**Section 1: Data Preprocessing**

**Norma 🡪 0**

**Else 🡪 1**

**Key Notes**

1. Last row in BIOCARD\_Entorhinal contained one instance of “UNK”. Took that to mean unknown so simply removed that row
2. One hot encoded categorical scan result
   1. QMC and OTH categories were underrepresented (less than 10 each of 744) so removed them to aid in generalizing model
3. Removed scan date – shouldn’t have an effect I think
4. Standardized data (including one-hot-encoded variables, not sure if this was a good idea or not)

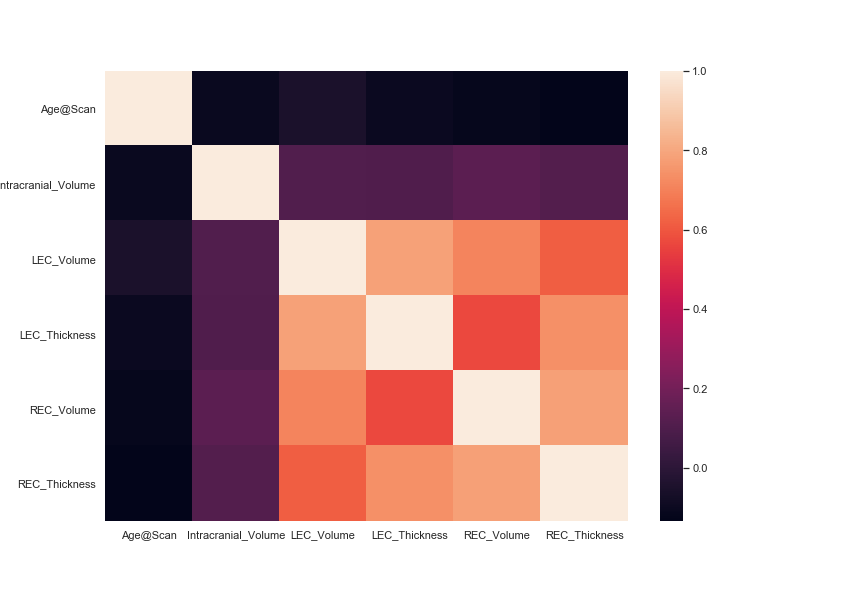
**Final Features:**

'Age@Scan', 'Intracranial\_Volume', 'LEC\_Volume', 'LEC\_Thickness',

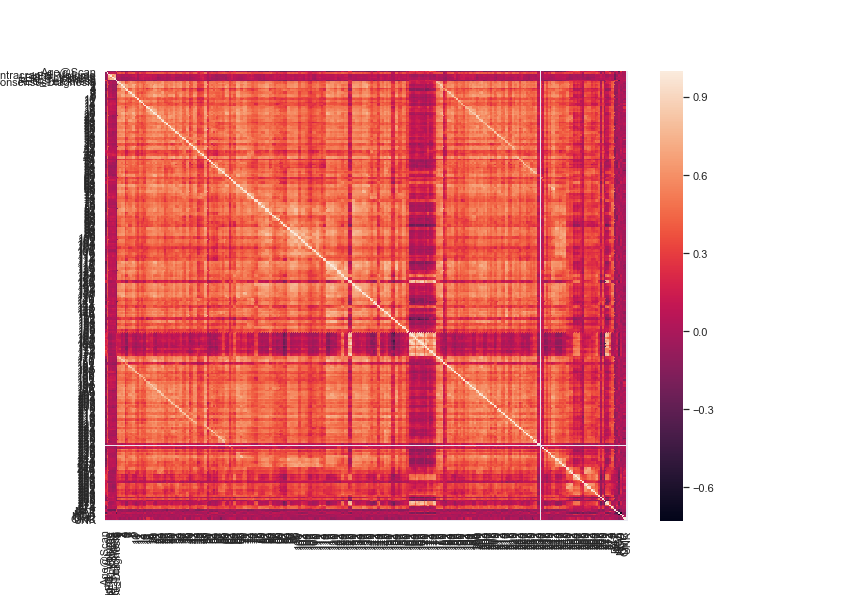
'REC\_Volume', 'REC\_Thickness', DAT, MCI, NCF, NCO

0 – 275 of L5 statistic

**Correlation Plots (checking for multicollinearity)**



Seems ok. Didn’t really check between all the files because of plot interpretability.



Hard-to-read plot for reference, correlation between all features. From briefly looking at it there are a couple areas of attention… but difficult to parse through.

**Section 2: Model Performance/Tuning**

Data was split in to 10/90 test/train.

Hyperparameter tuning was 10x cross validated executed using GridSearchCV.

**Section 3 : PCA**

After standardizing features, fit PCA transformer onto feature training set. PCA components specified to explain at least 95% of variance. Then transformed feature training, test and validation sets using the already fitted transformer. Ended up with around 100 components. Which is kind of an improvement, by factor of 2. I have 0 domain knowledge so interpretability doesn’t really matter anyways.

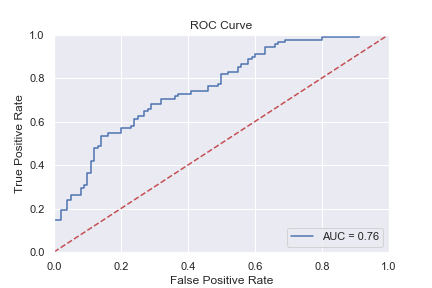
**Section 4 : Model Discussion**

**Logistic Regression**

* Decided to use Logistic Regression instead of OLS, OLS is better suited for continuous data? Need to double check w/ Hubert
* Train data was further split in to 40/60 validation/train, really wanted to avoid overfitting.

Hyperparameter Tuning

* Penalties (used both ridge and lasso regression)
  + Expect lasso shrinking to work better on non-transformed data but tested on both sets nonetheless
* Tolerance
  + Went for lower tolerances in increments of .1
* Intercept Scaling
  + Went up and down in increments.
  + Also tested for intercept on/off
* Tuned for balanced and no class weights

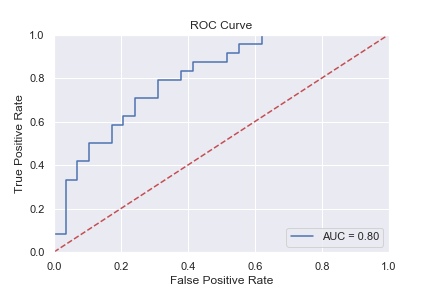


Results

Tuned Results

Balanced accuracy = 0.69

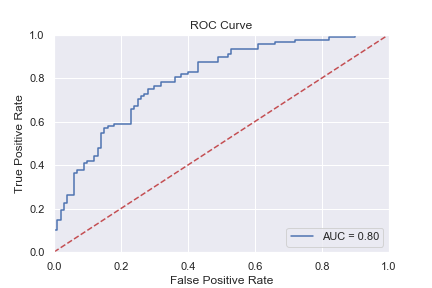
|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.68 | 0.69 | 0.70 |
| 1 | 0.69 | 0.68 | 0.67 |

Test results

Balanced accuracy = 0.71

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.76 | 0.67 | 0.75 |
| 1 | 0.67 | 0.76 | 0.68 |

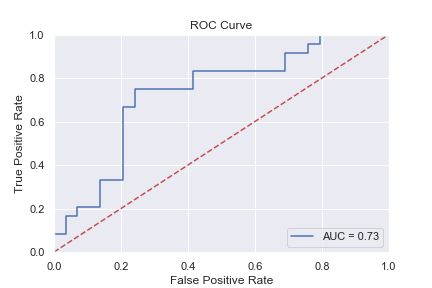
**PCA**



Tuned Results

Balanced accuracy = 0.71

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.76 | 0.67 | 0.74 |
| 1 | 0.67 | 0.76 | 0.69 |

Test results

Balanced accuracy = 0.71

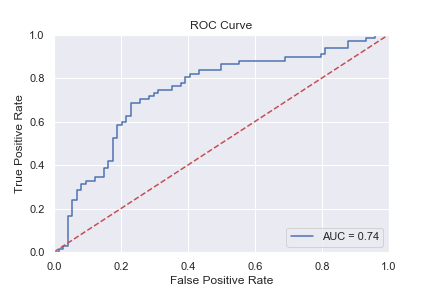
|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.79 | 0.62 | 0.75 |
| 1 | 0.62 | 0.79 | 0.67 |

**Linear Discriminant Analysis**

For this I split my training data .3/.7 . wanted to give it as much information to look at as possible

Hyperparameter Tuning

* Used least squares solver
* Tested ‘auto’ and no shrinkage
* Decreased tolerance in increments of 10
* Tested 0 and 1 component

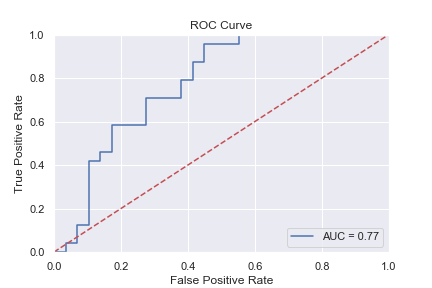
Results

Tuned Results

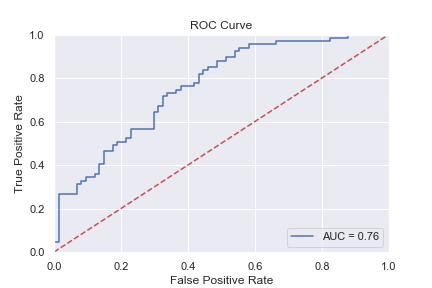
Balanced accuracy = 0.71

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.69 | 0.73 | 0.71 |
| 1 | 0.73 | 0.69 | 0.71 |

Test results

Balanced accuracy = 0.67

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.72 | 0.62 | 0.71 |
| 1 | 0.62 | 0.72 | 0.64 |

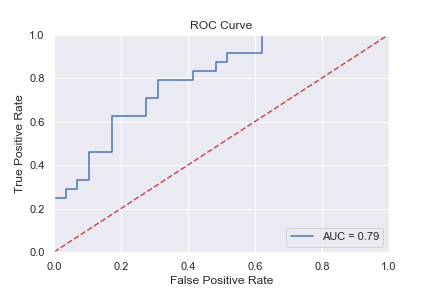


**PCA**

Tuned Results

Balanced accuracy = 0.67

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.76 | 0.58 | 0.72 |
| 1 | 0.58 | 0.76 | 0.62 |



Test results

Balanced accuracy = 0.71

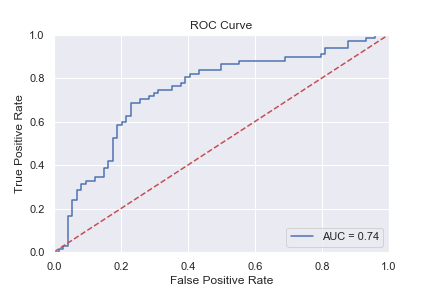
|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.79 | 0.62 | 0.75 |
| 1 | 0.62 | 0.79 | 0.67 |

**K-Nearest Neighbors**

For this I split my training data .3/.7 . wanted to give it as much information to look at as possible

Hyperparameter Tuning

* Tried with n\_neighbors = 1 to 10 (only odd numbers though)
* Tested both uniform and distance for weights
* Used “ball\_tree”, “kd\_tree”, “brute-force” algorithms



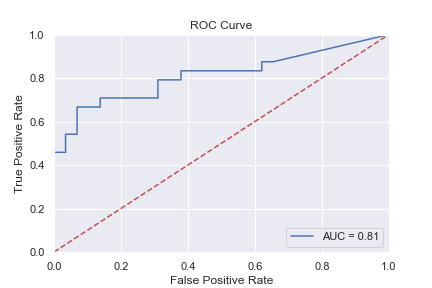
Results

Tuned results

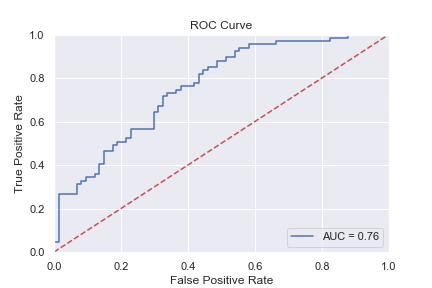
Balanced accuracy = 0.70

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.69 | 0.70 | 0.70 |
| 1 | 0.70 | 0.69 | 0.69 |

Test Results

Balanced accuracy = 0.72

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.69 | 0.75 | 0.73 |
| 1 | 0.75 | 0.69 | 0.71 |

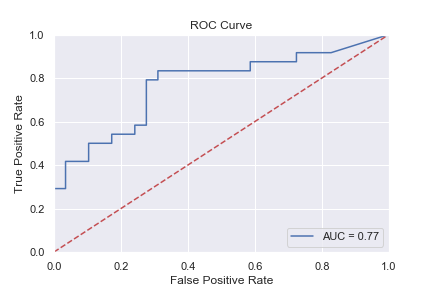
**PCA**

Tuned results

Balanced accuracy = 0.72

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.73 | 0.70 | 0.73 |
| 1 | 0.70 | 0.73 | 0.70 |

Test Results

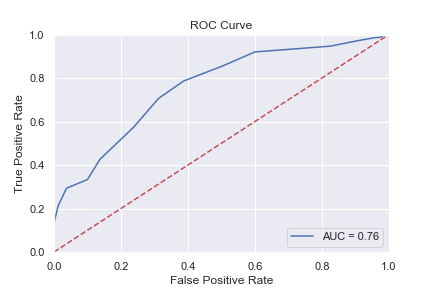
****Balanced accuracy = 0.70

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.72 | 0.67 | 0.72 |
| 1 | 0.67 | 0.72 | 0.67 |

**Random Forest**

Hyperparameter Tuning

* Surprisingly gini criterion performed best on the test set, although entropy performed better on validation.
* Tested on number of trees from 10 to 30 in increments of 5.

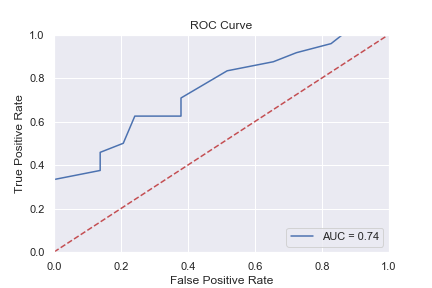


Results

Tuned results

Balanced accuracy = 0.70

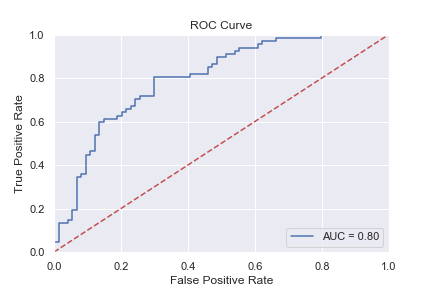
|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.69 | 0.71 | 0.70 |
| 1 | 0.71 | 0.69 | 0.69 |



Test Results

Balanced accuracy = 0.70

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.76 | 0.62 | 0.73 |
| 1 | 0.62 | 0.76 | 0.65 |

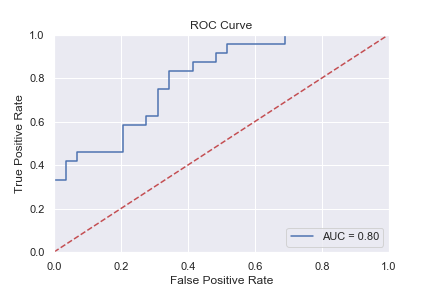
**PCA**

Results

Tuned results

Balanced accuracy = 0.74

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.70 | 0.78 | 0.74 |
| 1 | 0.78 | 0.70 | 0.74 |

****

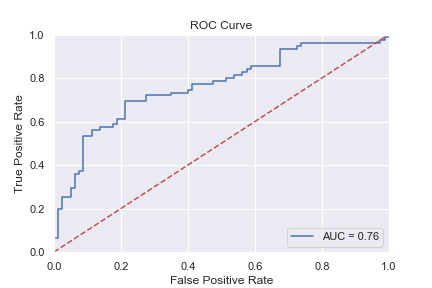
Test Results

Balanced accuracy = 0.66

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.69 | 0.62 | 0.69 |
| 1 | 0.62 | 0.69 | 0.62 |

**Neural Network**

Used sklearn’s MLPClassifier (multi-layer perceptron, not nlp lol). Sgd failed to converge so settled with newton’s method.



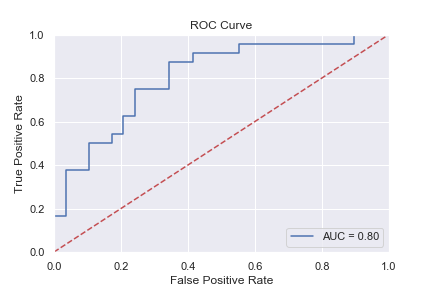
Results

Tuned results

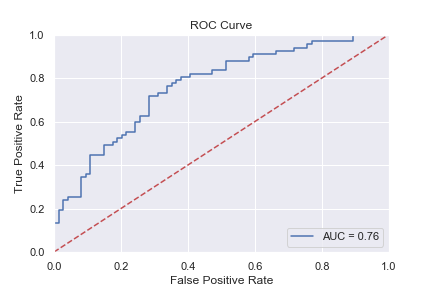
Balanced accuracy = 0.70

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.69 | 0.71 | 0.71 |
| 1 | 0.71 | 0.69 | 0.70 |

Test Results

Balanced accuracy = 0.75

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.76 | 0.75 | 0.77 |
| 1 | 0.75 | 0.76 | 0.73 |

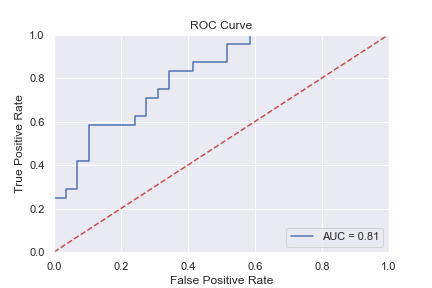


Tuned results

Balanced accuracy = 0.70

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.68 | 0.73 | 0.70 |
| 1 | 0.73 | 0.68 | 0.70 |

Test Results

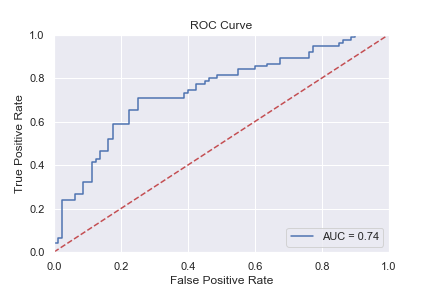
Balanced accuracy = 0.70

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.76 | 0.62 | 0.73 |
| 1 | 0.62 | 0.76 | 0.65 |

**Support Vector Machine**

* Used linear kernel

Hyperparameter Tuning

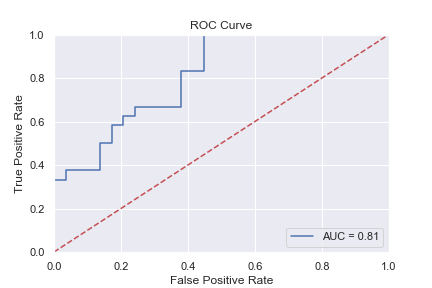
* Played around with costs and gammas, didn’t really know what I was doing but tried multiple amounts

Results

Tuned results

Balanced accuracy = 0.68

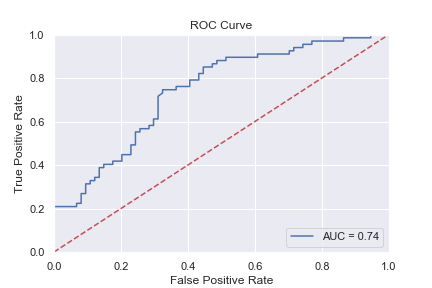
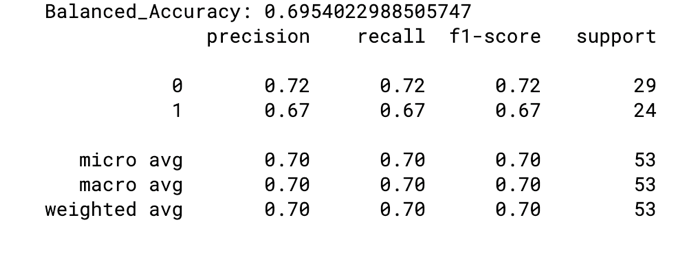
|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.66 | 0.71 | 0.68 |
| 1 | 0.71 | 0.66 | 0.68 |

Test Results

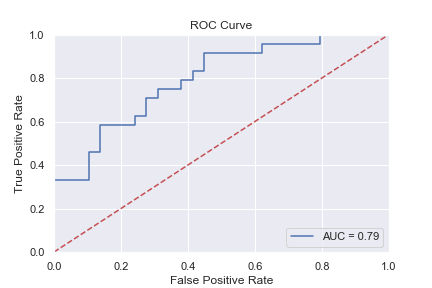
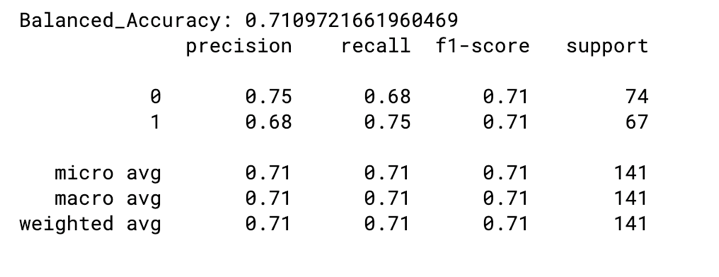
Balanced accuracy = 0.75

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.72 | 0.67 | 0.72 |
| 1 | 0.67 | 0.72 | 0.67 |

TRAIN



TEST



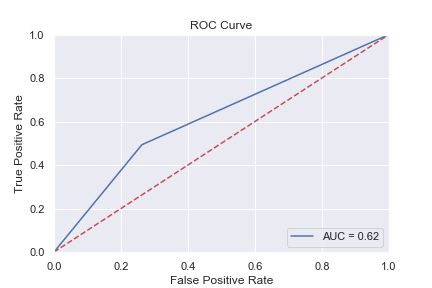
**Quadratic Discriminant Analysis**

* Pretty BS so far, have to figure out how to remove collinear variables!
* But performed decent so put it in here

Hyperparameter Tuning

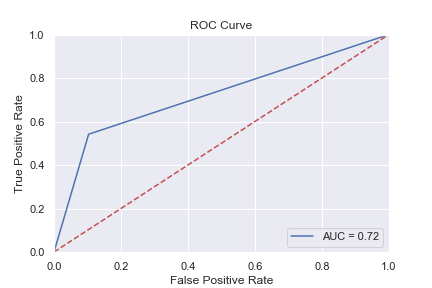
* Mainly played around with tolerance

Results

Tuned results

Balanced accuracy = 0.68

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.66 | 0.71 | 0.68 |
| 1 | 0.71 | 0.66 | 0.68 |

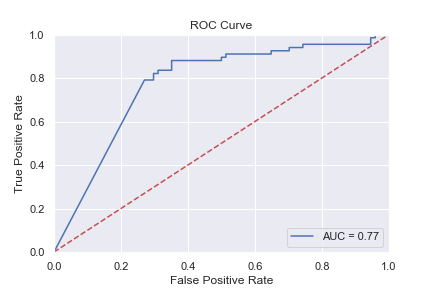
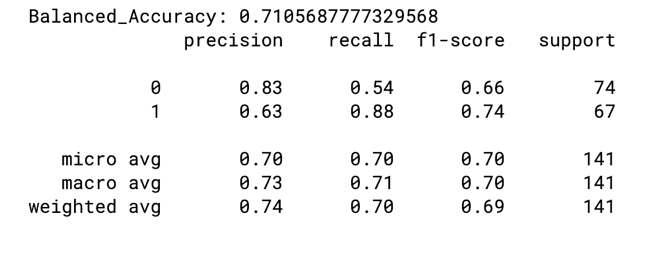


Test Results

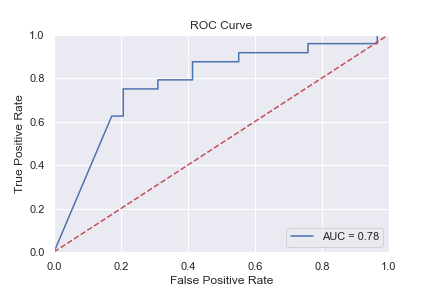
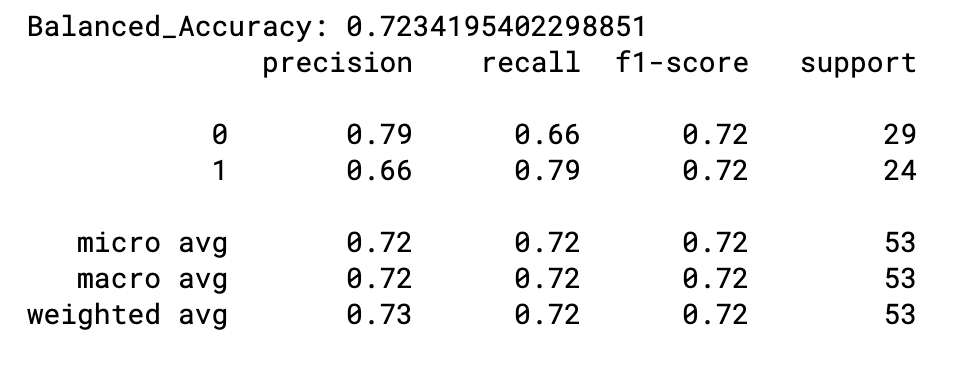
Balanced accuracy = 0.75

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recall | Specificity | F1 |
| 0 | 0.72 | 0.67 | 0.72 |
| 1 | 0.67 | 0.72 | 0.67 |

TRAIN



TEST



**CONCLUSIONS**

Most of the models hover around 70%.. which is a little weird. As this is a screening test I’d say negative recall (i.e. recall for 1) is the most important – probability of correctly predicting an unnormal brain.

With that being said PCA – decomposed QDA gave the best result for sensitivity, with 0.79. Neural Network, SVM, QDA all produced good models – with a balanced accuracy of around 0.75. But I think we can probably do better with QDA by removing collinear variables.

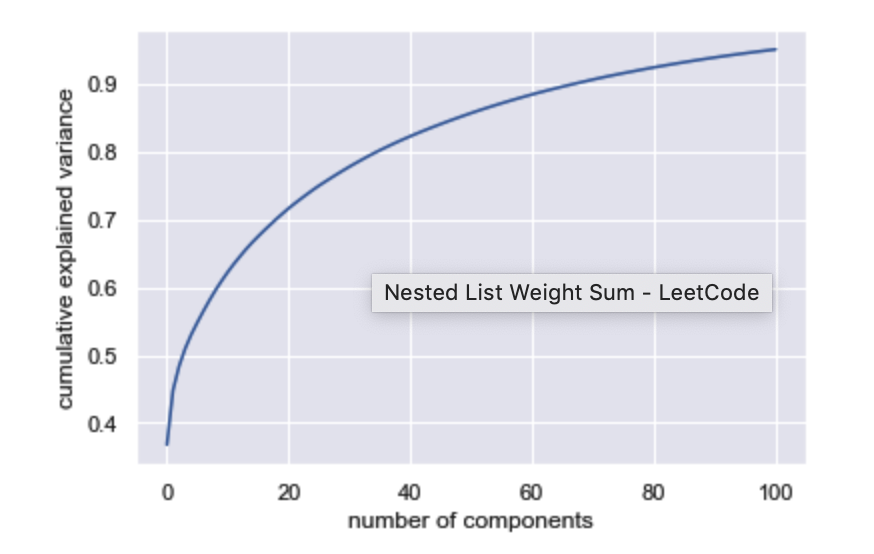
I was also a little worried about the amount of data I had. After downsampling, I was left with 522 rows. Maybe resampling would’ve been a better strategy.

My execution of PCA was also a little strange. I didn’t want to discard a lot of explanative data, but also didn’t want to hold a lot of components. I decided that information > feature compactness and settled with 100 components.

Overall, PCA performed worse across the board (using balanced accuracy as a metric) compared to nontransformed data.

NOTE\*\*

I did not think to plot this graph:



Maybe choosing 90 as a cutoff point would provide better results?

**Update**

Decided to go with the 1:2 class mix. As expected, overfitting immediately becomes a problem. Most obvious when recall for normal class spikes, and specificity decreases… leading to an objectively worse model. Accuracy hovers around 75 per usual. In the worst case, QDA led to a staggering 100% recall and 18% specificity… a little less than ideal. Neural Network performed the best. 77% accuracy, with 89 – 64 splits.

Would take a long time to take everything and push it in here so take my word for it. So, we turn to PCA to improve the model(??).

Neural Network actually performed very well on PCA. 79% accuracy, with 89 – 68 splits. Have no idea why in theory should perform the same?? But probably has something to do with the simplification of nn required to represent data.

**Update 2 🡪 look into standardization and normalization**