Imaging based enrichment criteria using deep learning algorithms for efficient clinical trials in MCI

(Appendix B)

This supplement contains additional technical and experimental details of our proposed algorithm, randomized denoising autoencoders (rDA), a deep learning based framework for problems where data dimensionality d is much larger than the number of examples n (e.g., voxel-wise imaging studies in neuroimaging). We also provide a brief discussion of background material and include references to relevant articles for the interested reader to pursue further. Finally, the document provides a high level overview of Multi-Kernel Learning marker (MKLm) [6], which serves as a baseline existing imaging marker for comparison purposes.

The supplement is organized as follows. Section 1 briefly introduces neural networks and deep learning models, and then describes a family of models known as stacked denoising autoencoders (SDA). Section 2 motivates the formulation in the $d \gg n$ regime for clinical trial design. We describe the technical details, the architecture, training procedure and other implementation issues. We then present the construction of rDA marker (rDAm) derived from the rDA model, and discuss the choice of model hyper-parameters and their influence on rDAm predictions.

1 Neural Networks

An artificial neural network (ANN) transforms a given example (a d-dimensional vector of features) using an interconnected network and the resultant (transformed) outputs may be used within the context of classification or regression. During training time, a set of training examples, $(\mathbf{x}_i, y_i)_{i=1}^n$ are presented to the so-called visible layer of the network which then fine-tunes its weights so that for each example, i, the network's output agrees with the corresponding y_i as best as possible. Once this "back-propagation" learning is done, ANNs can be used for prediction on new test examples, that is, data whose y_is are not known. The output can be a categorical class label or a dependent variable in a regression setting. The transformations applied via ANN weights on the input features (or covariates) are generally non-linear; these are calculated by first computing a linear projection of the input, followed by applying a monotonic squashing function over these projections. More precisely, a *one-layer* non-linear transformation will compute hidden representations $\mathbf{h}_i$ (of some d_1 dimensions, where $i = 1, \ldots, n$) as follows,

$$\mathbf{h}_i = \sigma(\mathbf{W}^l \mathbf{x}_i + b) \tag{1}$$

where $\mathbf{x}_i \in \mathbb{R}^d$ and $(\mathbf{W} \in \mathbb{R}^{d_1 \times d}, b \in \mathbb{R}^{d_1 \times 1})$ are the unknown transformation coefficients (i.e., weights). $\sigma(\cdot)$ is a point-wise monotonic function (referred to as non-linear squashing function). In general, $\sigma(\cdot)$ is given by a sigmoid (i.e., $\sigma(z) = \frac{1}{1+\exp(-z)}$), although other types of functions can be used. An ANN will typically comprise of multiple such transformations, concatenated end to end, which may be thought of as learning an invariant representation of the data (features) that predict y_is. Such ANNs are also referred to as a *multi-layer* neural network (MLNN). For an L-layered MLNN, the transformations are,

$$\mathbf{h}_i^l = \sigma(\mathbf{W}^l \mathbf{h}_i^{l-1} + b^l) \quad \mathbf{h}_i^0 = \mathbf{x}_i \quad l = 1, \ldots, L \quad i = 1, \ldots, n \tag{2}$$

Observe that the hidden representations across multiple levels can be of any length (i.e., dimensionality) $d_l, l = 1, \ldots, L$ (whenever $d_l > d$, the network learns over-representations, and in general, $d_l \leq d$ with decreasing d_l as l increases). Figure 1 shows the architecture of a typical L-layered MLNN, where the inputs $\mathbf{x}_i$ (of length d) are non-linearly transformed over L levels/layers to generate the desired outputs $\mathbf{y}_i$ ($i = 1, \ldots, n$). Layer to layer connections shown in the figure correspond to applying the corresponding transformation, i.e., $(\mathbf{W}^l, b^l)$, followed by point-wise sigmoid non-linearity $\sigma(\cdot)$.

Backpropagation Learning. Learning a MLNN essentially involves computing the transformation coefficients $(\mathbf{W}^l, b^l)$ for $l = 1, \ldots, L-1$, which can be done by comparing the final layer outputs $\mathbf{h}_i^L$ to the desired outputs $\mathbf{y}_i$

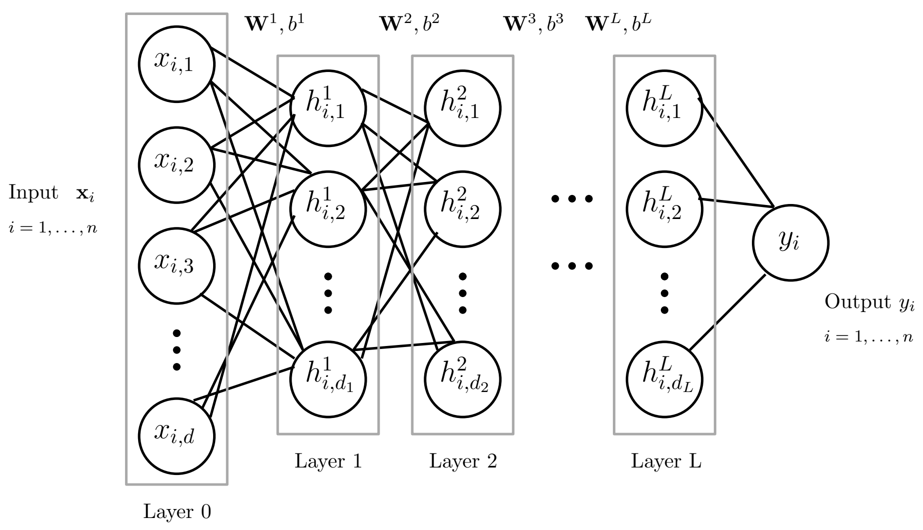


Figure 1: The architecture of a L-layered MLNN transforming the input features $\mathbf{x}_i$ to a desired output $\mathbf{y}_i$. The lengths of hidden layers are $d_1, d_2, \ldots, d_L$. Layer 0 is the visible layer corresponding to the inputs $\mathbf{x}_i$. Layers 1 to L are the L hidden layers, and y_is denote the outputs. Layer to layer transformations are represented by $\mathbf{W}^l$ and b^l. $i = 1, \ldots, n$ and $l = 1, \ldots, L$.

using a loss function $\ell(\cdot)$ (e.g., squared loss or divergences). Although neural networks have attractive theoretical properties (e.g., a 3-layer network can represent any polynomial of arbitrarily high degree, see Chapter 5 in [2]), training them involves a difficult optimization task due to the composition of nonlinear functions $\sigma(\cdot)$. Stochastic minimization methods nevertheless compute a local minima, but these may be sensitive to initialization. The *goodness* of such solutions (in terms of stability to noise, saddle point behaviour, sensitivity to perturbations) depend on the number of stochastic iterations, and in turn, on the number of training samples available to exhaustively search through the solution (hypothesis) space. These issues make efficient learning of MLNNs difficult, which may lead to poor generalization. Beyond this basic overview, additional details of Neural networks can be found in [2].

1.1 Deep Learning

Deep learning refers to a suite of learning algorithms for *efficient* training of large MLNNs by mitigating some of the issues raised previously [1]. These methods learn high-level abstractions of input data, and in turn build complex concepts by composing multiple non-linear transformations of the inputs. Each abstraction corresponds to a single non-linear transformation. *Depth* in deep learning refers to the many levels of transformations (i.e., L is large). To understand the behavior of these algorithms, recall that the main issue with MLNNs was that the multiple non-linear function compositions makes the search space non-convex with many local minima and saddle points. Deep learning algorithms partly mitigate this issue by disentangling the compositions and only working with one-layer (i.e, one transformation) at a time. Working with each layer individually implies that the objective is simpler (albeit, non-convex but likely with fewer local minima). Once *good* estimates of the transformation coefficients are obtained layer-wise, the entire network can then be initialized with these estimates and the resultant final layer predictions can be compared with desired outputs to fine tune these parameters. The basic rationale is that once reasonable enough initializations are obtained, the transformation coefficients are already in some *good* solution bowl in the gradient search space, which may be smoother and better behaved than the ones obtained with random initializations (and/or learning all layers concurrently). Hence, the influence of bad local optima and/or saddle points can be reduced. This two stage procedure of initializing the transformations (referred to as *pretraining*) followed by global fine tuning is a strategy common in deep learning algorithms. The challenge then is to construct efficient algorithms that learn the layer-wise initializations. Several such algorithms have been presented in the literature in the past few years, all of which broadly fall under two categories — restricted Boltzmann machines and autoencoders. Our framework utilizes the latter construct; in this section we briefly introduce autoencoders and their extensions, which form the building block of our method.

1.2 Denoising Autoencoders (DA) and Stacked DA (SDA)

An autoencoder is a single layer neural network (i.e., comprising of one non-linear transformation) that learns robust distributed *representations* of the input data. Given inputs $\mathbf{x}_i, i = 1, \ldots, n$, the autoencoder learns hidden/latent representations $\mathbf{h}_i = \sigma(\mathbf{W}\mathbf{x}_i + b)$, such that the reconstructions $\hat{\mathbf{x}}_i = \sigma(\mathbf{W}^T\mathbf{h}_i + c)$ are as *close* as possible to $\mathbf{x}_i$. More formally, they minimize the following input reconstruction error

$$\mathcal{Z}_a(\{\mathbf{x}_i\}_1^n, \theta) \coloneqq \arg\min_{\mathbf{W},b,c} \sum_{i=1}^{N} \ell(\mathbf{x}_i, \sigma(\mathbf{W}^T\sigma(\mathbf{W}\mathbf{x_i} + \mathbf{b}) + c)), \tag{3}$$

where $\ell(\cdot,\cdot)$ denotes a suitable loss function, e.g., squared loss. A stochastic gradient scheme [3] can be used to perform this minimization. Observe that with no other constraints on $(\mathbf{W}, b, c)$, the above minimization could potentially learn *identity* mappings, i.e., the hidden representations will do nothing and be identical to the inputs. This not only implies that the autoencoder cannot generalize to test instances, but also that the transformation parameters do not correspond to the statistical regularities (i.e., properties of input data distribution) of the inputs. Several approaches have been suggested to ensure that the encoding(s) actually correspond to regularities such as dependencies, correlations and/or linear/non-linear interactions of multiple input features (dimensions). Most methods add a regularization term(s) in the objective in (3) ensuring some structure on the parameters, for example, sparsity over $\mathbf{W}$. Here, we consider an alternative approach where the inputs $\mathbf{x}_i$ are *corrupted* stochastically and the autoencoder is forced to reconstruct the original (non-corrupted) versions. This setup is referred to as *denoising* autoencoder (DA) and the minimization in (3) is modified as

$$\mathcal{Z}_{da}(\{\mathbf{x}_i\}_1^n, \theta) \coloneqq \arg\min_{\mathbf{W},b,c} \sum_{i=1}^{n} \mathbb{E}_{\tilde{x}\sim\gamma(\tilde{\mathbf{x}}|\mathbf{x})}\ell(\mathbf{x}_i, \sigma(\mathbf{W}^T\sigma(\mathbf{W}\tilde{\mathbf{x}}_i + b) + c)) \tag{4}$$

where $\gamma(\cdot)$ is a stochastic corruption function, and $\tilde{\mathbf{x}}_i$ represents the corrupted $\mathbf{x}_i$. $\gamma(\mathbf{x}_{ij}) = \mathbf{x}_{ij}$ with some (given) probability η and 0 elsewhere (here, $j = 1, \ldots, d$ are the input data dimensions). DA is basically a stochastic version of the autoencoder, where in the learning process, it seeks to undo the input corruption process (hence the name, denoising). Not only does the corruption process ensures that identity mappings are not learned, but also forces the learned transformations to correspond to statistical properties of input data distribution. This is because the reconstruction error can only be decreased if the transformations pick out most informative dimensions of the data, and hence the hidden representations are abstract enough to *generate* the inputs. In other words, some irrelevant features that accidentally may seem informative will likely not survive corruption. In general, the informative features are also referred to as the features/factors of variation of input data. If the length of hidden layer is smaller than the size of the inputs, then the representations represent low-dimensional non-linear projections of the corrupted input distribution. Multiple DAs can then be concatenated to construct what is called *stacked* DA (SDA), where the hidden representations of l^{th} DA are the uncorrupted inputs to $(l+1)^{th}$ DA. The learning process then corresponds to

$$\mathcal{Z}_{sda}(\{\mathbf{x}_i\}_1^n, L, \theta) := \sum_{l=0}^{L-1} \mathcal{Z}_{da}(\{\mathbf{h}_i^l\}_1^n, \theta) \ ; \ \mathbf{h}_i^l = \sigma(\mathbf{W}^l\mathbf{h}_i^{l-1} + p^l) \ ; \ \mathbf{h}_i^0 = \mathbf{x}_i \tag{5}$$

It is straight forward to see that the structure of SDA is identical to that of a L-layered neural network. The SDA learning generates L sets of $(\mathbf{W}^l, b^l, c^l)$, which can then be used as initializations of a L-layered MLNN. The final layer representations $\mathbf{h}_i^l$ can then be compared to the desired outputs and the errors can be propagated back to fine-tune the transformation parameters (weights of the network). Figure 1.2 summarizes this DA learning process where the input $\mathbf{x}_i$ is corrupted using $\gamma(\tilde{\mathbf{x}}|\mathbf{x})$ to generate $\tilde{\mathbf{x}}_i$, which is then used to construct $\hat{\mathbf{x}}_i$. By comparing $\hat{\mathbf{x}}_i$ to $\mathbf{x}_i$ using the loss function $\lessdot(\cdot,\cdot)$ we can learn the transformation parameters $\mathbf{W}$, b and c. Learning proceeds layer-wise as shown in Figure 1.2 where the hidden representations learned from l^{th} layer will be the un-corrupted inputs to the $(l+1)^{th}$ layer, hence initializating a L-layered MLNN.

Although the DA setup learns the transformation weights layer-wise, the objective(s) remains non-convex. This non-convexity, together with the stochasticity that comes from the corruption process, require a very large number of gradient search iterations to effectively search through the solution space. Specifically, as the length of input layer increases, the number of data samples required (to ensure that sufficiently many combinations of corrupted dimensions are passed to the objective) increases proportionally. This has been shown in practice where large sample sizes are required to ensure sufficiently small generalization error of the hidden representations [1]. In most computer vision, natural language processing and machine learning problems, one has access to an extremely large set of unlabeled samples (for example, in object recognition, document analysis and so on). Hence, SDAs and other deep learning families yield state-of-the-art results on many benchmark problems, some of which correspond to complex learning tasks (like object recognition). It is reasonable to expect that these methods should be translatable to

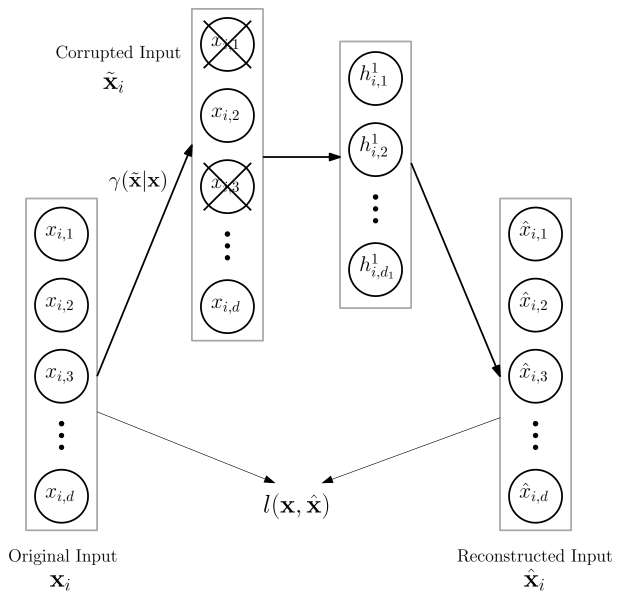


Figure 2: The learning process of DA where the input $\mathbf{x}_i$ is corrupted to generate $\tilde{\mathbf{x}}_i$, which is then used to reconstruct an approximation of $\mathbf{x}_i$ denoted by $\hat{\mathbf{x}}_i$. The crossed-out elements in $\tilde{\mathbf{x}}_i$ represented the stochastically corrupted units i.e., for these units $\tilde{\mathbf{x}}_{i,\cdot} = 0$, and for the rest of the non crossed-out units $\tilde{\mathbf{x}}_{i,\cdot} = \mathbf{x}_{i,\cdot}$. The loss function $\ell(\cdot,\cdot)$ then compares the original $\mathbf{x}_i$ to the reconstruction $\hat{\mathbf{x}}_i$.

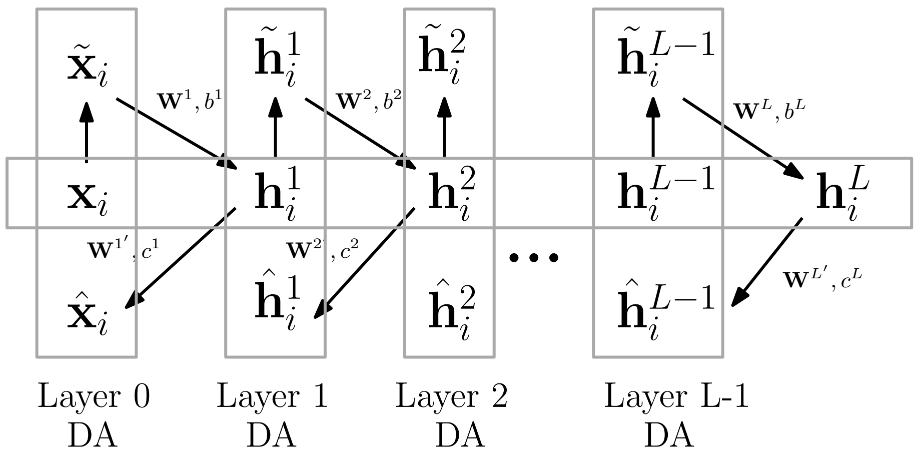


Figure 3: This process of corruption followed by reconstruction of Figure 1.2 is then performed layer-wise, where the hidden representations learned from l^{th} layer will the un-corrupted inputs to the $(l+1)^{th}$ layer.

neuroimaging. However, unlike the applications above, the situation is different in neuroimaging – the number of data dimensions (e.g., voxels) are much larger than the number of available data instances. A typical study will have < 500 subjects while the number of voxels will exceed a million. If SDAs are used in such problems where $d \gg n$, the outputs are not reliable — there is no guarantee that these solutions would be stable (i.e, sensitive to input noise) and/or generalizable. One contribution of the current paper is to translate the success of deep learning models to neuroimaging problems where $d \gg n$. Next, we present a simple strategy to adapt SDAs to our setting where $d \gg n$. When appropriate, we point out specific characteristics that make it suitable for use as an optimal enrichment criterion in imaging based clinical trials in AD.

2 Randomized Denoising Autoencoders (rDA) and clinical trial design

The simplest solution to mitigating the $d \gg n$ issue is by pre-selecting the most influential voxels using a statistical test (e.g., t-test) and using these voxels as features within our machine learning framework downstream. But this idea is lossy in the sense that features that do not survive the feature selection will get discarded, irrespective of how discriminative they are. An alternative is to *categorize* or *tessellate* the entire set of voxels into multiple subsets (e.g., spatially contiguous blocks) and learn a network on each block separately which will then serve as a 'weak' learner (as in boosting [4]). Later, the individual block-wise networks can be combined in a meaningful manner which will yield a better fit with the dependent variable, y. Next, we provide some rationale for why this proposal is viable.

Observe that if the number of voxels in each subset/block is comparable to n (and much smaller than d), the learning problem at the level of *this* block is well defined. For instance, d voxels can be divided into B subsets (or blocks) and we will train SDAs on each block separately. On its own, each block corresponds to only a small number of features and its output will not be useful. But our final output will utilize the contributions of outputs from all B blocks, treating each as a stub (or weak learner). Note that this scenario assumes that the tessellation is given or fixed. But in practice, the "correct" tessellation is not known in advance. However, it is possible to marginalize over this unknown quantity as described below.

Consider the set of all possible tessellations, $\mathcal{C}$. Clearly, this set is exponential sized and running a machine learning model, however simple, for each item in $\mathcal{C}$ is unrealistic. Instead, we can use the standard trick of marginalizing over $\mathcal{C}$ by drawing a large number of samples from it to approximate the summand. In other words, by resampling from $\mathcal{C}$ and deriving many possible B subsets (and learning network weights for them), we can obtain partial invariance to the tesselation by integrating over it. Note that B can be fixed ahead of time and each block can then be constructed arbitrarily (i.e., the process of assigning voxels to blocks) using domain information like neighbourhood interactions and/or consistency of correlations across all d dimensions or any other randomized procedure. One realization of this process corresponds to drawing one sample from $\mathcal{C}$; this provides randomization over clusterings where B outputs are generated for each input instance (we assume that B is sufficiently large). These outputs can then be combined to generate a single classification/regression output depending on the problem at hand.

The above discussion gives a high level overview of the first level of randomization in our algorithm. Before delving into additional details of the method, for presentation purposes, we will now describe the motivating application of designing efficient clinical trial enrichment criterion. This will make it easier to see how a good enrichment criteria can be designed by constructing the block(s) in a specific manner, which will appear shortly.

Sample enrichment: Consider a clinical trial designed to test the efficacy of some treatment (or intervention). The setup includes randomly dividing the population into treatment (intervened) and placebo (non-intervened) arms. If the drug indeed offers an improvement (denoted by η, referred to as the treatment/drug effect), then the treatment and placebo groups should show this change when measured using some outcome measure that summarizes the status of the disease. Assume that the standard change in this outcome measure from the trial start to end point (e.g., 2 years) in the placebo group as measured by this outcome is δ. Then, the treatment group is expected to have this change decreased to $(1-\eta)\delta$, which will correspond to the hypothetical improvement induced by the drug/treatment. Within this scenario, the number of samples s required per arm (treatment and placebo) is given by [8]

$$s = \frac{2(Z_\alpha - Z_{1-\beta})^2\sigma^2}{(1-\eta)^2\delta^2} \tag{6}$$

where $(1-\beta)$ denotes the desired statistical power at a significance level α, and σ^2 denotes the *pooled* variance (the mean of the variances of the outcome at trial start and end points) of the outcome. This expression directly follows from applying two sample t-test at the trial end points, where the Null hypothesis is that the treatment and placebo groups are not different.

Deriving a Minimum Variance Unbiased (MVUB) disease marker: The form in (6) suggests that for a given α and β, the required sample size s *increases* as σ *increases*, and *decreases* as δ *increases*. Alternatively, if the number of subjects s is kept fixed, then increasing σ and/or decreasing δ, will decrease the detectable drug effect η implying that smaller changes can now be detected at the same desired power. Either way, (6) implies that an efficient clinical trial requires large longitudinal changes of the outcome δ *and* small variance σ^2. Assuming that the trial outcome models the disease progression effectively, a large δ implies large changes in outcome during the trial. Ideally, the trial population should *not* include subjects who remain healthy (and/or do not decline) as time progresses. Another way of interpreting this is by observing that the trial population will be corrupted by including healthy and/or *weak decliners* (subjects who show weak characteristics of decline over the trial time period) because these participants will reduce the trial's sensitivity of detecting and quantifying any drug effect. Note that "corrupted" here means that the population is diluted by including those subjects who are unlikely to benefit from the drug, indicating that the alternative hypothesis that the drug induces some effect is moot (for this subgroup). Therefore, we can use an "inclusion criterion" (also referred to as a sample enricher) to filter the trial population, and include *only* those subjects who are likely to decline in the future. The inclusion criterion can be another disease marker (or some summary measure) that predicts future decline with *high* confidence).

Apart from a large δ, (6) also requires a small outcome variance σ^2. However, an inclusion criterion which predicts decline confidently may still give large δ (as described above), but may not necessarily ensure smaller σ^2 for the outcome. To address this, we need to explicitly reduce the outcome variance; this is generally not feasible because the outcomes are cognitive and/or blood flow based measures whose statistical characteristics (range, median, structure etc.) cannot be altered. Instead, we can select an inclusion criterion such that, a reduction of this enricher's variance will indirectly reduce the outcome variance. This can be done by constructing an enricher which has a strong dependence on the intended outcome measure (and hence the disease), and high prediction accuracy of future decline (also, small variance). Observe that if the correlation of this enricher with the outcome is strong, then the low variance characteristic of the enricher translates to the outcome due to the enricher's strong predictability of future decline. Overall, the foregoing discussion implies that the inclusion criterion should have the following properties – (a) strong discrimination power (and therefore *no approximation/modelling bias*) for different stages of the disease like AD, MCI, and so on; (b) strong correlation with an existing disease outcomes or biomarkers (and therefore strong *predictive power* of the disease); and (c) small variance of the outcome that will be used (and therefore small *prediction variance*) We can formalize the above requirements in terms of an estimation problem, where the estimator input can be imaging data (and/or other types of clinical and/or demographic input data) and the estimator output is a *new* disease marker satisfying the above three requirements. No approximation bias and strong predictive power implies that the estimator needs to be unbiased with respect to the classes/labels. Concurrently with the low prediction variance, our problem reduces to constructing a *minimum variance unbiased (MVUB) estimator* of the disease.

Ensemble learning and Randomization: The existence of a MVUB estimator is governed by whether the Cramer-Rao lower bound can be achieved i.e., any unbiased estimator that achieves this lower bound is a MVUB [7]. However, finding such an unbiased estimator can be difficult, and especially in our case where $d \gg n$, computing the lower bound itself is problematic. Instead, an alternative approach to designing such MVUBs is to generate set(s) of unbiased estimators that are approximately *uncorrelated* (in the ideal case, independent) to each other, and compose them to reduce the variance while retaining unbiasedness. The family of models that adapt this approach are broadly referred to as "ensemble learning" methods [4]. The variance reduction behaviour follows from ensuring that the estimators/learners have sufficiently small correlation, then, their linear combination will have smaller variance compared to that of each individual estimator/learner. To see this, let $\mathbf{z}_k$ for $k = 1, \ldots, K$ denote K different random variables (which are the outputs of K different estimators/learners), and $\bar{\mathbf{z}}$ denote their mean. Then, we have

$$\text{Var}(\mathbf{z}_k) = \sigma^2 \quad \text{Cov}(\mathbf{z}_k, \mathbf{z}_{k'}) = \rho \quad \text{then} \quad \text{Var}(\bar{\mathbf{z}}) = \frac{\sigma^2}{K} + \frac{(K-1)\rho\sigma^2}{K} \tag{7}$$

Depending on ρ, the variance of $\bar{\mathbf{z}}$ goes from $\frac{\sigma^2}{K}$ to σ^2 (under the assumption that $\rho > 0$ i.e., the estimators are not negatively correlated). Note that since all the K estimators/learners share the same set of input data, they cannot be independent. However, by reducing ρ and increasing K, we can make sure that the composed estimators (here $\bar{\mathbf{z}}$) have as small variance as possible. Several approaches have been suggested to ensure small ρ, most of which are based on *randomly* dividing the input dimensions, data instances and/or other estimation/learning parameters into K subsets and constructing one estimator from each of these subsets [4]. The outputs of these estimators can then be considered to be random approximations of the ideal output with zero-bias and small variance. Overall, this setup of ensemble learning provides an sub-optimal heuristic approximation of the MVUB estimator, which is sufficient for the purposes of our application.

When $d \gg n$, randomization over the number of data instances might result in estimators using too few examples to learn, and hence may not be un-biased. However, if d is large, it provides a way to randomize over the number of voxels, and generate $K = B$ such sets of voxels of some d_b dimensions each. These B sets of voxels (or blocks/tesselation) will then be passed to SDAs that can learn complex discriminant functions. That is, the outputs coming from these B blocks can discriminate multiple stages/classes of AD (and hence correlate with the disease spectrum). Further, it is known that the SDAs require a few learning parameters like denoising rate ($\gamma(\cdot)$) and gradient learning rate in the SD learning (refer to Section 1.1). This allows for a second level of randomization *with-in* each block, where some $T > 1$ sets of these learning parameters are generated and T SDAs are learned instead of just one. This not only increases the total number of SDAs that will be learned from B to $B \times T$ but also ensures that the influence of learning parameters is mitigated (we see this behavior in our experiments). This strategy of two level randomization over voxels and the learning parameters is referred to *randomized* denoising autoencoders (rDA). Recall that (7) suggests that the composition should have a very large number of estimators with small correlation. This is ensured by fixing B and T ahead of time to be reasonably large and d_b to be sufficiently small (on the order of n). More details about setting up these hyper-parameters are given in Section 2.3. Overall, the outputs of rDA are (heuristic/approximate) MVUB estimators of the disease whenever the $B \times T$ SDAs are trained to predict multiple stages of the disease. Figures 2 shows that rDA outputs indeed have smaller variance relative to a baseline on a real experiment on ADNI 2 data. Here, disease markers are constructed using rDA and a SVM based Multi Kernel models (as will be described in Sections 2.1, 2.2 and Section 3), and their coefficient of variation (CV, ratio of standard deviation to mean) is compared on MCI (left plot) and Family History positive (right plot) subjects respectively. Yellow bars corresponding to rDA outputs have smaller CV compared to MKL outputs. Further details

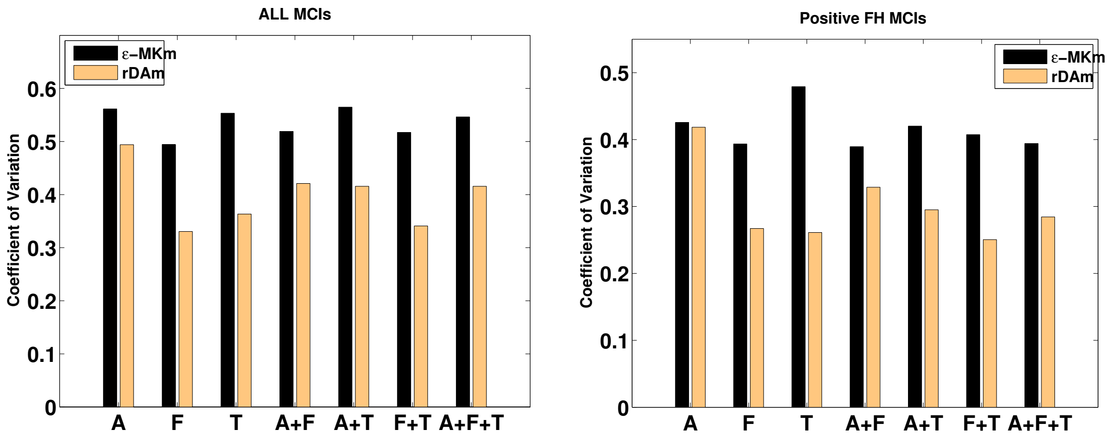


Figure 4: Coefficient of Variation of rDA outputs vs. MKL outputs. Left plot corresponds to the outputs computed using MCI subjects, and right plot corresponds to using Family History (maternal and/or paternal) positive. The x-axis within each plot corresponds to multiple combinations of imaging modalities that will be used as inputs to the rDA and MKL models: A – AV45 PET, F – FDG PET and T – MRI PET

about constructing a disease marker out of this rDA setup, including its architecture and training are discussed next.

2.1 rDA construction and training

Let $\mathcal{V} = \{v_1, \cdots, v_d\}$ give the set of voxels and $\tau(v)$ denote a probability distribution over $v \in \mathcal{V}$. In the simplest case, this is a uniform distribution. For each of the block $b = 1, \ldots, B$, we sample $d_b \ll d$ (fixed a priori) number of voxels without replacement using the distribution $\tau(v)$. We call this set of voxels, s_b. Each block is presented with T different sets of learning parameters (i.e., denoising rate, gradient learning rate and so on) denoted by $\theta_t \in \Theta$ for $t = 1, \ldots, T$ where Θ is the given hyper–parameter space. This means that each sample from the hyper-parameter space yields one weak learner (i.e., one SDA). Hence, a total of $B \times T$ number of SDAs are constructed via randomization over set(s) of voxels that are processed per block and set(s) of learning parameters. Figure 5(a) shows the structure of rDA with B blocks, each with T weak learners, where each weak learner corresponds to a L-layered SDA (as shown in Figure 5(b)). If the sampling distribution $\tau(v)$ is uniform, then asymptotically we expect to see one voxel in at least one of the B blocks. Note that the blocks may not be mutually exclusive and this depends entirely on the sampling distribution $\tau(\cdot)$. Instead of a uniform distribution, alternative choices of $\tau(v)$ can be used depending on what prior information about the "importance" of each voxel to be included in one/more blocks is available. The influence of hyper parameters (B, T, the number of voxels per block d_b, $\tau(\cdot)$) and the robustness of rDA to these choices is described in Section 2.3.

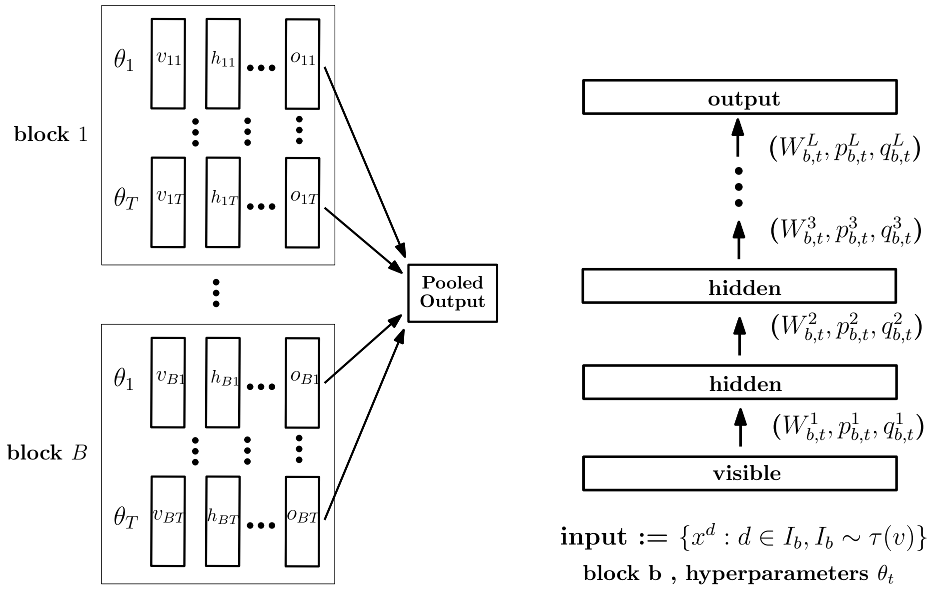
The overall architecture is simply an ensemble of $B \times T$ SDAs (Figure 5(a)) whose outputs are combined using ridge regression. Given training data denoted by $(\mathbf{x}_i, \mathbf{y}_i)$ for $i \in \{1, \ldots, n\}$, we first learn the transformations

$$(\mathbf{W}^l_{b,t}, p^l_{b,t}, q^l_{b,t}) \quad \forall b \in \{1, \ldots, B\} \quad t \in \{1, \ldots, T\} \quad l \in \{1, \ldots, L\}$$

Since each weak learner is SDA, the learning process for these $B \times T \times L$ transformations will follow the minimization of (4) and (5). Hence, each of the $B \times T$ weak learners will be pretrained as in (5), followed by fine tuning using the desired training outputs $\mathbf{y}_i$ for $i \in \{1, \ldots, n\}$. This process of rDA training the $B \times T$ learners is summarized as a pseudo-code in Algorithm 2.1.

The *Reweigh*$(\cdot)$ operation in the above algorithm ensures that un-sampled voxels (i.e., voxels that are not assigned to any block yet), will be given priority in the later blocks, and avoid oversampling same set(s) of voxels across multiple blocks. The simplest such re-weighting includes removing the already sampled voxels from $\tau(\cdot)$. Concatenating the final L^{th} layered outputs, we get $\mathbf{H}_i = [[\mathbf{h}^L_{b,t}]]^{B,T}_{1,1}$. The weighted regression pooling then composes all these $B \times T$ outputs to generate a single final output.

$$\mathbf{U} \leftarrow (\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I})^{-1}\mathbf{H}^T\mathbf{Y} \quad ; \quad \mathbf{H} = [[\mathbf{H}_i]]^n_1 \quad ; \quad \mathbf{Y} = [[\mathbf{y}_i]]^n_1 \,, \tag{8}$$



block 1
block B
θ_1
θ_T
v_{11}
h_{11}
o_{11}
v_{1T}
h_{1T}
o_{1T}
v_{B1}
h_{B1}
o_{B1}
v_{BT}
h_{BT}
o_{BT}
Pooled Output
output
$(W^L_{b,t}, p^L_{b,t}, q^L_{b,t})$
$(W^3_{b,t}, p^3_{b,t}, q^3_{b,t})$
hidden
$(W^2_{b,t}, p^2_{b,t}, q^2_{b,t})$
hidden
$(W^1_{b,t}, p^1_{b,t}, q^1_{b,t})$
visible
input := $\{x^d : d \in I_b, I_b \sim \tau(v)\}$
block b , hyperparameters θ_t

Figure 5: The architecture of rDA is an ensemble of BT weak learners, where each weak learner is a L-layered SDA. Each block processes fixed set of voxels s_b of length d_b. Within each block there are T number of SDAs.

Algorithm *rDA Blocks training*

```
Input: $\theta_t \sim \Theta, \mathcal{V}, B, s_B, L, T, \mathcal{D} \sim \{\mathbf{x}_i, \mathbf{y}_i\}_1^n, \lambda$
Output: $(\mathbf{W}^l_{b,t}, p^l_{b,t}, q^l_{b,t})$
    for $b = 1, \ldots, B$ do
      $I_b \sim \tau(\mathcal{V})$
      for $t = 1, \ldots, T$ do
        $(\mathbf{W}^l_{b,l}, p^l_{b,l}, q^l_{b,l}) \leftarrow \mathcal{Z}_{sda}(\mathcal{D}, L, I_b, \theta_t)$
      end for
      $\tau(\mathcal{V}) \leftarrow Reweigh(\tau(\mathcal{V}), I_b)$
    end for
```

where $\mathbf{U}$ are the regression coefficients, λ is the regularization constant and $\mathbf{Y} = [\mathbf{y_i}]_1^n$.

The pooling operation is simple a linear combinations of $B \times T$ outputs from the SDAs. Note that instead of regression, a simple mean of these outputs will also suffice to generate the final output. But since SDAs are already capable of learning complex concepts (see Section 1), we use a simple linear combination with ℓ_2-loss providing minimum mean squared error. Once the rDA is trained, its prediction on a new test instance/example is given by

$$\hat{\mathbf{y}} = \mathbf{h}\mathbf{U} \quad ; \quad \mathbf{h} = [[\mathbf{z}_{b,t}\mathbf{h}^L_{b,t}]]_{1,1}^{B,T} \ ; \ \mathbf{h}^l_{b,t} = \sigma(\mathbf{W}^l_{b,t}\mathbf{h}^{l-1}_{b,t} + p^l_{b,t}) \ ; \ \mathbf{h}^0_{b,t} = \mathbf{x}, \tag{9}$$

2.2 rDA marker (rDAm)

The rDA model discussed about will now be adapted to the problem of designing a MVUB of disease spectrum (refer to the discussion at the end of Section 2). Recall that the sigmoid non-linearity ensures that the outputs from the network lie in $[0, 1]$, and so the rDA predictions on new test examples are bounded between 0 and 1. Further, observe that we are interested in designing a bounded predictor so that the entire spectrum of dementia is 'covered'. To obtain the rDA marker (rDAm), we train the model *only* using healthy controls (CN) with labels $y = 1$, and completely demented (AD) with labels $y = 0$. No MCI subjects are used for training. Since only CNs and ADs are used in training the model, rDAms predictions on any MCI subjects are in $[0, 1]$, giving the confidence of the learned model whether the subject resembles a CN (rDAm closer to 1) or AD (rDAm closer to 1). If more than one imaging modality is used, then the blocks can be constructed for each modality separately, and pooling can be done across all the resulting weak learners. By the properties of rDA described above, rDAm is already a minimum variance unbiased estimator of the disease spectrum. Therefore, we obtain a multi-modal MVUB estimator of disease, which can used as an inclusion criterion for clinical trials design. The full implementation of the framework will be made available at http://pages.cs.wisc.edu/∼vamsi/rda concurrently with publication.

2.3 Hyperparameters

The hyper parameters of the rDA model include:

- B: The number of blocks.
- T: The number of SDAs trained within each block. Let the SDA learning parameter (denoising rate, learning rate) space be Θ. T number of samples are generated from this space, and one SDA is trained corresponding to each of the T samples.
- $\tau(\cdot)$: The sampling distribution over all the d voxels/dimensions i.e., while constructing the B blocks, a voxel v is sampled according to the probability $\tau(v)$ and assigned to a specific block b.
- d_b: This number of voxels within each block (or length of the input layer of SDAs that would be trained within each block).
- λ: The regularization parameter for ridge regression.

Observe that within each block the randomization is over the T set of learning parameters, hence if the hyper-parameter space is sampled uniformly, rDA's outputs will be robust to changes in T. The simplest choice for the block-wise sampler $\tau(\cdot)$ assigns uniform probability over all dimensions/voxels as described above. However, we can assign large weights on local neighborhoods which are more sensitive to dementia progression, if desired. We can also setup $\tau(\cdot)$ based on entropy or the result of a hypothesis test. More precisely, entropic measures (like KL divergence) or t-scores, z-scores can be sued to summarize to class discrimination power of each voxel. The resulting scores can then be normalized and used as the sampling distribution $\tau(\cdot)$. The $\mathit{Reweigh}(\cdot)$ step in Alg. 2.1 takes care of previously unsampled dimensions/voxels. For the experiments reported in the paper; $B = 5000$, $d_b = d/B$ such that each voxel appears in only one block, $\tau(\cdot)$ is based on KL divergence estimates of differentiating ADs and CNs and $\lambda = 1$. Multiple combinations of B, d_b and $\tau(\cdot)$ (including uniform and t-score based) were evaluated, however none of these gave any significant improvements in terms of both the correlations and sample estimates.

3 MKL marker (MKLm)

As presented in Section 2 of the main paper, we compared the performance improvement of rDAm against many existing disease markers including a support vector machine (SVM) based multi-modal multi kernel learning (MKL) marker. Support Vector Machines (SVM) classify subjects into separate categories (e.g., diseased and healthy) by finding a separating hyperplane which balances classification accuracy with separating margin between the classes. In the AD setting, each subject's brain image is a feature vector with each dimension corresponding to a single voxel intensity value, or other imaging-derived feature. We seek a separating hyperplane which not only places the controls on one side, and the AD subjects on the other (classification accuracy), but also puts the greatest possible distance between the two groups of points (separating margin). We can express the SVM training procedure as a quadratic program (QP) of the form as follows,

$$\min_{\mathbf{w},b,\eta\geq 0} \frac{\|\mathbf{w}\|^2}{2} + C\sum_i \eta_i \quad s.t. \quad y_i(\mathbf{w}^T x_i + b) + \eta_i \geq 1, \quad \forall i \tag{10}$$

where $(\mathbf{w}, b)$ is a weight vector which determines the separating hyperplane, and η is a vector of "slack variables" which allow the algorithm to make errors in case the data are not completely separable. Each constraint encodes the desired outcome that one training subject be placed at least a "unit" distance away from the hyperplane (with η_i taking up the slack). The actual "units" by which the margin is measured are given by $1/\|\mathbf{w}\|$, which we maximize by minimizing $\|\mathbf{w}\|^2/2$. C expresses a trade-off between accuracy and margin. The dual problem is,

$$\min_{0\leq\alpha\leq C} \mathbf{1}^T\alpha - (\alpha\circ\mathbf{y})^T\mathbf{K}(\alpha\circ\mathbf{y}) \quad s.t. \quad \mathbf{y}^T\alpha = 0 \tag{11}$$

where $\circ$ denotes the element–wise multiplication. Notice that the examples x_i only appear as inner products $x_i^T x_i$, which we substitute with as kernel matrix $\mathbf{K}$. Various non-linear functions of the kernel matrix ($\mathbf{K} \geq 0$) correspond to non-linear transformations of the data, allowing for a richer set of classifiers. In the context of multi-modality kernel learning, each separate modality generates a series of linear and non-linear kernels. A common methodology for combining these kernels is Multi-Kernel Learning (MKL) [6]. MKL solves this problem by simultaneously choosing a linear combination of kernels, and estimating a max-margin classifier. To preserve the margin, the coefficients β must also be regularized, which gives the MKL primal problem

$$\min_{\mathbf{w},b,\eta\geq 0,\beta} \sum_m \frac{\|\mathbf{w}_m\|^2}{\beta_m} + C\sum_i \eta_i + \|\beta\|_p^2 \quad s.t. \quad y_i(\sum_m \mathbf{w}_m^T\phi_m(x_i) + b) + \eta_i \geq 1, \quad \forall i \tag{12}$$

where ϕ_m transforms the observed data to the m^{th} kernel space. The constraints now express the classifier as a sum of contributions from each kernel space. Thus, we can construct a series of kernels based on all imaging modalities, distinct image processing pipe-lines, feature selection methods, and kernel functions, which are then used to train an MKL classifier. By inspection, it is easy to see that SVM based methods using imaging data can be obtained as a special case of the formulation in (12).

For the experiments in the current work, we did not regularize over the kernel combination weights β_m, and instead a uniform combination is imposed (i.e. $\beta_m = 1/M$, where M denotes the number of kernels used). Existing work [5] shows that this is a robust and competitive baseline in neuroimaging experiments, and avoids the use of computationally demanding optimization schemes. A total of 11 different kernels are used: one linear kernel, 5 polynomial and 5 Gaussian kernels. Similar to the setup used in constructing rDAm, this uniform weighted MKL model is trained on ADs and CNs *alone*, and tested on MCIs, resulting the MKL marker (MKLm). The regularization parameter (C) is chosen based on 10 fold cross validation over the training accuracy (AD vs. CN).

References

[1] Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2:1–127, 2009.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[4] Thomas G Dietterich. Ensemble methods in Machine learning. *Multiple Classificer Systems*, pages 1–15, 2000.

[5] Chris Hinrichs, Vikas Singh, Jiming Peng, and Sterling Johnson. Q-mkl: Matrix-induced regularization in multi-kernel learning with applications to neuroimaging. In *Advances in Neural Information Processing Systems*, pages 1421–1429, 2012.

[6] Chris Hinrichs, Vikas Singh, Guofan Xu, and Sterling C Johnson. Predictive markers for AD in a multi-modality framework: an analysis of MCI progression in the ADNI population. *Neuroimage*, 55:574–589, 2011.

[7] Stephen Kay. *Fundamentals of Statistical Signal Processing: Estimation Theory.* Prentice Hall, 1993.

[8] Bernard Rosner. *Fundamentals of Bioinformatics.* PWS-Kent Publishing Company, Third Edn, 1990.

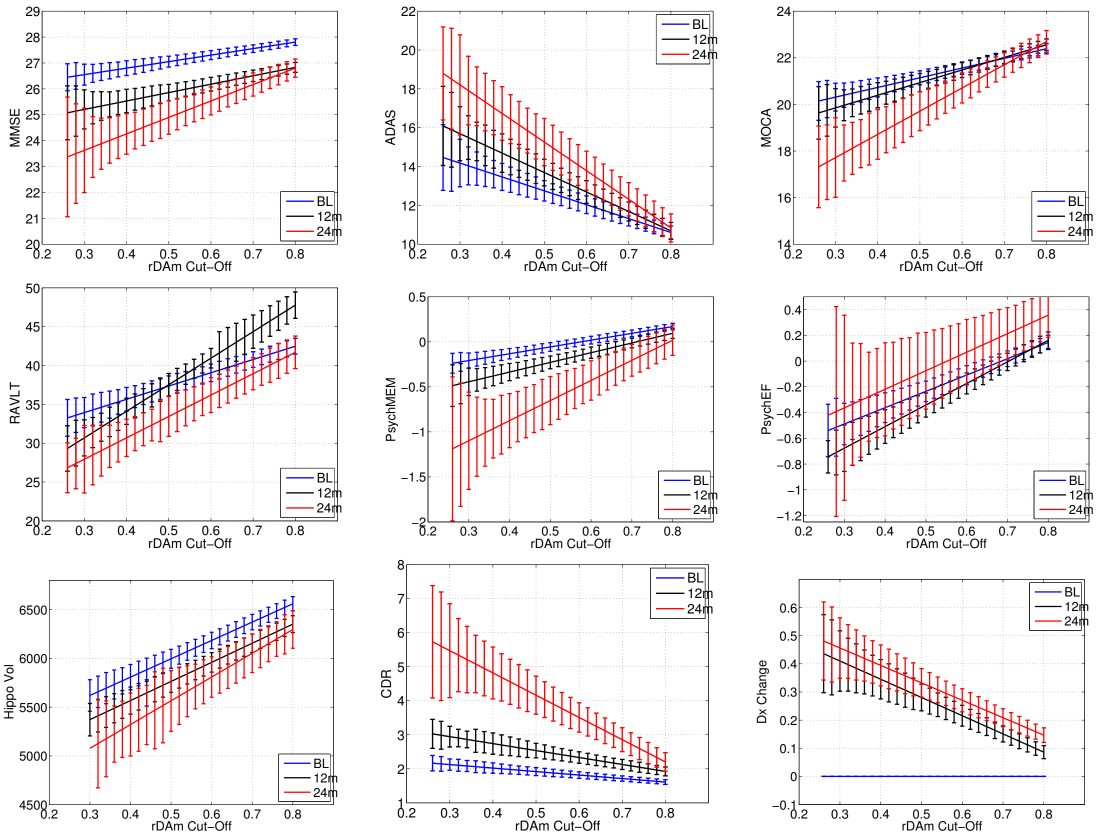


Figure 6: Mean of several disease markers as a function of baseline rDAm enrichment threshold. Each plot corresponds to one disease marker (which include MMSE, ADAS, RAVLT, MOCA, PsychMEM, PsychEF, Hippocampal Volume, CDR-SB and DxConv. x-axis represents the baseline rDAm enrichment cut-off (t). For each t, the subjects who have baseline rDAm $\gg t$ are filtered-out, and the mean of within subject change in the diseas marker is computed on the remaining *un*-filtered subjects. Lines are the corresponding linear fit, and the error bars correspond to the stnadard errors. Blue, black and red represent baseline, 12 and 24 months respectively.