimensions.group(1), dimensions.group(2), dimensions.group(3)

    # Create Numpty data matrix of this size.
    x_size = int(dimensions.group(1));
    y_size = int(dimensions.group(2));
    z_size = int(dimensions.group(3));

    # Resulting data should either be 32-bit floats of 16 bit integers.
    if (OUTPUT_FLOAT_32):
        data = np.zeros((x_size,y_size,z_size), dtype=np.float32)
    else:
        data = np.zeros((x_size,y_size,z_size), dtype=np.int16)

    count = 0;

    for line in fo:
        expression_line = re.match(r'(\d+),(\d+),(\d+),([.\d|+)', line, 0)
        x = int(expression_line.group(1));
        y = int(expression_line.group(2));
        z = int(expression_line.group(3));
        expression = float(expression_line.group(4))

        # Make binary if required.
        if (BINARY_THRESHOLD and expression > 0.0):
            expression = 1.0

        data[x,y,z] = expression

        # Add symmetric brain hemisphere position if required.
        if (BRAIN_SYMMETRY):
            data[x,y,z_size - z + 1] = expression


    rotation = np.zeros((4, 4))

    print rotation

    # Data voxels are 0.2 mm in size.
    # Need to apply 90 degrees 'roll' and 'yaw'.

    rotation[0,2] = 0.2;
    rotation[1,0] = -0.2;
    rotation[2,1] = -0.2;

    rotation[0, 3] = -5.9;
    rotation[1, 3] = 6.8;
    rotation[2, 3] = 4.2;

    # Create NifTI image and write out.
    img = nib.Nifti1Image(data, rotation)

    header = img.get_header()
    header.set_xyzt_units('mm', 'sec')

    if (BINARY_THRESHOLD):
        img.to_filename(os.path.join(path, infile + '_BINARY.nii'))
    else:
        img.to_filename(os.path.join(path, infile + '.nii'))

    fo.close()
```

Code Listing 11 “Translate NiFTI to SVA.py”

```
#!/usr/bin/python

#
# Translate NifTI files to Allen Brain Atlas SVA format.
#

import os
import nibabel
import glob
import numpy

# Path of NifTI files to translate.

path = "C:\\Users\\Alex\\Desktop\\Huntingtons\\SVA_DATA"


for infile in glob.glob(os.path.join(path, '*.hdr')):

    print "Processing File: ", infile, "\n"

    img = nibabel.load(infile)
    data = img.get_data()

    # Create SVA file and write header.
    sva_output_fh = open(infile + '.sva', 'w')

    sva_output_fh.write("Comment: Atrophy output.\n")
    sva_output_fh.write("Dimensions:%d,%d,%d\n" % data.shape)

    # Write image values to SVA file.
    for x in range(1, data.shape[0]):
        for y in range(1, data.shape[1]):
            for z in range (1, data.shape[2]):
                if (data[x,y,z] != 0.0 and not numpy.isnan(data[x,y,z])):
                    sva_output_fh.write("%d,%d,%d,%f\n" % (x, y, z, data[x,y,z]))

```

Code Listing 12 “SortByAtrophy.c”

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <dirent.h>

#define GENE_DIRECTORY "/home/ucackxb/Projects/Project_Students/2012_2013/Alexandra_Browne/BRAIN_EXPRESSION_DATA/RAW_EXPRESSION_DATA_DOWNLOAD"

#define MAP_OUTPUT "/home/ucackxb/Projects/Project_Students/2012_2013/Alexandra_Browne/BRAIN_EXPRESSION_DATA/ExpressionMappingTrainAtrophy.csv"

#define REGION_DIRECTORY "/home/ucackxb/Projects/Project_Students/2012_2013/Alexandra_Browne/BRAIN_EXPRESSION_DATA/Brain Regions Atrophy"

#define OUTPUT_DIRECTORY_TEST "/home/ucackxb/Projects/Project_Students/2012_2013/Alexandra_Browne/BRAIN_EXPRESSION_DATA/Sorted By Region Atrophy Test"
#define OUTPUT_DIRECTORY_TRAIN "/home/ucackxb/Projects/Project_Students/2012_2013/Alexandra_Browne/BRAIN_EXPRESSION_DATA/Sorted By Region Atrophy Train"

#define PATH_SEPARATOR "/"


typedef struct {
    int matrixIndex;
    char *gene;
}idIndexMap;


int geneNumber = -1;
int idx = -1;
int i, j, k;
int nslices = 67;
int nrows = 41;
int ncols = 58;
int ngenes = 25551;
idIndexMap mapping[25551]; //Hold the mapping between gene and index in matrix
float**** matrix;


DIR *get_geneIDS() {
    DIR *dir = opendir(GENE_DIRECTORY);
    return dir;
}


```

```
41 DIR *get_regionIDS() {
42     DIR *dir = opendir(REGION_DIRECTORY);
43     return dir;
44 }
45
46 char *get_gene_file_path(char *geneID) {
47     char *path = (char*)malloc(strlen(geneID) + strlen(GENE_DIRECTORY) + 3);
48     strcpy(path, GENE_DIRECTORY);
49     strcat(path, PATH_SEPARATOR);
50     strcat(path, geneID);
51     return path;
52 }
53
54
55 char *get_region_file_path(char *regionID) {
56     char *path = (char*)malloc(strlen(regionID) + strlen(REGION_DIRECTORY) + 3);
57     strcpy(path, REGION_DIRECTORY);
58     strcat(path, PATH_SEPARATOR);
59     strcat(path, regionID);
60     return path;
61 }
62
63 char *get_output_file_path_train(char *regionID) {
64     char *path = (char*)malloc(strlen(regionID) + strlen(OUTPUT_DIRECTORY_TRAIN) + 3);
65     strcpy(path, OUTPUT_DIRECTORY_TRAIN );
66     strcat(path, PATH_SEPARATOR);
67     strcat(path, regionID);
68     return path;
69 }
70
71
72 char *get_output_file_path_test(char *regionID) {
73     char *path = (char*)malloc(strlen(regionID) + strlen(OUTPUT_DIRECTORY_TEST) + 3);
74     strcpy(path, OUTPUT_DIRECTORY_TEST );
75     strcat(path, PATH_SEPARATOR);
76     strcat(path, regionID);
77     return path;
78 }
79
```

```
80
81 void read_gene_file(char *geneID, FILE *fp_map_out) {
82
83    char *id = (char*)malloc(strlen(geneID) + 1);
84    strcpy(id, geneID);
85
86    char *geneFilePath = get_gene_file_path(id);
87    FILE *fp = fopen (geneFilePath, "r");
88    char line[81];
89    fgets(line, 80, fp); //Skip comment line
90    fgets(line, 80, fp); //Skip dimensions line
91
92    char *imageID = (char*)malloc(strlen(id) + 1);
93    strcpy(imageID, id);
94    imageID[strlen(id) - 4] = '\0';
95
96    idIndexMap map = {++geneNumber, imageID}; //Increase the matrix gene number counter and map gene id to matrix index
97    mapping[++idx] = map; //Store this mapping
98
99    fprintf(fp_map_out, "%d, %s\n", idx+1, imageID);
100
101    printf("Storing expression data in matrix for gene %s\n", id);
102    while (fgets(line, 80, fp) != NULL) { //Iterate through every line of the gene file, storing the exp data into the matrix
103       int x, y, z;
104       float exp;
105       x = atoi(strtok(line, ","));
106       y = atoi(strtok(NULL, ","));
107       z = atoi(strtok(NULL, ","));
108       exp = atof(strtok(NULL, ","));
109       matrix[geneNumber][x][y][z] = exp;
110    }
111    fclose(fp);
112
113}
114
115
```

```
116int write_expression_values(int x, int y, int z, char *id, float atrophy_value, FILE *fp_out) {
117    int i, matrixIndex;
118    float exp;
119    char *gene;
120
121    fprintf(fp_out, "%f ", atrophy_value);
122
123    for (i=0; i<ngenes; i++) {  //Iterate through mapping, finding index for each gene
124        gene = mapping[i].gene; //Found gene ID
125        matrixIndex = mapping[i].matrixIndex; //Found matrix index for that gene ID
126        exp = matrix[matrixIndex][x][y][z];   //Read expression data for that gene at voxels x, y, z
127        if(exp != 0) {
128            fprintf(fp_out, "%d:%g ", i+1, exp);
129        }
130    }
131
132    fprintf(fp_out, "\n");
133
134    return 0;
135}
136
137void read_region_file(char *regionID) {
138
139    char *id = (char*)malloc(strlen(regionID) + 1);
140    strcpy(id, regionID);
141    char *regionFilePath = get_region_file_path(id);
142    FILE *fp = fopen (regionFilePath, "r");
143
144    char *regionTrain = (char*)malloc(strlen(regionID) + 1);
145    strcpy(regionTrain, regionID);
146    char *outputFilePathTrain = get_output_file_path_train(regionTrain);
147    FILE *fp_out_train = fopen (outputFilePathTrain, "w");
148
149    char *regionTest = (char*)malloc(strlen(regionID) + 1);
150    strcpy(regionTest, regionID);
151    char *outputFilePathTest = get_output_file_path_test(regionTest);
152    FILE *fp_out_test = fopen (outputFilePathTest, "w");
153
154
155    char line[81];
156    printf("Reading expression data for region %s\n", id);
157
158    fgets(line, 80, fp); //Skip comment line
159    fgets(line, 80, fp); //Skip dimensions line
160
161    int train = 1;
162
163    while (fgets(line, 80, fp) != NULL) {
164      int x, y, z;
165      x = atoi(strtok(line, ","));
166      y = atoi(strtok(NULL, ","));
167      z = atoi(strtok(NULL, ","));
168      float atrophy_value = atof(strtok(NULL, ","));
169
170      if (train) {
171       write_expression_values(x, y, z, id, atrophy_value, fp_out_train);
172       train = 0;
173      } else {
174       write_expression_values(x, y, z, id, atrophy_value, fp_out_test);
175       train = 1;
176      }
177    }
178    printf("Expression data written to file for region %s\n", id);
179    fclose(fp_out_train);
180    fclose(fp_out_test);
181    fclose(fp);
182
183    free(id);
184    free(regionFilePath);
185    free(regionTrain);
186    free(regionTest);
187    free(outputFilePathTrain);
188    free(outputFilePathTest);
189}
```

```
190
191
192
193void create_matrix() {
194    matrix = (float****) malloc(sizeof(float***)*ngenes);
195    for (i=0; i<ngenes; i++) {
196        matrix[i] = (float***) malloc(sizeof(float**)*nslices);
197        for (j=0; j<nslices; j++) {
198            matrix[i][j] = (float**) malloc(sizeof(float*)*nrows);
199            for (k=0; k<nrows; k++) {
200                matrix[i][j][k] = (float*) malloc(sizeof(float)*ncols);
201            }
202        }
203    }
204}
205
206
207void write_exp_data_to_matrix() {
208    struct dirent *geneID;
209    DIR *geneDir = get_geneIDS();
210
211    FILE *fp_map_out = fopen (MAP_OUTPUT, "w");
212
213    while ((geneID = readdir(geneDir)) != NULL) {
214
215     if (strcmp(geneID->d_name, ".") != 0 &&  strcmp(geneID->d_name, "..")) {
216          read_gene_file(geneID->d_name,fp_map_out);
217        }
218    }
219
220    fclose(fp_map_out);
221}
222
223
224void write_exp_data_to_file() {
225    struct dirent *regionID;
226    DIR *regionDir = get_regionIDS();
227
228    while ((regionID = readdir(regionDir)) != NULL) {
229
230     if (strcmp(regionID->d_name, ".") != 0 &&  strcmp(regionID->d_name, "..")) {
231          read_region_file(regionID->d_name);
232        }
233    }
234}
235
236main() {
237    create_matrix();
238    write_exp_data_to_matrix();
239    write_exp_data_to_file();
240}
241
```

Code Listing 13 “Automate SVM Creation Stage 3.py”

```
#Set paths for python scripts and executables needed
import os
svmRegionTrainDir = "/home/ucackxb/Projects/Project_Students/2012_2013/Alexandra_Browne/BRAIN_EXPRESSION_DATA/LIBSVM_Atrophy_Train"
svmRegionTestDir = "/home/ucackxb/Projects/Project_Students/2012_2013/Alexandra_Browne/BRAIN_EXPRESSION_DATA/LIBSVM_Atrophy_Test"
regionName = ""
trainingFileDir = ""
svmCreationErrors = ""
trainingFile = ""
gridSearchOut = ""
gridSearchPng = ""
finerSearchOut = ""
finerSearchPng = ""
coarseBestC = float(0)
coarseBestG = float(0)
coarseBestRate = float(0)
finerBestC = float(0)
finerBestG = float(0)
finerBestRate = float(0)

checkData = "/home/ucackxb/Software/libsvm-3.17/tools/checkdata.py"
gridSearch = "/home/ucackxb/Software/libsvm-3.17/tools/grid.py"
scaleData = "/home/ucackxb/Software/libsvm-3.17/svm-scale"
trainData = "/home/ucackxb/bin/train"
classifyNewData = "/home/ucackxb/bin/predict"


#Entry point of script
def start():
    regionListing = os.listdir(svmRegionTrainDir)
    for listing in regionListing:
        global regionName
        regionName = listing
        print("[INFO] Current region is " + regionName)

        set_variables()

        check_training_file_format()
        scale_training_file()
        scale_testing_file()

        create_svm_CV_classifier()
        create_svm_classifier()
        create_svm_test_classifier()



def set_variables():
    global svmCreationErrors
    global trainingFileDir, testingFileDir, trainingFile, gridSearchOut, gridSearchPng
    global finerSearchOut, finerSearchPng
    trainingFileDir = svmRegionTrainDir + "//" + regionName
    testingFileDir = svmRegionTestDir + "//" + regionName
    svmCreationErrors = trainingFileDir + "//SVMCreationErrors.txt"
    trainingFile = trainingFileDir + "//" + regionName + ".txt"

    gridSearchOut = trainingFileDir + "//" + regionName + "_ScaleGrid.coarse.out"
    gridSearchPng = trainingFileDir + "//" + regionName + "_ScaleGrid.coarse.png"


#Check training file is in correct format using checkdata.py from libsvm
def check_training_file_format():
    print("[INFO] Checking " + trainingFile + " training file " +
          "is in correct format")
    os.system("python " + checkData + " " + trainingFile +
              " > " + svmCreationErrors)


```

```
68 #Scale the data in the range [0,1] using svm-scale, save the parameters
69 #these will be used later on to scale testing data
70 def scale_training_file():
71     print("[INFO] Scaling " + regionName + " training file, " +
72         "saving result & paramaters used")
73     os.system(scaleData + " -l 0 -u 1 -s " + trainingFileDir +
74             "//" + regionName + "Scale.params " + trainingFile +  " > " +
75             trainingFileDir + "//" + regionName + ".scale")
76
77
78
79 #Scale the data in the range [0,1] using the saved parameters.
80 def scale_testing_file():
81     print("[INFO] Scaling " + regionName + " testing file.")
82
83     os.system(scaleData + " -r " + trainingFileDir + "//" + regionName + "Scale.params " +
84             testingFileDir + "//" + regionName + ".txt > " +
85             testingFileDir + "//" + regionName + ".scale")
86
87
88
89 def create_svm_CV_classifier():
90
91     print("[INFO] Creating the SVM cross-validation classifier for region " + regionName)
92     os.system(trainData +
93             " -s 11 -v 5 " + trainingFileDir + "//" +
94             regionName + ".scale > " + trainingFileDir + "//" +
95             regionName + ".CV_out")
96
97
98 #Compare CV rates from coarse and finer grid search to obtain c and g
99 #parameters for creating the SVM region classifier
100 #Create the SVM using the best c & g parameters obtained from coarse and
101 #finer grid searches
102 def create_svm_classifier():
103
104     print("[INFO] Creating the SVM classifier for region " + regionName)
105     os.system(trainData +
106             " -s 11 " + trainingFileDir + "//" +
107             regionName + ".scale " + trainingFileDir + "//" + regionName + ".model > " +
108             trainingFileDir + "//" + regionName + ".FULL_out")
109
110
111
112
113 def create_svm_test_classifier():
114
115     print("[INFO] UNSEEN prediction for region " + regionName)
116     os.system(classifyNewData + " " +
117             testingFileDir + "//" + regionName + ".scale " +
118             trainingFileDir +  "//" + regionName + ".model " +
119             testingFileDir +  "//" + regionName + ".predicted > " +
120             trainingFileDir + "//" + regionName + ".UNSEEN_out ")
121
122
123 ########################Begin#########################
124 start()
125
126
```

References

Aggarwal, M. et al., 2012. Spatiotemporal mapping of brain atrophy in mouse models Huntington's disease using longitudinal in vivo magnetic resonance imaging. NeuroImage, 60(4), pp.2086–2095.

Bates & Murphy, 2002. Huntington's Disease (eds Bates GP, Harper PS, Jones AL) 387–426. Oxford, UK: Oxford University Press.

Charles River Laboratories, 2013. Huntington's disease models. Available at: http://www.criver.com/en-US/ProdServ/ByType/Discovery/CNS/Huntingtons/Pages/R62_Mouse.aspx [Access 12 March, 2013].

Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011. Software available at

http://www.csie.ntu.edu.tw/~cjlin/LIBSVM.

Chih-Wei Hsu et al, 2010. A Practical Guide to Support Vector Classification. Department of Computer Science, National Taiwan University, Taipei 106, Taiwan. http://www.csie.ntu.edu.tw/~cjlin.

Chris Thornton. Machine Learning - Lecture 15 Support Vector Machines. Available at: http://www.sussex.ac.uk/Users/christ/crs/ml/lec08a.html [Accessed March 16, 2013].

David Corney, 2002. Food bytes: intelligent systems in the food industry. British Food Journal, Vol. 104 Iss: 10, pp.787 – 805.

HOPES, 2010. Huntington's Outreach Project For Education, At Stanford. An Introduction to Animal Models of Huntington's Disease. Available at: https://www.stanford.edu/group/hopes/cgi-bin/wordpress/2010/07/an-introduction-to-animal-models-of-huntingtons-disease/ [Accessed 12 March, 2013].

Jones, A.R., Overly, C.C. & Sunkin, S.M., 2009. The Allen Brain Atlas: 5 years and beyond. Nat Rev Neurosci, 10(11), pp.821–828.

Landles & Bates, 2004. Huntingtin and the molecular pathogenesis of Huntington's disease. Fourth in molecular medicine review series. EMBO Reports 5: 958–963.

Lein, E.S. et al., 2007. Genome-wide atlas of gene expression in the adult mouse brain. Nature, 445(7124), pp.168–176.

Miler et al, 2010. Divergence of human and mouse brain transcriptome highlights Alzheimer disease pathways. Proc. Natl Acad. Sci. USA 107, 12698–12703.

NHS, 2012. Huntington's disease – Symptoms. Available at: http://www.nhs.uk/Conditions/Huntingtons-disease/Pages/Symptoms.aspx [Accessed 9 March, 2013].

Roberto de Souza, C, 2012. Kernel Functions for Machine Learning Applications. Available at: http://crsouza.blogspot.co.uk/2010/03/kernel-functions-for-machine-learning.html [Accessed March 16, 2013].

Roze, E. et al., 2011. Huntington's Disease and Striatal Signaling. Front Neuroanat. 2011; 5: 55. Published online 2011 August 23. doi: 10.3389/fnana.2011.00055 PMCID: PMC3188786.

Zhang, J. et al., 2010. Longitudinal characterization of brain atrophy of a Huntington's disease mouse model by automated morphological analyses of magnetic resonance images. NeuroImage, 49(3), pp.2340–2351.