



MR Image texture analysis applied to the diagnosis of Alzheimer's Disease

Mathias Bjørn Jørgensen & Mirza Hasanbasic

13th June 2016

Contents

1	Introduction	3
1.1	Problem Definition	3
2	Data	4
2.1	ADNI data	4
2.2	Preprocessing	4
3	Method	6
3.1	Image texture analysis methods	6
3.1.1	Co-occurrence matrix	6
3.1.2	Texture features from co-occurrence matrix	9
3.2	Machine learning	9
3.2.1	10-fold cross-validation	9
3.2.2	Feature selection	10
3.2.2.1	Naive	12
3.2.3	K Nearest Neighbors algorithm	12
3.3	Erode	13
4	Implementation	15
4.1	Preparation of Data	15
4.2	Data Calculations	16
4.2.1	Calculating GLCMs	17
4.3	Naive feature	18
4.3.1	Calculating(Computing) the GLCM Features	18
4.4	Plotting the GLCM features	20
4.5	Forward feature selection	21
5	Result	23
5.1	Plots of 2D data	23
5.1.1	Left Hippocampus, normalized and eroded	23
5.2	Plots of 3D data	30
5.2.1	Left Hippocampus, not normalized and eroded	30
6	Discussion	31
6.1	Comparison of image textures	31
6.2	GOD SECTION TITEL	34
6.2.1	Naive Selection	35

6.3 Overfitting	38
7 Conclusion	40
Appendices	41
A Co occurrence matrix derivation features	42
B Feature Offset helper	44
B.1 For the 3D	44
Litteratur	45

List of Figures

3.1	Example of offsets that are transpose	7
3.2	Example of the offsets for the 2D	7
3.3	Image I that is 4-by-4	8
3.4	Four different COM's for a gray-tone image. Shows how the GLCM are calculated of the 4-by-4 image I 3.3.	8
3.5	Example of the offsets for the 3D	8
3.6	The KNN grows a spherical region until it encloses k training samples, and labels the test point by a majority vote of these samples. In this $k=5$ case, the test point \mathbf{x} would be labelled the category of the black points	12
3.7	Hippocampus at slice 10 on the X-axis as bitwise	13
3.8	Text	13
3.9	Example of a city-block erosion with before and after	14
3.10	2D example of the 3D city-block	14
4.1	Plot of the Angular Second Moment features where the data have been normalized and eroded. This is the left Hippocampus. The red are the patients with AD and the blue are the rest	21
5.1	Plot of the Angular Second Moment features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest	23
5.2	Plot of the Contrast features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest	24
5.3	Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest	24
5.4	Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest	25
5.5	Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest	25
5.6	Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest	26
5.7	Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest	26
5.8	Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest	27
5.9	Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest	27

5.10 Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest	28
5.11 Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest	28
5.12 Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest	29
5.13 Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest	29
6.1 Plot of the accuracy for the 2D of our model with a feature accuracy of 91% and model accuracy with k values from 1 to 10 for KNN	37
6.2 Plot of the accuracy for the 3D of our model with a feature accuracy of 95% and model accuracy with k values from 1 to 10 for KNN	38

List of Tables

3.1	Testing redundancy	11
4.1	C_{x-y} for k value 254	18
4.2	C_{x-y} for k value 246	18
4.3	C_{x+y} for k value 511	19
4.4	C_{x+y} for k value 500	19
6.1	2D: Accuracy of final KNN-model L = left, E = erosion, N = normalized	31
6.2	3D: Accuracy of final KNN-model, two values for 3DL since feature selection resulted in a tie between k = 1 and k = 3	32
6.3	Feature selected for 3D no erosion, without normalization	33
6.4	Feature selected for 3D no erosion, with normalization	33
6.5	Feature selected for 3D with erosion, without normalization	33
6.6	Features selected for 3D erosion and normalization	34
6.7	Accuracy for number of features with a unknown k value, we looked after the best accuracy. So this table tells us that no matter what, we would get the best accuracy with only 4 features selected	35
6.8	Where F stands for Feature #. For feature 3 and feature 1, where it seem that we get the best accuracy with feature 4 and 5. The reason that rows for feature 3 and 1 are zeros is because those features are already selected.	36
6.9	Six features shown for $k = 3$	36
6.10	Eleven features shown for $k = 1$	37
B.1	Add caption	44

List of Corrections

Fatal: skriv Matthew har skrevet data beskrivelsen	4
Note: Noget her	12
Note: features for KNN then we run 10 fold crossvall and resample the data 100 times	12
Note: Tal her	18
Note: Snak om Datacalculation filen	18
Note: Snakke om hvorfor normalized data. Features vi har valgt fra freebourug har gjort det og måske gjort i det mente om at de kan ende med pæne værdier – i method muligvis	20
Note: MERE?	22
Fatal: Create a table to show what each data set contains?	31
Note: Skriv at vi kun kigger på 3d/2d normalized erode	34
Note: argument her	35
Note: Mere her	35
Note: Skriv de offsets her	35

Mathias Bjørn Jørgensen & Mirza Hasanbasic

Abstract

This report will examine MRI scans of brains, using image texture analysis and machine learning

Chapter 1

Introduction

Alzheimer's Disease (AD) is the most common cause of dementia among people and is a growing problem in the aging populations. It has a big impact on health services and society as life expectancy increases. In 2010 the total global costs of dementia was estimated to be about 1% of the worldwide gross domestic product¹. AD is the cause in about 60%-70% of all cases of dementia[1] and about 70% of the risk is believed to be genetic [7]. Currently there are no way to cure dementia or to alter the progressive course. But however, much can be done to support and improve the lives if AD is diagnosed in the early stage of progression [1]

In this report we will examine MRI data of the hippocampus using image texture analysis and apply machine learning in order to diagnose AD in patients. Our dataset contain 100 patients 50 control and 50 with AD.

We will be using two different image texture analysis method, one which will me in 2D[9][6] and the other one will be in 3D[11], from which we calculate the data to the gray level co-occurrence matrix (GLCM).

1.1 Problem Definition

Is it possible to classify MRI data of the hippocampus into groups of healthy controls vs Alzheimer's patient, using a predefined set of image texture metrices, with an accuracy greater than 80%?

Is there a difference in diagnosing AD successfully by calculating the co-occurence matrix in 3D compared to 2D.

¹With the terms as of direct medical costs, direct social costs and costs of informal care

Chapter 2

Data

Fixme Fatal:
skriv
Matthew har
skrevet data
beskrivelsen

Fixme Fatal: skriv Matthew har skrevet data beskrivelsen

2.1 ADNI data

The data in this study was provided already downloaded and preprocessed. It had been previously obtained from the ADNI database (adni.loni.usc.edu). The ADNI was launched in 2003 by the National Institute on Aging, the National Institute of Biomedical Imaging and Bioengineering, the Food and Drug Administration, private pharmaceutical companies, and nonprofit organizations, as a \$60 million, 5-year, public-private partnership. The primary goal of ADNI has been to test whether serial MRI, positron emission tomography (PET), biological markers, and clinical and neuro-psychological assessments can be combined to measure the progression of MCI and early AD. Determination of sensitive and specific markers of very early AD progression is intended to aid researchers and clinicians to develop new treatments and monitor their effectiveness, as well as to lessen the time and cost of clinical trials. ADNI is the result of efforts of many co-investigators from a broad range of academic institutions and private corporations, and subjects have been recruited from over 50 sites across the U.S. and Canada. For up-to-date information, see <http://www.adni-info.org/> We were provided with a random subset (50 controls, 50 AD) of the “complete annual year 2 visits” 1.5T dataset from the collection of standardized datasets released by ADNI <http://www.adni.loni.usc.edu/methods/mri-analysis/adni-standardized-data/> [13]. The complete dataset (504 subjects) comprised one associated 1.5T T1-weighted MRI image out of the two possible from the back-to-back scanning protocol in ADNI [10] at baseline, 12-month follow-up, and 24-month follow-up.

2.2 Preprocessing

The data were provided for use already preprocessed. This preprocessing, and subsequent hippocampal segmentation was performed with the freely available FreeSurfer software package (version 5.1.0) [8] using the cross-sectional pipeline with default parameters. The original

MRI image resolution of [0.94, 1.35] x [0.94, 1.35] x 1.2mm was conformed to a 1.0 x 1.0 x 1.0 mm resolution, and all MRIs were bias field corrected. The bias field correction in Free-Surfer utilizes the nonparametric nonuniform intensity normalization algorithm [12], often referred to as N3. The input data for this study was therefore the corrected T1-weighted MRI image volume, and corresponding separate binary masks of left and right hippocampi.

Chapter 3

Method

3.1 Image texture analysis methods

Image texture is a feature that can help to segment images into regions of interest and to classify those regions. Textures gives us some information about the arrangement of the intensities in an image. Texture analysis is a technique for evaluating the position and intensity of signal features[6]. Statistical texture analysis methods evaluate the interrelationship of voxels, based on mathematical parameters computed from the distribution and intensities of voxels in the image.

3.1.1 Co-occurrence matrix

The co-occurrence matrix (COM) is second-order statistics methods, which is based on information about colours in pair of pixels. The matrix is defined over the image with distribution values at a given offset. Mathematically we have a COM matrix \mathbf{C} which is defined over an $n \times m$ image \mathbf{I} , with $\Delta x, \Delta y$ being the parameterized offset, is calculated by [4]

$$C_{\Delta x, \Delta y}(i, j) = \sum_{p=1}^n \sum_{q=1}^m \begin{cases} 1, & \text{if } I(p, q) = i \text{ and } I(p + \Delta x, q + \Delta y) = j \\ 0, & \text{otherwise} \end{cases}$$

The element (4,3) in the COM, figure 3.3, can be translated to meaning how many times there exist an element in the image \mathbf{I} with GI 4 and another element offset $\Delta x, \Delta y$ from the original with greyscale intensity¹ GI 4 i.e. if the offset is (0,1) and the first element is COM_(4,3) is two, it translates into there being two instances with element (x,y) = 4 and (x+Δx,y+Δy) = 3. COMs calculated on GIs are often called gray-level co-occurrence matrices (GLCM).

A single image have multiple GLCMs as different offsets creates different relations. Consider a 3×3 matrix looking at element (2,2) we can then create eight different offsets, GLCM_(1,0), GLCM_(1,1), GLCM_(0,1), GLCM_(-1,1), GLCM_(-1,0), GLCM_(-1,-1), GLCM_(0,-1), and GLCM_(1,-1) however they are not unique.

¹The pixel value is a single number that represents the brightness of the pixel. Typically one is taken to be black and 256 is taken to be white and values in between make up the different shades of grey

Focusing on the two offsets $(0,1)$, $(0,-1)$ in element $(2,2)$ and $(1,2)$ with GI 1 and 2 respectfully increases the entry $GLCMs_{(1,0)}(1,2)$ and $GLCMs_{(-1,0)}(2,1)$ with one, showing that

$$GLCMs_{(1,0)} = GLCMs_{(-1,0)}^T$$

With an example in figure 3.1a that shows $GLCMs_{(1,0)}$ and figure 3.1b for $GLCMs_{(-1,0)}$ where the image I can be seen in figure 3.3

		j			
		1	2	3	4
i	1	2	0	1	1
	2	0	2	0	0
	3	0	0	1	0
	4	0	2	0	2

		j			
		1	2	3	4
i	1	2	0	0	0
	2	0	2	0	2
	3	1	0	1	0
	4	1	0	0	2

(a) Example of the offsets for the 2D with $GLCMs_{(1,0)}$ (b) Example of the offsets for the 2D with $GLCMs_{(-1,0)}$

Figure 3.1: Example of offsets that are transpose.

There exist the same relation between

$$\begin{aligned} GLCMs_{(1,1)} &= GLCMs_{(-1,-1)}^T \\ GLCM_{(0,1)} &= GLCM_{(0,-1)}^T \\ GLCMs_{(-1,1)} &= GLCMs_{(1,-1)}^T \end{aligned}$$

This leaves four different offsets for analysis $(0,1), (-1,1), (-1,0), (-1,-1)$ in general $(0,d), (-d,d), (-d,0), (-d,-d)$ where d is the distance which are commonly named angles $0^\circ, 45^\circ, 90^\circ$ and 135° as seen in figure 3.2.

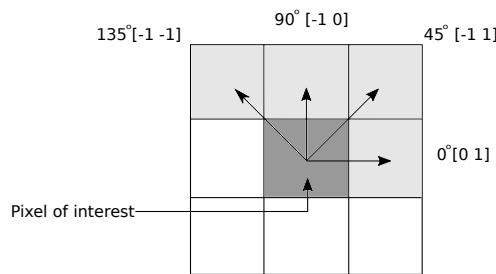


Figure 3.2: Example of the offsets for the 2D

and let figure 3.3 illustrate this concept with a 4×4 image I with four different COM's for $I : C_{(0,1)}, C_{(-1,1)}, C_{(-1,0)}$ and $C_{(-1,-1)}$

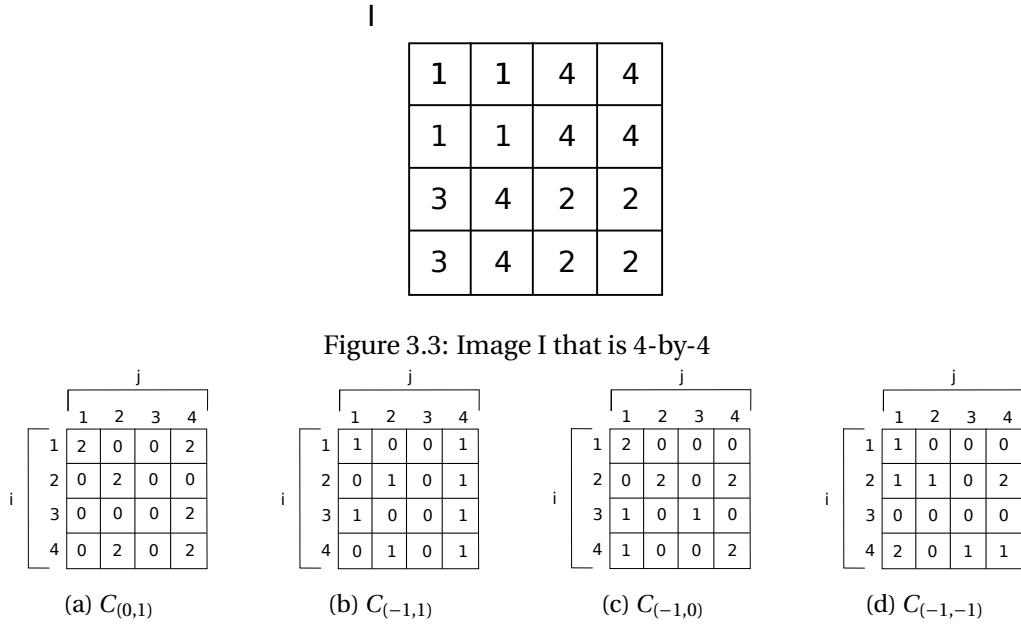


Figure 3.4: Four different COM's for a gray-tone image. Shows how the GLCM are calculated of the 4-by-4 image I 3.3.

The co-occurrence matrix is quadratic with the number of rows and columns equal to the amount of GI, for example if we have 256 GI we get a 256×256 GLCM.

Extending this method to three-dimensions it is necessary to look on how the offsets are defined because the size of the GLCM is defined by the amount of GIs and not by the images it is derived from. Considering a $3 \times 3 \times 3$ matrix we have a possible of 26 offsets. In two-dimensions it is possible to eliminate half of the offsets because of the relation $\text{GLCM}_{d,d}^T = \text{GLCM}_{-d,-d}$, and it is the same case in three-dimensions with the relation being $\text{GLCM}_{d,d,d}^T = \text{GLCM}_{-d,-d,-d}$. This leaves 13 offsets which are illustrated below.

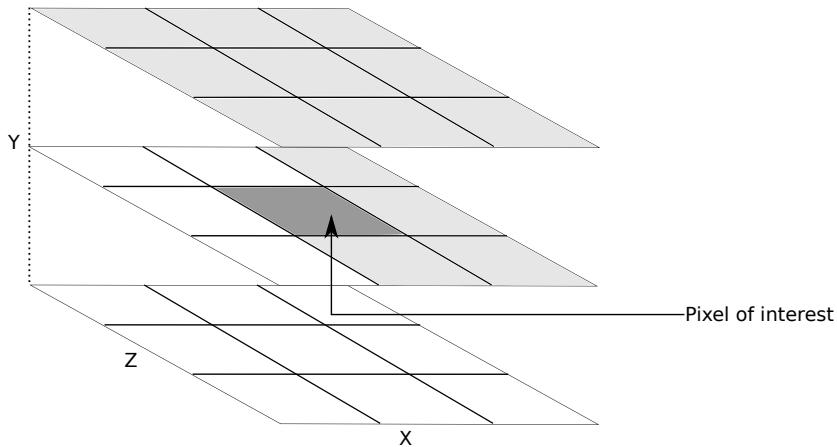


Figure 3.5: Example of the offsets for the 3D

To create a two-dimensional GLCM on three-dimensional image we create slices through the image. Given a $n \times m \times l$, $n = 20$, $m = 20$ $l = 20$, image we can create a slice for each n. Slice₁ is

equal to the matrix M ($n = 1 \times m \times l$), $\text{Slice}_2 = M$ ($n = 2 \times m \times l$), and so on, giving a total of 20 $m \times l$ images instead. It is possible for each slice to calculate the $\text{GLCM}_{d,d}$ for each slice, and we define $\text{GLCM}_{d,d} = \sum_{n=1}^2 \text{GLCM}_{d,d}(\text{Slice}_n)$

These slices are done in all three directions of the image for the four difference offsets resulting in 12 GLCMs. There is an overlap between six of the GLCMs. Slicing through n axis with offset $(0,d)$ is equal to slicing through the m axis with the same offset. Slicing through the m axis with offset $(-d,d)$ is equal to slicing through the l axis with offset $(-d,0)$. Slicing through the l axis with offset $(-d,d)$ is equal to slicing through the n axis with offset $(d,0)$, which is the transposed offset of $(-d,0)$. Removing the duplicates leaves nine GLCMs which we calculate at distances $d = (1,2,\dots,10)$, giving a total of 90 GLCMs for each mri scan.

The three-dimensional versions is also calculated for distances $d = (1,2,\dots,10)$ resulting in 130 GLCMs for each mri scan.

3.1.2 Texture features from co-occurrence matrix

To compare the differences between the GLCMs 13 different features are computed. They are the same as used in [9] except for one difference. The difference is how the correlation is calculated as $\sum_{i,j} \frac{(i-\mu_i)(j-\mu_j)g\text{lcm}(i,j)}{(\sigma_i \cdot \sigma_j)}$, where $\mu_i, \mu_j, \sigma_i, \sigma_j$ are the means and standard deviations of C_i and C_j respectively. This correlation is called the Pearson product moment correlation coefficient and it determines how correlated a pixel is to its neighbour, over the whole image[2][3]. We calculate two versions of these features, one where we normalized the COMs and one where we do not.

3.2 Machine learning

Machine learning is a method used to create complex models and algorithms that lend themselves to prediction, when the models are exposed to new data that should be able to teach themselves to grow and change. There are several different categories of machine learning, one of them being supervised learning. In supervised learning given a large sample of input and output pairings, the goal is to find the complex function that maps that relation between input and output. With a sufficient large data set it would be possible to train a supervised algorithm to predict output values for some new input values that it have not seen before.

3.2.1 10-fold cross-validation

Considering we have a small sample size with 100 observations, it would not be ideal to leave data to only test on. Not having a test set to validate the training, results in a model that only tells how well it predicts the accuracy of the training set. Given a model with unknown parameters and a training set to which the model can be fit then the fitting process optimizes the models parameters to fit the training data. Validating this model against independent data (test data) from the same data pool as the training, it will generally turn out that the model does not fit the test data as well as the training data. It is known as overfitting and is a

problem when the size of the training data set is small. Cross-validation is used to counteract overfitting while still using all of the data to train, which is very useful for small data sets.

Randomly dividing the entire data set into 10 groups, one sub sample is saved for testing and the remaining nine is used to fit the model. The procedures is done so all sub samples get to be used to validate exactly one time and the validation result can then be averaged to produce a single estimation. This process is repeated an appropriate number of times and averaged to minimize the influence of the variance. How many times it is repeated depends on the size of the folds as we do not want duplicated folds. We will be using 100 repetitions. The structure of the folds are also important considering we have equal amounts of Control and AD data, it "If data are stratified, the proportions of the different strata should (approximately) be the same in the sample and in each training and validation sample." [5] Meaning every fold should have five control and five AD. This reduce the problem of overfitting, as the validation data set is never used in to fit the model.

3.2.2 Feature selection

Feature selection is the process of selection af subset of relevant features for use in model construction. The main goal of feature selection is to choose a subset of the entire set of input features so the subset can predict the output with an accuracy comparable to using the entire set to predict the output, but with a large reduction in the computational cost. When a data set contains many features that are either redundant or irrelevant it is possible to remove them without incurring much loss of information. Features may be redundant due to the presence of another feature with which it is strongly correlated, while they may be very informative for the model their information have already been provided by a different feature. Only choosing a subset of the possible features have the added advantage of decreasing the complexity of the model.

Sequential forward feature selection(SFS) is a greedy algorithm to choose features. It starts off with finding the best possible single feature to describe the model. Given the feature set of that single feature, the next step is to find which other feature would improve the predictiveness of the model and then add that to the set of selected features. It continues to grow the set of selected features, until the goal have been reached. The goal can either be a specific amount of features, a specific accuracy for the model or it can stop if all choices of a new feature would decrease the accuracy. A problem with SFS is due to its greedy nature, there is no guarantee that the first feature selected is part of the optimal solution. Given three features X_1 , X_2 and X_3 where X_1 is the best single feature, X_2 is second best and X_3 the worst, it does not necessitate that pairing X_1 , X_2 is better than X_2 , X_3 , nor does it secure any other relation, meaning that the SFS does not guarantee the optimal solution. Which leads to one of the drawbacks of this feature selection, if a feature is selected it is not possible to exclude it later even if it would increase the evaluation score to do so.

Prior to running the SFS algorithm we eliminate incomplete features, as in if not all 100 patients have a value for set feature we chose to ignore it. The reason for this is we have over 1000 features for each patient, so each feature have at minimum nine very similar features, as in they are calculated on the same offset, but at different distances. This will make a lot of features redundant, and to optimize the run time we eliminate all incomplete features. This

similarity between features is also used in the SFS algorithm. When a feature is selected we remove it from the set of features, but we also remove the features which only difference is they are calculated on a different distance, i.e. if the feature selected is correlation with offset (0,d) with $d = 1$, we remove all correlation features with offset(0,d) from the feature set. However the relation does not hold entirely, as a few combinations do return good values. In table 3.1 we have for offsets $\{(d,0,0), (d,0,d), (0,0,d)\}$ computed a 10-fold cross validation on the Imoc2 feature with 100 different samples, for all distance to see how they affect the accuracy. In general the accuracy does not improve significantly by adding more of the same features, however that does not hold for the first offset. We do still use it in order to decrease the computational cost.

0.6699	0.6699	0.6056	0.6056	0.6195	0.6195
0.6953	0.6946	0.6625	0.6625	0.6086	0.6059
0.6441	0.673	0.6956	0.6367	0.5961	0.6457
0.669	0.6593	0.7115	0.6186	0.6948	0.6622
0.5341	0.6927	0.6176	0.6374	0.6857	0.6458
0.5616	0.6999	0.5529	0.6362	0.7384	0.6794
0.6587	0.7031	0.4986	0.6182	0.6782	0.6528
0.4668	0.7773	0.5333	0.5762	0.5738	0.6738
0.489	0.7451	0.4307	0.5929	0.6574	0.6697
0.4817	0.7616	0.4552	0.632	0.5149	0.6779

Table 3.1: Testing redundancy

```

1 The algorithm for SFS with 10-fold cross validation
2 Step 1:
3 Set selected features Y = \emptyset
4 X = entire feature set
5 Separate data set into 10 folds of equal size with 5 of each class , {K1,K2 ,...,K10} .
6
7 Step 2:
8 For feature i= 1 to No. of features
9   for fold j=1 to No. of folds
10    Train a KNN model on {Y,Xi} using fold K1,2,...,10\ Kj as training
11    Calculate miss classification error of model on Kj
12 Average the error for all 10 folds.
13
14 Step 3:
15 F = feature with lowest error
16 X = X \ F
17 Y = Y ∪ F
18
19 Step 4:
20 Continue step 2-3 until the miss classification error worsen or 15 features have been
   ↵ selected.

```

The algorithm is run on 10 different k values for the KNN model, $k = 1,2,\dots,10$. SFS does not terminate until the accuracy worsen in an iteration or 15 features have been identified, however when we create the model we only select features up until the accuracy does not improve. The unselected data does not improve the accuracy, so is considered to be redundant.

3.2.2.1 Naive

Fixme Note: Noget her

Fixme Note:
Noget her

3.2.3 K Nearest Neighbors algorithm

The K Nearest Neighbor (KNN) is a method that is used for classification and regression. It should be noted, that KNN is a non parametric lazy learning algorithm, which means that it does not make any assumptions on the underlying data distribution. In other words, it means that the training phase is fast and KNN keeps all the training data. It should be noted that KNN makes decision based on the entire training data set and in the KNN an object is classified by majority vote of its neighbors, with the object being assigned to the class most common amongst its *KNN*'s.^{3.6}

Since the training phase is minimal, then it should be noted that the testing phase is very costly for KNN in both memory and time and often it can be a worst case for time needed, since all points might take part in the decision making.

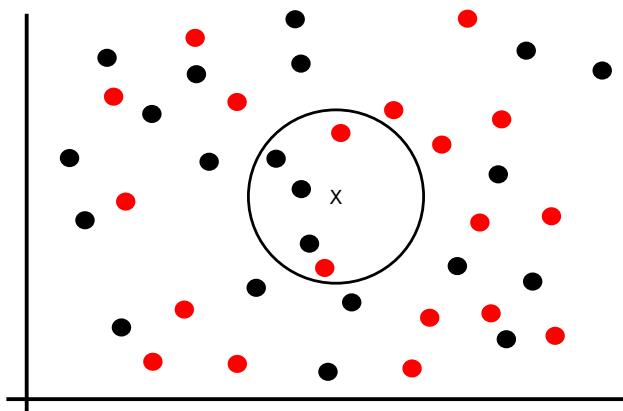


Figure 3.6: The KNN grows a spherical region until it encloses k training samples, and labels the test point by a majority vote of these samples. In this $k = 5$ case, the test point x would be labelled the category of the black points

Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Imagine that we have a large scale variables, they will have a much larger effect on the distance between observations and hence the KNN classifier, than variables on a small scale. Often a good way to handle this problem is to standardize the data.

But not only this will affect KNN. For KNN to be more precise it will need some features it can use to classify the data. We have chosen to only work with a maximum of 15 features. This means for a $k = 1$ KNN will have to look at nearest neighbour of 1 in a 15 dimensional space and another reason to our limit of 15 features is to keep the calculations down.

Fixme Note: features for KNN then we run 10 fold crossval and resample the data 100 times

Fixme Note:
features for
KNN then we
run 10 fold
crossval and
resample the
data 100
times

The disadvantage of KNN is that choosing a k may be tricky, so we are left to test the algorithm on multiple k 's and often it needs a large number of samples for better accuracy.

Typically the KNN classifier will be based on a distance, commonly it is based on the Euclidean distance between a test sample and the specified training samples. Let x_i be an input sample with p features $(x_{i1}, x_{i2}, \dots, x_{ip})$, and n be the total number of input samples $i = 1, 2, \dots, n$ and p the total number of features $j = 1, 2, \dots, p$. The Euclidean distance between sample x_i and x_l , $l = 1, 2, \dots, n$ is defined as $d(x_i, x_l) = \sqrt{(x_{i1} - x_{l1})^2 + (x_{i2} - x_{l2})^2 + \dots + (x_{ip} - x_{lp})^2}$

3.3 Erode

Each patients hippocampus has been segmented in the MRI scan. The problems we can run into are that the background will blur with the segmentation i.e. the hippocampus. Performing a erosion can solve this problem and we can focus on the hippocampus, with the maximum number of details. As seen in figure 3.7a the erosion has not been performed yet, but we might have some problems with data surrounding the hippocampus is blurring out the edges of the hippocampus. To solve this, we create a mask to separate the hippocampus from the background data and end up with what is left in figure 3.7b

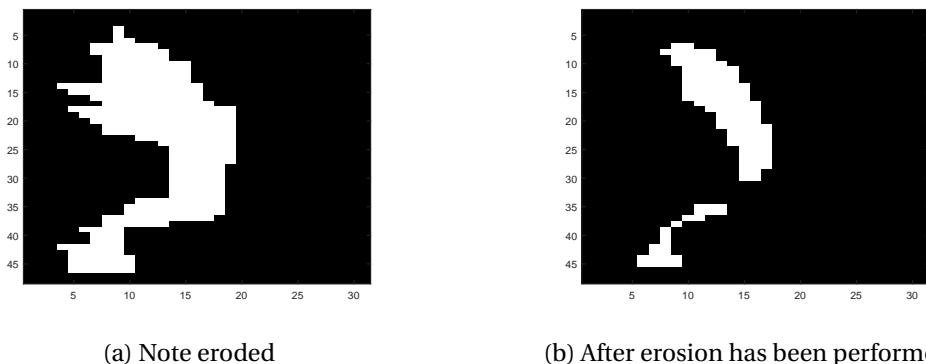


Figure 3.7: Hippocampus at slice 10 on the X-axis as bitwise

The erosion we have used is called a city-block metric and can be seen in figure 3.8.

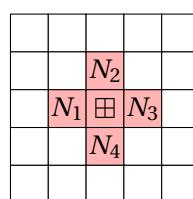


Figure 3.8: Text

To give an example of how the erosion works, it will be illustrated and we will use the city-block metric for this purpose. So we will use figure 3.8 and erode the image in figure 3.9.

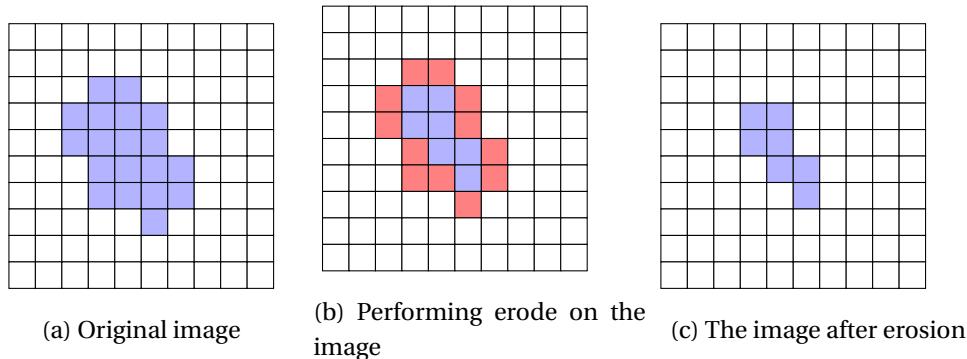


Figure 3.9: Example of a city-block erosion with before and after

As seen in figure 3.9 the noise (background) have been removed. This is an example in 2D. Now we wish to extend the erosion city-block to 3D. As seen in figure 3.8 it have 4 neighbours and when we extend this to 3D we will end up with 6 neighbours instead as seen in figure 3.10 and the concept is still the same as in 2D

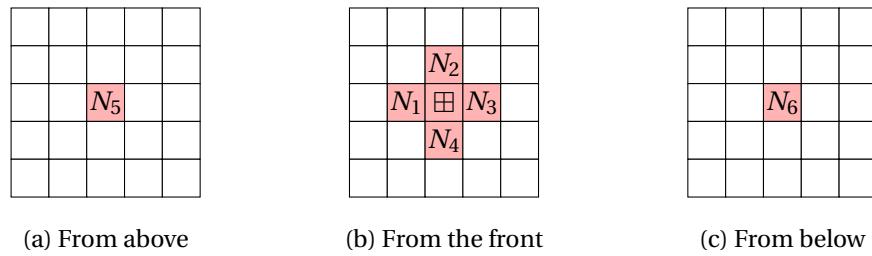


Figure 3.10: 2D example of the 3D city-block

Chapter 4

Implementation

4.1 Preparation of Data

The mri files are $256 \times 256 \times 256$ matrices but we are only interested in the small part which overlaps with the masks from segmentation, i.e. where the elements in segmentation is either 1 (left hippocampus) or 2 (right hippocampus). We have created the function `HippoMatrix`, which takes three variables, which file to load, whether or not erosion should be performed, and if the left or right hippocampus is desired. First we assign a value based on if we are looking for the right hippocampus, as they are associated with 1 and 2 respectfully. If erosion is desired we create the city-block for erosion by taking advantage of distances calculations, $\text{distance} = \sqrt{x^2 + y^2 + z^2}$. All parts of the city-block have distance 1 from the origin point. With the city-block defined we can use matlab's build in function `imerode` to perform the erosion. From matlab's website we have the function `imerode` <http://se.mathworks.com/help/images/ref/imerode.html> which shows that `imerode` takes two arguments, the image and the argument of which erosion type we use.

```
1 if (strcmp(var, 'noterode') == 1)
2     sum_of_1 = size(find(segmentation == tmp), 1);
3     left = zeros(sum_of_1, 4);
4
5 elseif (strcmp(var, 'erode') == 1)
6     [xx, yy, zz] = ndgrid(-1:1);
7     nhood = sqrt(xx.^2 + yy.^2 + zz.^2) <= 1.0;
8     segmentation = imerode(segmentation, nhood);
9
10    sum_of_1 = size(find(segmentation == tmp), 1);
11
12    left = zeros(sum_of_1, 4);
13 end
```

Looping through the entire segmentation matrix we identify all the datapoints where segmentation is one for the left hippocampus or two if we are trying to identify the right hippocampus. For each instance in segmentation we save the coordinate (i,j,k) and the mri(i,j,k) value in an array as $v(1) = (i_1, j_1, k_1, \text{mri}(i_1, j_1, k_1))$.

```
1 counter1 = 0;
```

```

1 for i = 1:256
2     for j = 1:256
3         for k = 1:256
4             if segmentation(i, j, k) == tmp;
5                 counter1 = counter1 + 1;
6                 left(counter1,1) = i;
7                 left(counter1,2) = j;
8                 left(counter1,3) = k;
9                 left(counter1,4) = mri(i,j,k);
10            end
11        end
12    end
13 end
14

```

On the basis of this we can create a three-dimensional matrix which contains all the data-points, $\text{hippoBox} = \max(i) - \min(i) + 1 \times \max(j) - \min(j) + 1 \times \max(k) - \min(k) + 1$. Then we simply loop through our array with the relevant data and input them into `hippoBox`, all other elements inside the matrix are set to NaN.

```

1 for i = 1:sum_of_1
2     hippoBox(left(i,1)+1-minI, left(i,2)+1-minJ, left(i,3)+1-minK) = left(i,4);
3 end

```

The return value from the function is the matrix `hippoBox` containing only the relevant data.

4.2 Data Calculations

In our function file, we do a lot of stuff that will be described in details. But in this file, we load our labels file, and take care of calculating every patient file to find a GLCM and from this GLCM we find the GLCM features.

First we check whether we have a patient with AD or not and name them respectively to their group.

Now we calculate the GLCM for the 2D and 3D which we have two functions doing the work. These functions, `glcm2dFast` and `GLCM3D`, take the **HippoMatrix** data, as mentioned in preparation data, and the desired distance that we wish to calculate to.

Now we initiate two cells for the GLCM Features which we derive using the function **GLCMDerivations** which will take the GLCM data and if we wish to normalize the GLCM or not as input.

```

1 data_glcm2D = glcm2dFast(HippoMatrix(files(j).name, erode, leftright), 10);
2 data_glcm3D = GLCM3D(HippoMatrix(files(j).name, erode, leftright),10);
3
4 data_Derivations2D = cell(90, 1);
5 data_Derivations3D = cell(130, 1);
6
7 for k = 1:size(data_Derivations2D, 1)
8     data_Derivations2D{k} = GLCMDerivations(data_glcm2D{k}, norm);
9 end
10 for k = 1:size(data_Derivations3D, 1)

```

```
11 data_Derivations3D{k} = GLCMDerivations(data_glc3D{k}, norm);
12 end
```

Lastly we save the data to their respectively folders.

4.2.1 Calculating GLCMs

To calculate the GLCMs in two-dimensions we have taken advantage of Matlabs built-in function graycomatrix. It calculates as described in methods. It is then a matter of giving the proper offsets, and the right number of GIs. We can then loop through the hippoBox slices and sum up the GLCMs.

```
1 tmp0 = graycomatrix(squeeze(data(:, :, :)), 'Offset', offsets, 'NumLevels', 256, '
2     ↪ GrayLimits', [0 256]);
3 anglex(:, :, 1) = anglex(:, :, 1) + tmp0(:, :, 1);
4 anglex(:, :, 2) = anglex(:, :, 2) + tmp0(:, :, 2);
5 anglex(:, :, 3) = anglex(:, :, 3) + tmp0(:, :, 3);
6 anglex(:, :, 4) = anglex(:, :, 4) + tmp0(:, :, 4);
```

And the same counts for angle y and angle z, but where `data(:, i, :)` is for angle y and `data(:, :, i)` for angle z.

We ultimately save all 90 GLCMs in a cell.¹

To implement the three-dimensional GLCMs we have created our own function. The function GLCM3D takes a hippoBox as data and how many distances desired. I then for each distance loop through the entire matrix, and for each element it checks if it is NaN value and larger than zero. The check utilizes that NaN is not larger than zero, so `data(i,j,k) > 0` returns false incase of `data(i,j,k) = NaN`. The reason we also insist that it should also be larger than zero is because a few of the right hippocampus include GI of value zero in their hippoBox, but as zero is the value the mri scans have outside the brain we choose to ignore the few instances. To include them would mean we had to increase our GLCMs by 1 in size, which would make them differ from the GLCMs derived in two-dimensions, making the comparison unfair. In addition matlab start their index for their matrices with one and not zero so we would also have to add every index with one creating greater complexity.

```
1 ...
2 if (datapoint > 0)
3     %Four first equals to GLCM2D in x axis
4     %and same order as in 2D
5     if (i + d < sizex)
6         %if different for NaN set into GLCM
7         if (isnan(data(i+d, j, k)) == 0 && data(i+d, j, k) > 0)
8 ...
```

Given that the datapoint is relevant, i.e. larger than zero, we then have to look at the thirteen offsets, to see if we need to increment an element in on of the GLCMs. For each offset check if the offset is inside the hippobox, and is the offset elemnt a non NaN nonzero value. If

¹A cell array in MATLAB is a general container that will hold any object

so we then increment the relevant GLCM, lets say it is the offset($d, 0, 0$), $d = 1$, in element $\text{GLCM}_{(d, 0, 0)}(x, y)$ ($\text{hippoBox}(i, j, k), \text{hippoBox}(i+1, j, k)$).

```
1 glcm1(datapoint, data(i+d, j, k)) = glcm1(datapoint, data(i+d, j, k)) + 1;
```

We have defined our thirteen offsets as $\{(d, 0, 0), (d, 0, d), (0, 0, d), (-d, 0, d), (d, -d, 0), (d, -d, d), (0, -d, d), (-d, -d, d), (-d, -d, 0), (-d, -d, -d), (0, -d, -d), (d, -d, -d)\}$. Because of the relationship between the offsets, as in $\text{GLCM}_{(d, 0, 0)} = \text{GLCM}_{(-d, 0, 0)}^T$, the results do not change depending of the dimensions as long as there are no offsets where $\text{offset}_i = \text{offset}_j^T$ holds. We calculate those thirteen offsets for distances one through ten, and save all 130 GLCMs in a cell.

4.3 Naive feature

only done it 3D Left Hippo Features selected: 8 Imoc2: A7D2, A3D3, A13D3 Imoc1: A13D9, A7D6 Entro: A13D10, A7D6 SumAv: A7D6

FiXme Note: Tal her

FiXme Note:
Tal her

4.3.1 Calculating(Computing) the GLCM Features

FiXme Note:
Snak om
Datacalcula-
tion
filen

FiXme Note: Snak om Datacalculation filen

In the implementation of the GLCM Feature derivation we are taking two inputs. The first input variable is the GLCM matrix and the second is wether we wish to normalize the data.

What we are doing first is to make sure that all variables are implemented. Firstly we find the size of the GLCM which will be the greylevels. Hereafter we can initiate the C_x , C_y , C_{x+y} and C_{x-y} since we know the size of the GLCM.

For the pixel values in the GLCM we are using MATLAB's **ind2sub** function, that is a command that determines the equivalent subscript values corresponding to a single index into an array. We are using these variables in the GLCM Features as seen in Appendix A.

To give some examples of how C_{x-y} , look at table 4.1 where the k value is 254. This gives us 4 values, since we subtract the values, whereas it has to match $k = 254$

i	j
1	255
2	256
255	1
256	2

Table 4.1: C_{x-y} for k value 254

For a k value of $k = 249$ we have 14 different possibilities as seen in table 4.2

i	1	2	3	4	5	6	7	250	251	252	253	254	255	256
j	250	251	252	253	254	255	256	1	2	3	4	5	6	7

Table 4.2: C_{x-y} for k value 246

and for C_{x+y} that can be a max value of 512 and this will of course make both i and j to be their maximum value 256. But for 511, then we again have two possibilities as seen in table 4.3

i	j
255	256
256	255

Table 4.3: C_{x+y} for k value 511

but for lower k until we reach $k = 256$ we will have more entries for i and j as seen in table 4.4, but as we diverge from $k = 256$ we will get fewer and fewer entries.

i	244	245	246	247	248	249	250	251	252	253	254	255	256
j	256	255	254	253	252	251	250	249	248	247	246	245	244

Table 4.4: C_{x+y} for k value 500

To calculate the C_{x+y} and C_{x-y} we have two for loops as seen in Appendix A.3 and A.4 where N of course is the greylevels. As seen for A.4 then for our GLCM that is 256 we only have 255 possible outcomes whereas for A.3 we end up with a maximus of 512 outcomes!

```

1 for i = 2:2*nGrayLevels
2     cXplusY(i-1,1) = C_xplusy(glcm, i);
3 end
4
5
6
7 for i = 0:nGrayLevels-1
8     cXminusY(i+1,1) = C_xminusy(glcm, i);
9 end

```

As seen in the above snip code from our **GLCMDerivation** we have made both C_{x+y} and C_{x-y} we calculate them only once for all i and j entries.

To find the mean and standard deviation for C_x and C_y we just use the functions that MATLAB have.

The GLCM features, as seen in Appendix A, utilizes MATLABs use of vectorization. This is rewarding in the vectorized code appears more like the mathematical expressions and makes the code easier to understand and is shorter. There is often a performance gain in using vectorized code than the corresponding code containing loops.

It should be obvious for the reader to tell that the code looks alot like the mathematical expression like in Appendix A.

```

1 HXY1 = -nansum(glcm(tmpsub) .* log(cX(I).*cY(J)));
2 HXY2 = -nansum(cX(I).*cY(J).*log(cX(I).*cY(J)));
3 HX = -nansum(cX.*log(cX));
4 HY = -nansum(cY.*log(cY));
5 HXY = -nansum(glcm(:).*log(glcm(:)));

```

```

6
7 stats.angularSecondMoment          = sum(glcm(:).^2);
8 stats.contrast                    = sum(abs(I-J).^2.*glcm(tmpsub));
9 stats.correlation                 = (sum(I.*J.*glcm(tmpsub))-muX*muY)./(stdX
   *stdY);
10 stats.variance                   = sum(((I - mean(glcm(:))).^2).*glcm(tmpsub));
11 stats.inverseDifferenceMoment    = sum(glcm(tmpsub)./(1+(I-J).^2));
12 stats.sumAverage                = sum(bsxfun(@times,(2:2*nGrayLevels)',cXplusY
   ));;
13 stats.sumVariance               = sum(((2:2*nGrayLevels)-stats.sumAverage)
   .^2.*cXplusY((2:2*nGrayLevels)-1,1));
14 stats.sumEntropy                = nansum(cXplusY.*log(cXplusY));
15 stats.entropy                   = HXY;
16 stats.differenceVariance       = var(cXminusY);
17 stats.differenceEntropy        = nansum(cXminusY.*log(cXminusY));
18 stats.informationMeasuresOfCorrelation1 = (HXY - HXY1)./(max(HX,HY));
19 if (strcmp(norm, 'normalize') == 1)
20   stats.informationMeasuresOfCorrelation2 = sqrt(1-exp(-2.*(HXY2 - HXY)));
21 else
22   stats.informationMeasuresOfCorrelation2 = NaN;
23 end

```

As seen from line 19 to 23, we have an if-statement. This checks if we call our plot on the normalized data or not, since the values on informationMeasuresOfCorrelation2 end up being $\pm\infty$ when the data are not normalized. **FiXme Note: Snakke om hvorfor normalized data. Features vi har valgt fra freebourug har gjort det og måske gjort i det mente om at de kan ende med pæne værdier – i method muligvis**

FiXme Note:
Snakke om
hvorfor
normalized
data.

Features vi
har valgt fra
freebourug
har gjort det
og måske
gjort i det
mente om at
de kan ende
med pæne
værdier – i
method
muligvis

4.4 Plotting the GLCM features

Now that we have calculated the 13 GLCM features, we can plot them. Remember that one GLCM matrix have one specific distance for a specific offset, so this equals 90 GLCMs for the 2D, after some cuts and 130 GLCMs for the 3D version. To plot, you would simply have to call the function **simpleAllplot** that takes 4 inputs, the DATA which are the GLCM data, NumberOfPatients i.e. how many patients we wish to plot, looping which tells the function how many features it should count on, counting from feature one and Lastly in the **simpleAllplot** function we give us self the possibility to chose between plotting the mean values, for a specific number of patients or both.

We have discussed how our plain data is sorted when **datacalculation**, now we wish to sort it differently for our plots, so it is easier to handle. Since we have 13 GLCM features we create 13 cells to easier name our plots for the for loop sorting the data. The way we chose to sort our data is to have it in the following way `Dataset (NumberOfPatients*10, 9, 13)`. So we have 9 subplots per Feature where each subplot for every plot have distance 1 to 10

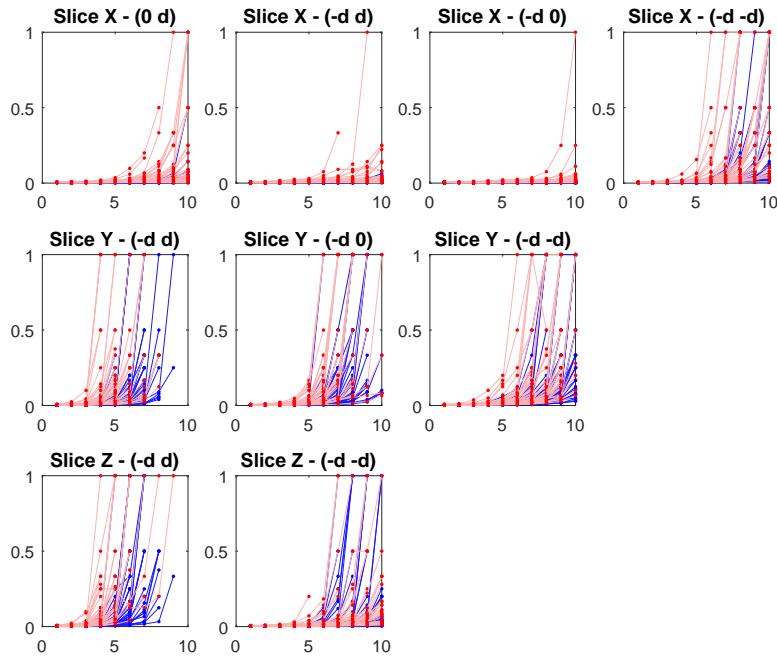


Figure 4.1: Plot of the Angular Second Moment features where the data have been normalized and eroded. This is the left Hippocampus. The red are the patients with AD and the blue are the rest

4.5 Forward feature selection

We want to use cross validation on our feature selection. To do achieve this we need to create 10-folds for our patients, which means we have to randomize the order of the patients. We use Matlab's build in function `data sample` to randomize the data and the pick five Control patients, followed by five AD, for each fold, which we continue until all 100 patients have been selected, leaving us with 10 folds with each five control and five AD.

To make the data easy to work with we sort it into a matrix, $F = (\text{No. of patients} \times \text{No. of offsets} \times \text{No. of GLCM features} \times \text{No. of distances})$. It is not necessary to split the matrix up into one for each fold, it is easier just to remember the first ten are fold one, fold two are $F(11-20, :, :, :)$, etc.

Firstly we wish to calculate how well each feature is at predicting on its own. So we create a matrix, $\text{evaluate} = (\text{No. of offsets} \times \text{No. of GLCM features} \times \text{No. of distances})$. For the two-dimensional data evaluate is a $9 \times 13 \times 10$, which is equal to 1170 different features for each patient, in three-dimensions we have $13 \times 13 \times 10 = 1690$. This huge amount of features allow us to make a preliminary cursory feature elimination, where any feature that is not complete i.e. any feature that has a NaN value for one or more of the patients we choose to ignore. In practice this is done by setting their entry in evaluate to zero. The check for NaN is done with

```
1 if (~isempty(find(isnan(data_set(:, i, j, k)) == 1, 1))) == 1
```

For the GLCM feature j calculated at offset i with distance k , it finds for all the time that value is NaN for all the patients, and checks if that set is an empty set. If the set is not empty it returns 0, which is negated and is equal to 1, so the if statement returns true and evaluate(i, j, k) = 0. We set it to zero as we evaluate each features over how well it predicts, and not how many miss classifications it makes. It is a trivial difference as 1 - success = error.

The prediction of each feature is evaluated using the function knnWithCrossval. The function splits the data up into the appropriate sets and trains a knn for each training set. We use Matlabs fitcknn function to fit the model, with euclidean distance, standardized data and for $k = 1, 2, \dots, 10$. However we run the entire feature selection for each k separately. The functions returns the averaged prediction score for the folds. We then find the feature with the highest accuracy and for the next iteration of evaluations the selected data is used in the creation of the KNN models in knnWithCrossval.

```
1 knnmmodels{i} = fitcknn(horzcat(trainKfolds{i}, chosenTrainKfolds{i}), label90, '
  ↪ Distance', 'euclidean', ...
2 'NumNeighbors', k, 'Standardize', 1);
```

Where horzcat is a horizontal concatenation of matrices. If the best evaluation of the features is worse than if no new feature is selected the algorithm breaks, and returns a matrix of selected features and iterated accuracy, as well as the last best feature not to be selected. Incase of ties for best feature we select the first entry in the matrix.

Fixme Note:
MERE?

Fixme Note: MERE?

Chapter 5

Result

5.1 Plots of 2D data

5.1.1 Left Hippocampus, normalized and eroded

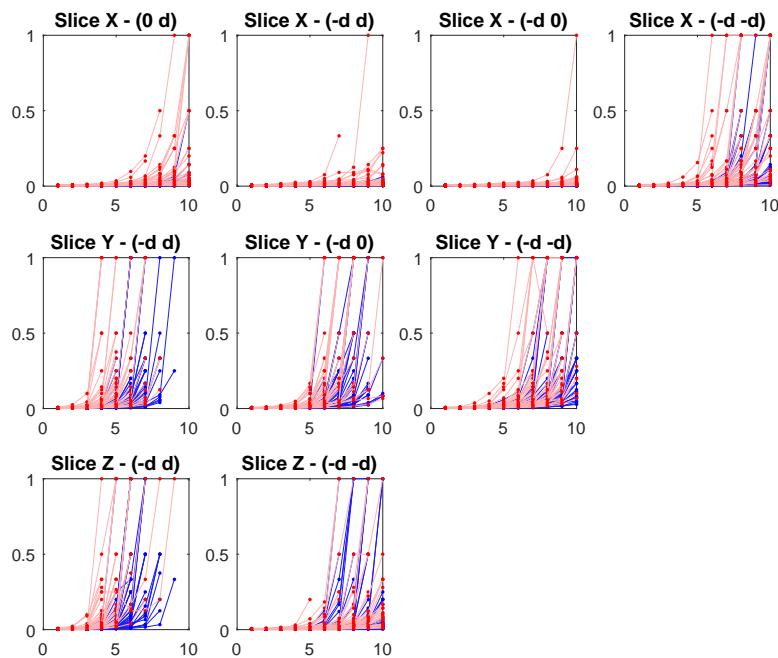


Figure 5.1: Plot of the Angular Second Moment features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest

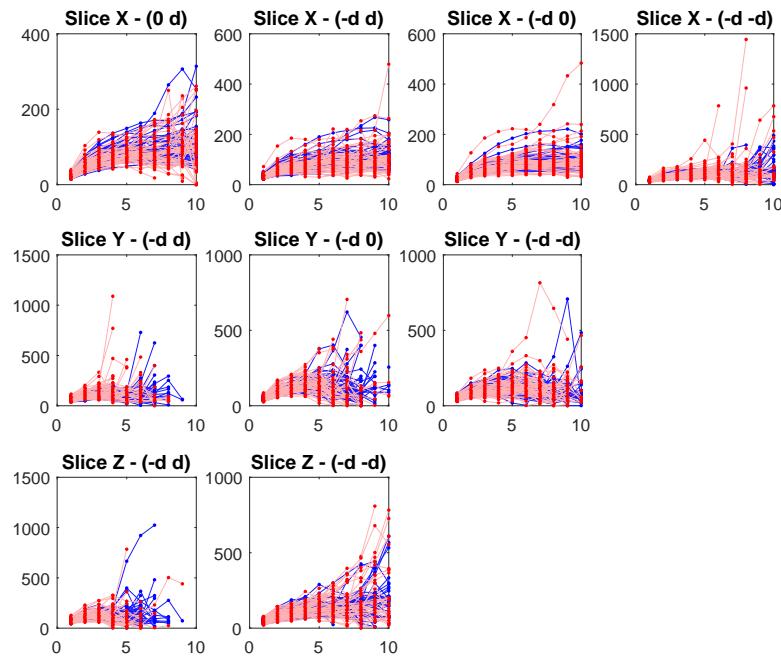


Figure 5.2: Plot of the Contrast features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest

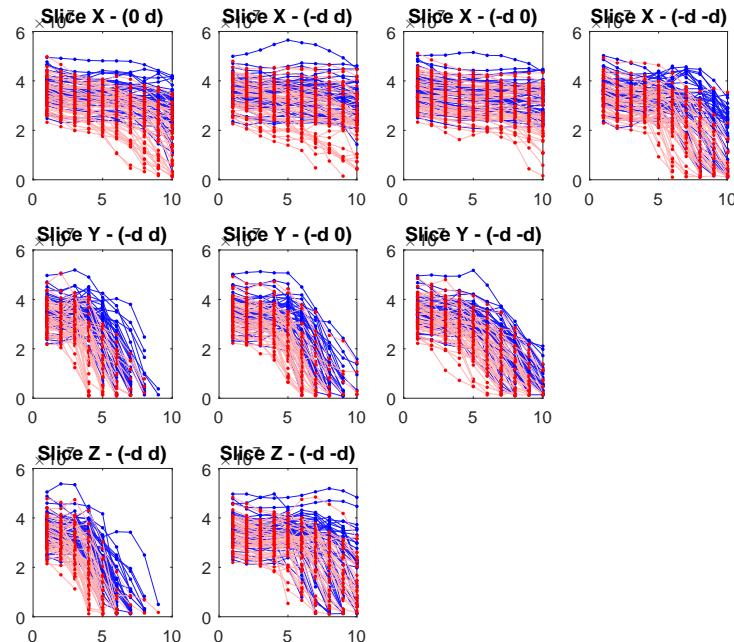


Figure 5.3: Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest

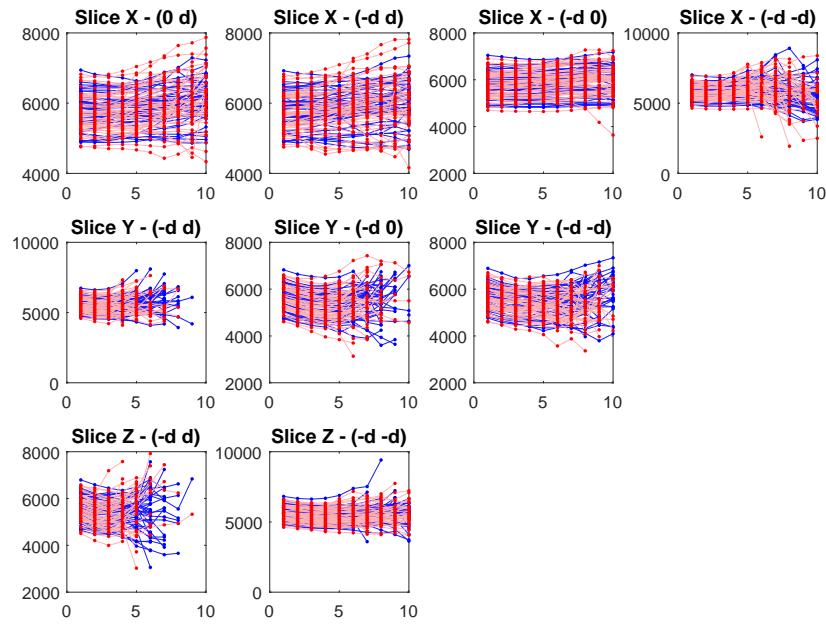


Figure 5.4: Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest

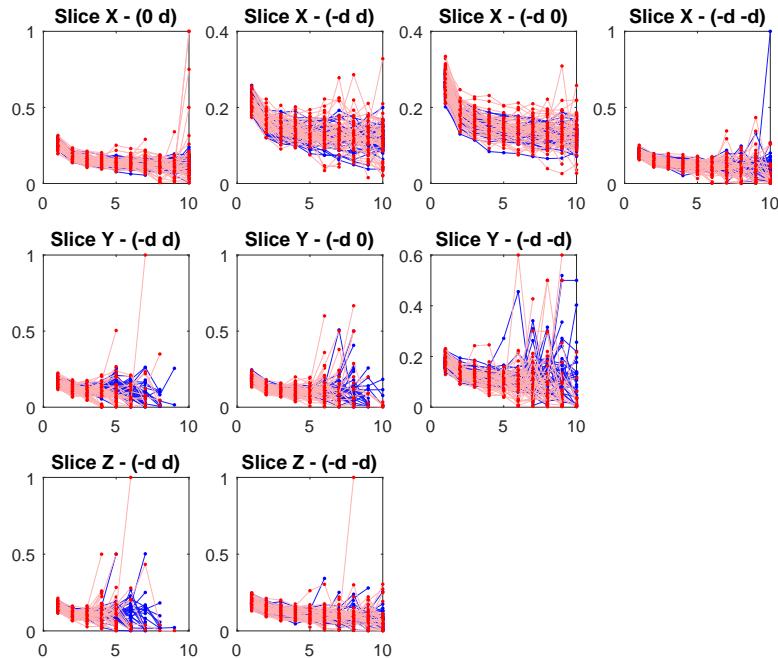


Figure 5.5: Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest

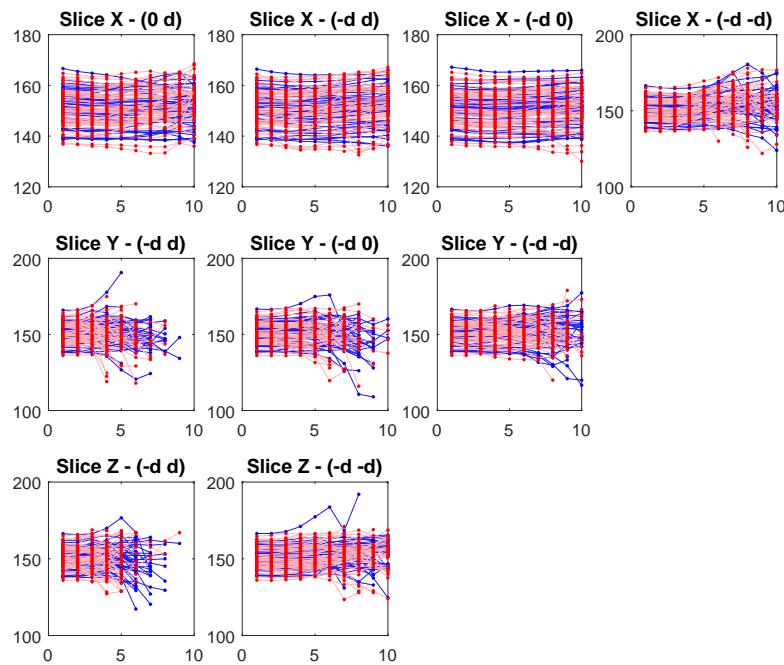


Figure 5.6: Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest

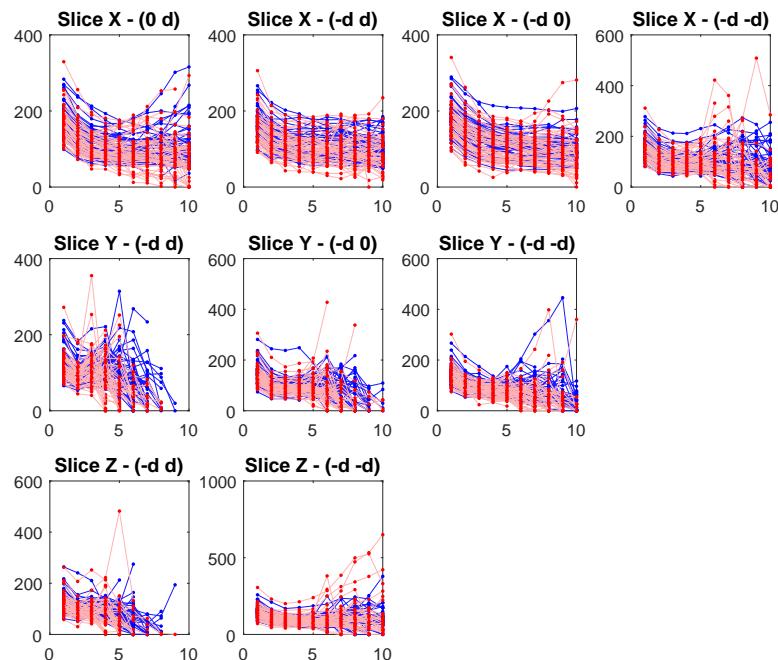


Figure 5.7: Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest

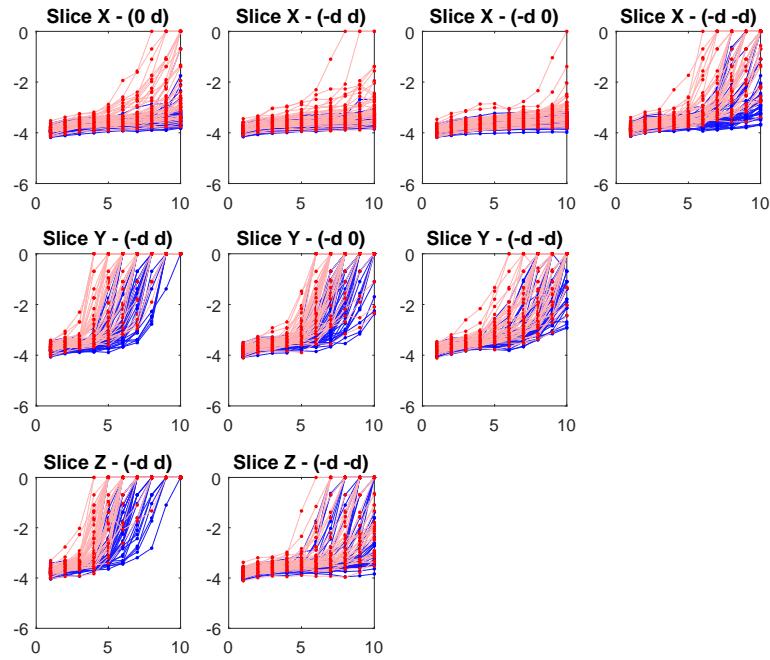


Figure 5.8: Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest

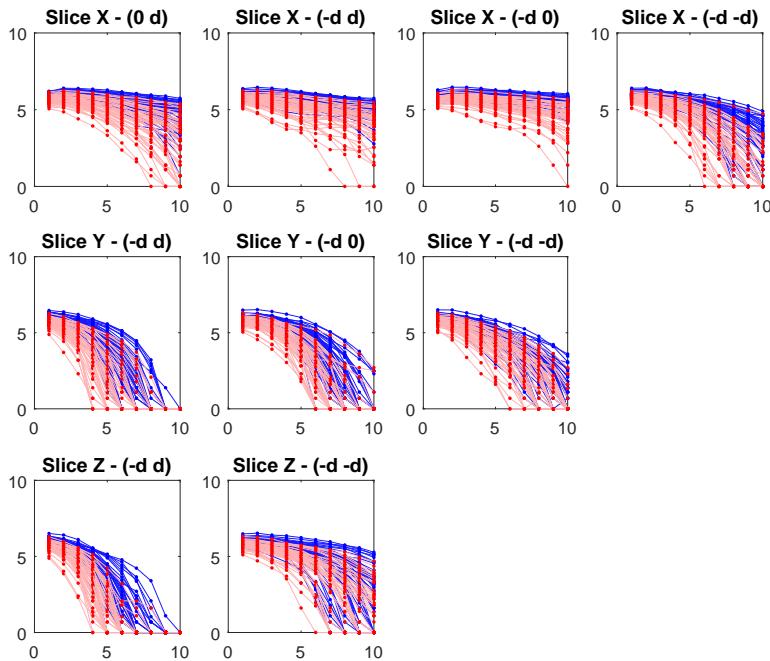


Figure 5.9: Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest

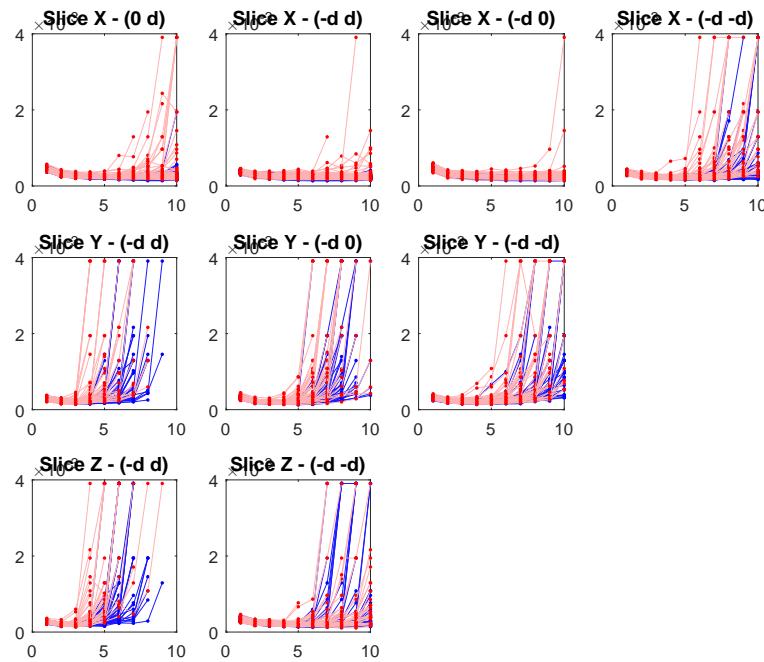


Figure 5.10: Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest

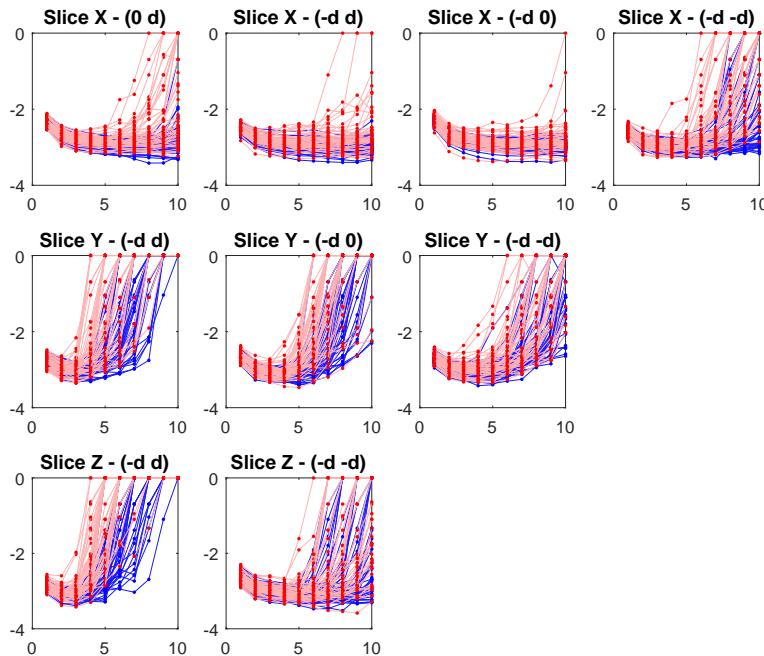


Figure 5.11: Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest

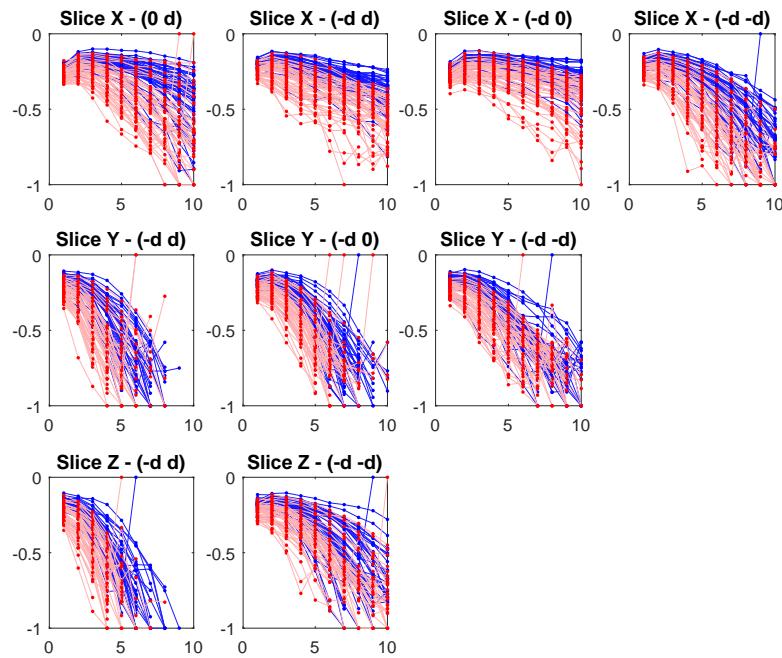


Figure 5.12: Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest

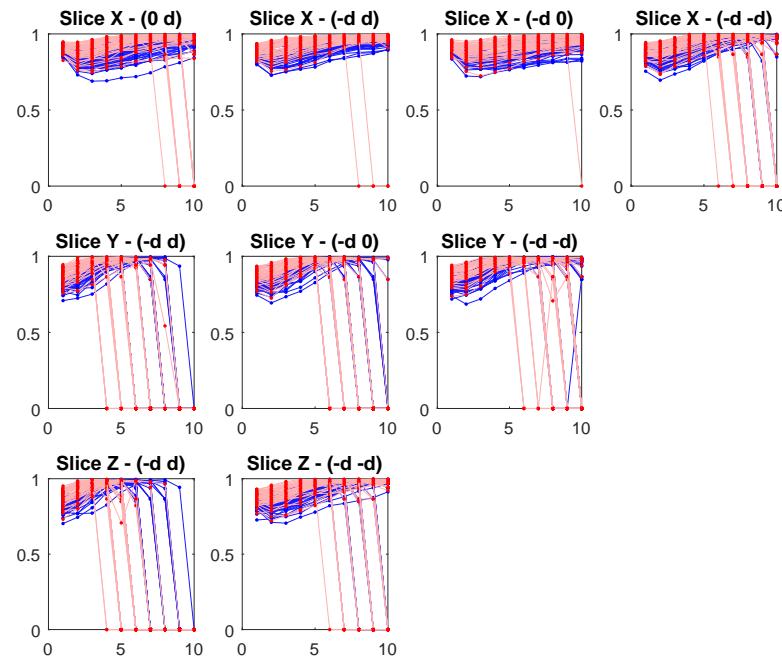


Figure 5.13: Plot of the Correlation features where the data have been normalized and eroded. The red are the patients with AD and the blue are the rest

5.2 Plots of 3D data

5.2.1 Left Hippocampus, not normalized and eroded

Chapter 6

Discussion

We wish to compare the two-dimensional (2D) GLCM methods to the three-dimensionals (3D), to see if one is superior. To make this comparison we have for our data of 100 patients extracted four data sets for left hippocampus. Data set one, D_1 , we performed erosion before calculating the GLCMs which are normalized, both are done for the 2D and 3D methods. Data set two, D_2 , is the same as D_1 except no erosion has been performed. Data set three, D_3 , is with erosion, but no normalizing. Data set four, D_4 , is without erosion and normalizing. We also wish to determine which of the left or right hippocampus is best suited to diagnose AD, if there is a difference. Data sets D_5 , D_6 , D_7 , D_8 are created as D_{1-4} but on the right hippocampus. **Fixme Fatal: Create a table to show what each data set contains?** Lastly we wish to determine if we can correctly diagnose AD with an accuracy higher than 80%.

Fixme Fatal:
Create a
table to show
what each
data set
contains?

6.1 Comparison of image textures

For each of the methods, and for each data set we have run a feature selection and fitted a model the features selected for each model. For the 3D model on D_4 there was a tie between two different k , and the accuracy for both models were calculated.

K	2DLNE	2DLE	2DLN	2DL
1	0,8816	0,9203	0,8441	0,9255
2	0,8487	0,8511	0,8688	0,9186
3	0,8882	0,8641	0,8837	0,875
4	0,865	0,8351	0,8878	0,8376
5	0,8617	0,8011	0,8997	0,8385
6	0,8483	0,8003	0,8712	0,8317
7	0,8883	0,7975	0,8713	0,8187
8	0,8841	0,7945	0,8751	0,8286
9	0,869	0,7938	0,869	0,8249
10	0,8688	0,783	0,8496	0,8266

Table 6.1: 2D: Accuracy of final KNN-model L = left, E = erosion, N = normalized

Interestingly for the the 2D method is the best accuracies occur when the data is not normalized, suggesting that their is a significant difference in the amount GIs between the two groups. Their is also a substantial drop in accuracy with increased k value for the KNN, further validating that the size difference between the groups is relevant. This is in stark contrast to the normalized data, as they do not vary a lot between the ks, there is some peaks though. There is no clear difference between the eroded and non eroded models. We erode the image with a three-dimensional model, so it is possible that the offsets in the 2D model does not get the benefit of this erosion. It could also be that the original segmentation is done very well and erosion is not necessary for these images. It does not decrease performance either, so it does not suggest relevant data is lost.

K	3DLNE	3DLE	3DLN	3DLk1	3DLk3
1	0,9223	0,8561	0,8374	0,8892	0,8436
2	0,8397	0,8954	0,7977	0,8072	0,8034
3	0,8063	0,8199	0,8914	0,8261	0,8426
4	0,7935	0,7828	0,8191	0,7697	0,763
5	0,7961	0,7617	0,8041	0,7626	0,7295
6	0,7824	0,7717	0,8002	0,7299	0,704
7	0,8048	0,7714	0,7712	0,7077	0,6922
8	0,7865	0,7777	0,7875	0,7193	0,7121
9	0,7966	0,7808	0,7795	0,7294	0,7152
10	0,7786	0,793	0,78	0,7524	0,7371

Table 6.2: 3D: Accuracy of final KNN-model, two values for 3DL since feature selection resulted in a tie between k = 1 and k = 3

The poor performance of 3DLk3 compared to 3DLk1 suggest that the feature selection overfit the data to its specific sample, as in that separation into 10-folds is biased toward a k = 3 for its solution. It also shows that k = 1 is most likely the best KNN model for that data set, as even though 3DLk3 is biased toward k = 3, it performs equally well for k = 1, where as 3DLk1 is clearly performs better for k = 1 opposed to k = 3. All the models best k value is also equal to the k returned from their feature selection, highlighting the overfitting towards that specific model done in the feature selection. For the normalized data the model created on the data set with erosion perform better than without erosion, but there is no difference on the data that is not normalized. On the data sets that is not normalized the number of observations is a relevant factor, then it is not unrealistic that the erosion removes a similar percentage of data from both control and AD, thus not being creating a difference between a model using erosion and one that does not, and would only impact their overall accuracy if there is a lot of noise in the border region of the segmentation. But as we saw in the 2D method, where erosion did not play a significant role in accuracy it is not clear it is does for the 3D method either.

The top accuracies for the 2D methods is 92.0% and 92.5% and for the 3D methods it is 92.2% which makes it very hard to declare a winner, also considering that the rest of the predictions is just below 90% the methods returns very similar results. However overfitting appears to be more prevalent in the feature selection in the 3D methods, as can be seen by the rapidly

deteriorating accuracy when k diverges from the optimal k used to select the features. The reason for the overfitting will be discussed later.

The difference in methods between 2D and 3D are in fact only four offsets, so if method based in 3D is to be superior to a 2D one the four additional angles would need to contain information describing the inter-relations between voxels better. The four offsets are $\{(d, -d, -d), (-d, -d, -d), (-d, -d, d), (d, -d, d)\}$ which translates into No. $\{6, 8, 10, 12\}$ and when we look at the features selected by the 3D methods we notice they are being used, but not exclusively.

3DL F	Offset	Feature	Distance
Feature 1	12	4	1
Feature 2	4	5	10
Feature 3	5	9	9
Feature 4	4	8	10

Table 6.3: Feature selected for 3D no erosion, without normalization

3DLN F	Offset	Feature	Distance
Feature 1	6	12	3
Feature 2	5	11	7
Feature 3	4	11	10
Feature 4	4	8	10

Table 6.4: Feature selected for 3D no erosion, with normalization

3DLE F	Offset	Feature	Distance
Feature 1	5	8	3
Feature 2	3	8	1
Feature 3	1	9	4
Feature 4	2	10	1
Feature 5	11	7	7
Feature 6	12	11	5
Feature 7	8	8	10
Feature 8	5	7	3
Feature 9	2	7	3
Feature 10	12	7	4
Feature 11	1	7	4
Feature 12	4	7	5
Feature 13	8	7	6

Table 6.5: Feature selected for 3D with erosion, without normalization

3DLNE F	Offset	Feature	Distance
Feature 1	10	13	2
Feature 2	4	13	5
Feature 3	12	13	7
Feature 4	12	8	7
Feature 5	5	8	8
Feature 6	10	8	7
Feature 7	2	13	6
Feature 8	6	8	10
Feature 9	8	13	7

Table 6.6: Features selected for 3D erosion and normalization

The first two only uses one out of four, the third uses four out of thirteen, and the last uses six out of nine, totalling to twelve out of thirty which is equal 40%. They compromise four out of thirteen of the offsets which is 30% so they are over represented, suggesting they do contain better data than the remaining offsets. They are also selected as the first feature in three of the four feature selections, meaning the single best feature is found in one of these offsets.

The advantage with using offsets compared to the radius computational wise is that despite an increase in distance the amount of angles does not increase, the amount of GLCMs to calculate are No. of offsets \times distances, where as if we were to take the radius the No. of offsets at a given distance, increases with the distance, even when we take into account that we can ignore half of the angles. For the radius the No. of GLCMs to calculate at distance d is equal to $\sigma_{i=1}^d 3 \cdot i + 1$. So for each additional distance one would have to calculate $3 \cdot d + 1$ additional GLCMs if we were using the radius, compared to just the four offsets. These calculations are just for the two-dimensional, where as in three dimensions this problem is amplified, as each increase in distance increases the number of GLCMs by $\frac{1}{2}((2d+1)^3 - 1)$. However as this clearly demonstrates the radius method would gather a lot more information, but considering that we have over a thousand features for each patients, we are not in need of more information. In addition the subject of this paper is inspired by the paper from Peter A. Freeborough and Nick C. Fox [9], so we are keeping our method similar to theirs, so it is possible to compare the results.

6.2 GOD SECTION TITEL

In the previous chapter we saw the plots of our data, which are early AD patient, more specific 24-month follow-ups and controles. As our data consists of early AD patients, it can be very difficult to differentiate one from another and thus make it challenging to get some good results, specially if we are to select some features to do machine learning. But luckily we can lean on our algorithm to select features better than we can. In consideration of that we have to feature selection method, the first one is naive selection and the second one is Sequential Feature Selection.**Fixme Note: Skriv at vi kun kigger på 3d/2d normalized erode**

Fixme Note:
Skriv at vi
kun kigger
på 3d/2d
normalized
erode

6.2.1 Naive Selection

As described previously, often there is no clear visual difference in the plots of the GLCM features, which makes it hard to make a naive selections. This forces us to look after some kind of relationship in the slope, if the data increases or decreases from a distance to another or if either AD or control have a steep slope where the other would have a straight slope. We have chosen to only select features for GLCM 3D since we **Fixme Note: argument her** and the features are normalized and eroded.

We have chosen to select the following 8 features. The Information measures of correlation 2 with the offsets $\{(0 -2 2), (0 0 3), (0 -3 0)\}$ because we can see that generally the control data have a shift down, e.g. the slope is behaving differently than the slope for the AD patients. The next two we elected to our features are information measures of correlation 1 with offsets $\{(0 -9 0), (0 -6 6)\}$ since the AD data seems to have lower values than the control and AD is more spread and has a steep sleep downwards compared to the control. Entropy with the offsets $\{(0 -6 6), (0 -10 0)\}$ is chosen because the AD seems to be more spread and have lower values than control whereas the control is more concentrated in the same spot and the slope seems to be linear for the control. Lastly we have chosen Sum Average with only one offset $\{(0 -6 6)\}$ and this is because it seems that the AD data deviate more than the control.

Fixme Note:
argument
her

We found the accuracy with crossvalidation **Fixme Note: Mere her**

Fixme Note:
Mere her

With our tests we came to the conclusion that we got the best accuracy for only 4 of our 8 features as seen in table 6.7. This table is compares the best accuracy we can get for an unspecific k but compares how many features we have to chose out of the 8.

0.79	0.83	0.84	0.85	0.79	0.79	0.79	0.79
------	------	------	------	------	------	------	------

Table 6.7: Accuracy for number of features with a unknown k value, we looked after the best accuracy. So this table tells us that no matter what, we would get the best accuracy with only 4 features selected

The highest accuracy we end up with is 85% for the 4 features selected. As you can see in table 6.8, those features are feature 4, 5, 3 and 1 which is equivalent to IMOC2 angle 7 distance 2, IMOC2 angle 13 distance 3, IMOC1 angle 13 distance 9, IMOC1 angle 7 distance 6 **Fixme Note: Skriv de offsets her**. This also suggests that a naive feature selection is not the best, since 50% of the data will lower the accuracy rather then increase it as we thought it would as seen in table 6.7.

Fixme Note:
Skriv de
offsets her

	F 1	F 2	F 3	F 4	F 5	F 6	F 7	F 8
Feature 1	0	0	0	0	0	0	0	0
Feature 2	0	0	0	0.74	0.79	0.82	0.78	0.71
Feature 3	0	0	0	0	0	0	0	0
Feature 4	0	0	0	0	0.85	0.79	0.82	0.74
Feature 5	0	0	0	0	0	0.80	0.79	0.70
Feature 6	0	0	0	0	0	0	0.81	0.79
Feature 7	0	0	0	0	0	0	0	0.72
Feature 8	0	0	0	0	0	0	0	0

Table 6.8: Where F stands for Feature #. For feature 3 and feature 1, where it seem that we get the best accuracy with feature 4 and 5. The reason that rows for feature 3 and 1 are zeros is because those features are already selected.

It should be noted that there are other feature combinations that gives an accuracy of 85%, but this is the first feature combination we run in to and the highest accuracy is 85%

With an algorithm we wish to tell wether we can get a better accuracy or not and for this we use the SFS. As described we have a limit of picking a maximum of 15 features or if the accuracy changes in a negative way. So for every feature to be selected, we choose that which at the given time describes the model with the best accuracy and this means that we can end up with n features.

For the 2D SFS have a feature accuracy of 91% for $k = 3$ as seen in table 6.9

	Feature Accuracy	Offset	Metric	Distance
Feature 1	0.81	4	8	7
Feature 2	0.87	2	13	7
Feature 3	0.89	8	1	3
Feature 4	0.9	8	13	8
Feature 5	0.91	2	1	7
Feature 6	0.91	4	13	7

Table 6.9: Six features shown for $k = 3$

This means that we have a feature accuracy of 91% with only 5 features, but it is still interesting to see how well the model can predict the data with these features

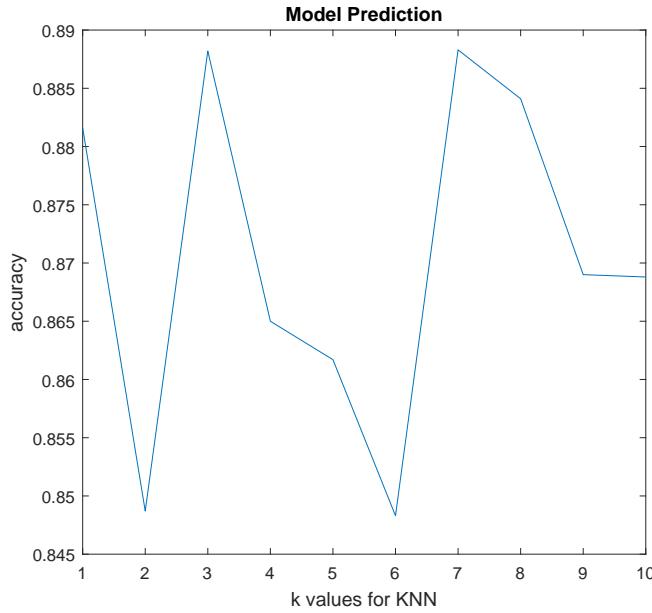


Figure 6.1: Plot of the accuracy for the 2D of our model with a feature accuracy of 91% and model accuracy with k values from 1 to 10 for KNN

It should be noted that there is a possibility that for $k = 3$ the model could be biased because we have chosen features for $k = 3$ as seen in table 6.9. As seen in figure 6.1 the model has the highest accuracy for both $k = 3$ and $k = 7$ which means that for nearest neighbours 3 and 7 we get of 88.83% which is slightly higher than the naive feature selection, but it should be noted that this is done for 2D and the naive is 3D. So let's compare the 3D SFS with the naive selection which is more fair.

With the 3D SFS it seems that $k = 1$ is in favor with the highest feature accuracy as seen in tab 6.10 with a whole of 10 features before the feature accuracy does not improve any more

	Accuracy	Offset	Metric	Distance
Feature 1	0.82	10	13	2
Feature 2	0.86	4	13	5
Feature 3	0.87	12	13	7
Feature 4	0.87	12	8	7
Feature 5	0.87	5	8	8
Feature 6	0.88	10	8	7
Feature 7	0.92	2	13	6
Feature 8	0.93	6	8	10
Feature 9	0.94	8	13	7
Feature 10	0.95	3	10	7
Feature 11	0.95	4	1	3

Table 6.10: Eleven features shown for $k = 1$

With a feature accuracy of 95% with 10 features needed, we are now interested to see what

the model can predict.

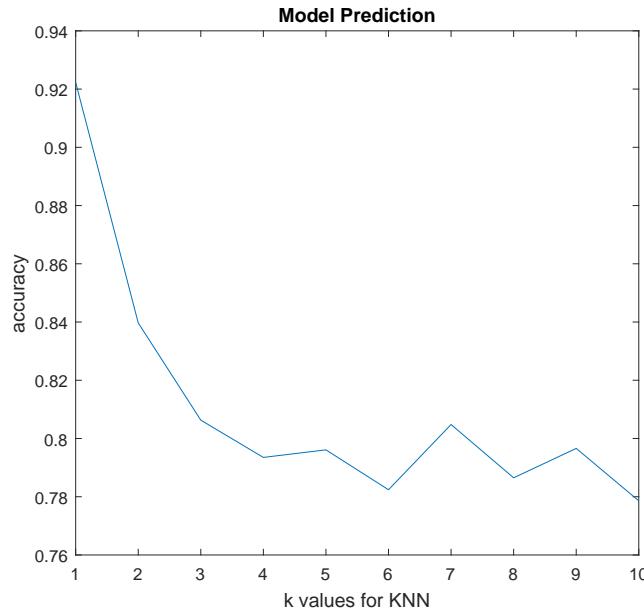


Figure 6.2: Plot of the accuracy for the 3D of our model with a feature accuracy of 95% and model accuracy with k values from 1 to 10 for KNN

As it is possible to see in figure 6.2 the accuracy is 92.23% for $k = 1$, which can be biased. This accuracy is also a lot higher than what we got with the naive feature selection. It is shown in figure 6.2 that when we expand our search to more than 1 nearest neighbour that the accuracy falls drastically.

6.3 Overfitting

In the forward feature selection algorithm we attempt to avoid overfitting by creating a 10-fold cross validation, but in order to avoid overfitting more successfully it would be a better idea to keep randomizing the folds, our implementation creates the folds firstly and then run the entire feature selection without shuffling the folds. Since we do not reshuffle the data we are making our feature selection biased towards this specific partition. We can see this bias by running a test using our forward feature selection on 60 patients, and using the last 40 to test. Running the feature selection on the data gained from using three-dimensional GLCMs with erosion and normalizing, and using 10-fold cross validation, with 6 (3 Control and 3 AD) in each folds. The feature selection results in an accuracy of 98% on k-nn model, with $k = 1$. However testing this model on the remaining 40 patients using 10-fold cross validation with 4(2 Control and 2 AD) in each fold, the accuracy is instead 80% and the best k value for the knn model is instead $k = 3$ or $k = 5$. Running the same test on the two-dimensional we get an accuracy of 95% with optimal $k = 1$ or $k = 3$, where each k does not select the same features but do have some overlap. The model created using the features from the first k gives us an accuracy of 82.50% when validating with the test set and optimal $k = 9$, the other

model has an accuracy of 80%, obtained for each of k =2,3,5,7. This biased could have been prevented by shuffled the data a significant amount times, and for each sample run a 10-fold cross-validation to test each feature, this however would be very computational heavy, as each feature selection currently takes roughly thirty minutes to run.

Start med at snakke om Normalized Erode

Sammenligne med 2D og 3D

Hvorfor er 2D bedre end 3D eller omvendt. hvilke angles er i virkeligheden forskellige?

Hvorfor har vi valgt de algoritmer vi har

Hvorfor, hvorfor ikke normalisere.

Diskutere Erode vs Ikke Erode

Left Hippo vs right Hippo

Plots af breaks i accuracy og ændring i k for knn

Chapter 7

Conclusion

Appendices

Appendix A

Co occurrence matrix derivation features

$$C_x(i) = \sum_{j=1}^N C(i, j) \quad (\text{A.1})$$

$$C_y(i) = \sum_{i=1}^N C(i, j) \quad (\text{A.2})$$

$$C_{x+y}(k) = \sum_{i=1}^N \sum_{\substack{j=1 \\ i+j=k}}^N, \quad k = 2, 3, \dots, 2N \quad (\text{A.3})$$

$$C_{x-y}(k) = \sum_{\substack{i=1 \\ |i-j|=k}}^N \sum_{j=1}^N, \quad k=0,1,\dots,N-1 \quad (\text{A.4})$$

Where A.5 is the Angular second moment

$$f_1 = \sum_{i=1}^N \sum_{j=1}^N \{C(i, j)\}^2 \quad (\text{A.5})$$

and A.6 is the Contrast

$$f_2 = \sum_{n=0}^{N-1} n^2 \{C_{x+y}(k)\} \quad (\text{A.6})$$

and A.7 is the Correlation

$$f_3 = \frac{\sum_{i=1}^N \sum_{j=1}^n ij C(i, j) - \mu_x \mu_y}{\sigma_x \sigma_y} \quad (\text{A.7})$$

where μ_x , μ_y , σ_x and σ_y are the means and standard deviations of C_x and C_y respectively.

The A.8 is the Variance

$$f_4 = \sum_{i=1}^N \sum_{j=1}^N (i - \mu)^2 C(i, j) \quad (\text{A.8})$$

and A.9 is the Inverse Difference Moment

$$f_5 = \sum_{i=1}^N \sum_{j=1}^n \frac{1}{1 + (i - j)^2} C(i, j) \quad (\text{A.9})$$

and A.10 is the Sum Average

$$f_6 = \sum_{i=2}^{2N} i C_{x+y}(i) \quad (\text{A.10})$$

and A.11 is the Sum Variance

$$f_7 = \sum_{i=2}^{2N} (i - f_6)^2 C_{x+y}(i) \quad (\text{A.11})$$

and A.12 is the Sum Entropy

$$f_8 = \sum_{i=2}^{2N} C_{x+y}(i) \log(C_{x+y}(i)) \quad (\text{A.12})$$

and A.13 is the Entropy

$$f_9 = - \sum_{i=1}^N \sum_{j=1}^N C(i, j) \log(C(i, j)) \quad (\text{A.13})$$

and A.14 is the Difference Variance

$$f_{10} = \text{variance of } C_{x-y} \quad (\text{A.14})$$

and A.15 is the Difference Entropy

$$f_{11} = - \sum_{i=0}^{N-1} C_{x-y}(i) \log(C_{x-y}(i)) \quad (\text{A.15})$$

and A.16 is the Information measures of correlation

$$f_{12} = \frac{HXY - HXY1}{\max\{HX, HY\}} \quad (\text{A.16})$$

and A.17 is the Information measures of correlation

$$f_{13} = \sqrt{1 - \exp\{-2(HXY2 - HXY)\}} \quad (\text{A.17})$$

Where HX and HY are the entropies of C_x and C_y and

$$HXY = - \sum_{i=1}^N \sum_{j=1}^N C(i, j) \log\{C(i, j)\} \quad (\text{A.18})$$

$$HXY1 = - \sum_{i=1}^N \sum_{j=1}^N C(i, j) \log\{C_x(i)C_y(j)\} \quad (\text{A.19})$$

$$HXY2 = - \sum_{i=1}^N \sum_{j=1}^N C_x(i)C_y(j) \log\{C_x(i)C_y(j)\} \quad (\text{A.20})$$

Appendix B

Feature Offset helper

B.1 For the 3D

Offset	Offset as number
d 0 0	1
d 0 d	2
0 0 d	3
-d 0 d	4
d -d 0	5
d -d d	6
0 -d d	7
-d -d d	8
-d -d 0	9
-d -d -d	10
0 -d -d	11
d -d -d	12
0 -d 0	13

Table B.1: Add caption

Bibliography

- [1] Dementia. <http://www.who.int/mediacentre/factsheets/fs362/en/>. Accessed: 2016 April.
- [2] Mathworks graycopros. <http://se.mathworks.com/help/images/ref/graycoprops.html>. Accessed: 2016 May.
- [3] Pearson product. https://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient. Accessed: 2016 May.
- [4] Fritz Albregtsen et al. Statistical texture measures computed from gray level cooccurrence matrices. *Image processing laboratory, department of informatics, university of oslo*, pages 1–14, 2008.
- [5] Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.
- [6] G Castellano, L Bonilha, LM Li, and F Cendes. Texture analysis of medical images. *Clinical radiology*, 59(12):1061–1069, 2004.
- [7] Anne Corbett Carol Brayne Dag Aarsland Emma Jones Clive Ballard, Serge Gauthier. Alzheimer's disease. *Lancet*, 377:1019–1031, March 2011.
- [8] Bruce Fischl, David H Salat, Evelina Busa, Marilyn Albert, Megan Dieterich, Christian Haselgrove, Andre Van Der Kouwe, Ron Killiany, David Kennedy, Shuna Klaveness, et al. Whole brain segmentation: automated labeling of neuroanatomical structures in the human brain. *Neuron*, 33(3):341–355, 2002.
- [9] Peter A Freeborough and Nick C Fox. Mr image texture analysis applied to the diagnosis and tracking of alzheimer's disease. *Medical Imaging, IEEE Transactions on*, 17(3):475–478, 1998.
- [10] Clifford R Jack, Matt A Bernstein, Nick C Fox, Paul Thompson, Gene Alexander, Danielle Harvey, Bret Borowski, Paula J Britson, Jennifer L Whitwell, Chadwick Ward, et al. The alzheimer's disease neuroimaging initiative (adni): Mri methods. *Journal of Magnetic Resonance Imaging*, 27(4):685–691, 2008.
- [11] Rouzbeh Maani, Yee Hong Yang, and Sanjay Kalra. Voxel-based texture analysis of the brain. *PloS one*, 10(3):e0117759, 2015.

- [12] John G Sled, Alex P Zijdenbos, and Alan C Evans. A nonparametric method for automatic correction of intensity nonuniformity in mri data. *Medical Imaging, IEEE Transactions on*, 17(1):87–97, 1998.
- [13] Bradley T Wyman, Danielle J Harvey, Karen Crawford, Matt A Bernstein, Owen Carmichael, Patricia E Cole, Paul K Crane, Charles DeCarli, Nick C Fox, Jeffrey L Gunter, et al. Standardization of analysis sets for reporting results from adni mri data. *Alzheimer's & Dementia*, 9(3):332–337, 2013.