MACHINE LEARNING WITH FMRI BRAIN IMAGES

DATASTARS: KEVIN KAO, ANANTH SUBRAMANIAM, VALERIE LIU (CS194-16)

1. INTRODUCTION

There are numerous applications to machine learning, one of which is with making predictions on how the human brain works. Many scientists have spent careers trying to reconstruct what the mind sees by measuring brain activity. The benefit might not be obvious. After all, if we wanted to see what someone else was seeing, couldn't we attach a camera to them instead? But the interesting applications come when the brain is seeing something that the eye is *not* seeing. For example, during dreams, the visual cortex is still firing wildly even though it receives no input from the eyes at all. Other applications include seeing what patients with mental illnesses see--specifically those with illnesses that cause a warped perception of reality. This paper is just brushing the surface of this established field.

2. BACKGROUND

In this paper, we applied machine learning methods to predict brain responses to certain images. Our objective was to predict which region of the brain would be most activated in response to a certain stimulus image, as well as finer problems such as predicting the actual BOLD (blood oxygen level dependent) response of a single voxel. Specifically, overall regional brain activity and individual voxel responses were analyzed. This paper discusses our methods and findings.

2.1 EXPERIMENT OVERVIEW

The experiment was run by Gallant Labs. There were two subjects: S1 and S2. Each subject was presented 1970 images to view, one at a time, for a time interval of 1 second. Their brain activity was then measured after they were shown each image. The measurements spanned over 73,000 voxels, which are the smallest unit of measurement in an fMRI brain image.

A plethora of Matlab files were provided to us as part of the experiment's data. The two major groups of datasets were the images that were shown to the subjects and the measured BOLD responses from viewing the image. A BOLD response simply represents a brain's response to a stimulus; it is represented by a numeric value. The higher the numeric value, the more the brain responded to the stimulus. We took 0.0 to be the activation level of the brain at resting state, so negative values were interpreted as less activity than at resting state.

The datasets were split into a training and test set, where the number of training images was 1750 and the number of test images was 120. The number of voxel measurements provided to us were also cut down by approximately a third, from 73,000 to 25,000 voxels.

In the `EstimatedResponses.mat` file, there were eight sets of matrices, each representing its own dataset. The matrices consisted of:

Dataset	Matrix Size
`dataTrnS1`	`1750 x ~25000`
`roiS1`	`1 x ~25000`
`voxIdxS1`	`1 x ~25000`
`dataValS1`	`120 x ~25000`
`dataTrnS2`	`1750 x ~25000`
`roiS2`	`1 x ~25000`
`voxIdxS2`	`1 x ~25000`

dataValS2	120 x ~25000

The **dataTrnS#** datasets represent the measured BOLD responses by each of the approximately 25,000 voxels of each subject for the 1750 training images. Likewise, **dataValS#** represents the BOLD responses but for the 120 test images.

The **roiS#** datasets is an index that maps each voxel to an ROI. An ROI represents a region of interest in the brain, such as the visual cortex. In total, there were eight ROI categories, ranging from [0 – 7].

The **voxIdxS#** datasets simply assigned each voxel a unique number that maps it to some location in the fMRI brain scan. This is mostly used for labelling and voxel identification purposes.

Finally, the **dataValS#** datasets were provided as the BOLD responses to the test stimuli.

Note that all of the matrices have the same dimension of approximately 25000 – the indices of each matrix all correspond to each other (e.g. the first column of roiS1 corresponds to the first column of voxIdxS1).

We were also provided the full resolution training and test images. These images were 500 x 500 pixels grayscale. The 1750 training images were split into smaller Matlab files so that a machine with less computing power would be able to load the data. In particular, the images were split into `Stimuli_Trn_FullRes_xx.mat` files, where `xx` ranged from 00 to 15. The test images were provided in the file `Stimuli_Val_FullRes.mat`. These matrices were mostly of the form 120 x 500 x 500, i.e. 120 images per file.

3. METHODS

The first step to take was to figure out what kind of data we had and how all the datasets relate to one another.

3.1. DATA EXPLORATION

3.1.1. LOADING IMAGES

There are a handful of ways to view display the stimuli images that were shown to the subjects. These are Matlab files, so they can be loaded into Matlab and displayed using the `imshow` function in Matlab.

Rather than use Matlab directly, we loaded the images in IPython using numpy, hdf5, and matplotlib instead. The images were grayscale, so we had to set the color mapping appropriately.

FIGURE 1. A SAMPLE STIMULUS IMAGE SHOWN TO EACH OF THE SUBJECTS. THE IMAGES ARE CENTERED IN THE CIRCLE, SO THE CORNERS OF THE 500 X 500 MATRIX ACTUALLY DO NOT PROVIDE ANY USEFUL INFORMATION TOWARDS VOXEL RESPONSE PREDICTION.

3.1.2. DISTRIBUTION OF DATA

Because the number of voxels was so large, we wanted to sample smaller numbers of voxels to represent the various ROIs. To do so, we first looked at a few different distributions.

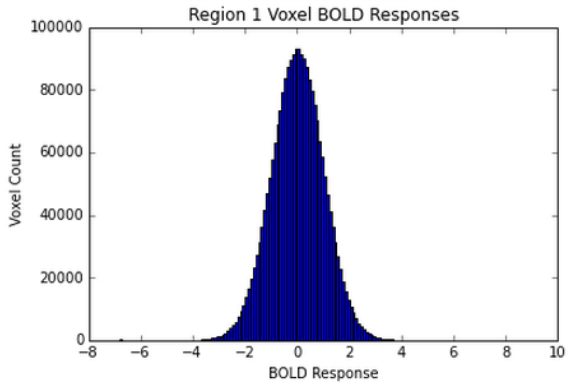
DISTRIBUTION OF VOXELS ACROSS REGIONS

The first question of interest is to determine where all of the measured voxels were located in the brain scan, i.e. what ROI was each voxel from. This information was easily extrapolated using the ROI index information.

Region 0	Region 1	Region 2	Region 3	Region 4	Region 5	Region 6	Regio
17127	1331	2208	1973	484	314	1550	928

FIGURE 2. THE NUMBER OF VOXELS IN EACH REGION CATEGORY.

From this, we could tell that the measurements were heavily skewed towards ROI 0. Approximately 66% of the voxels belonged to this region. However, ROI 0 simply represents "Other," so it most likely was several small brain regions combined or regions that are not yet currently distinguishable.



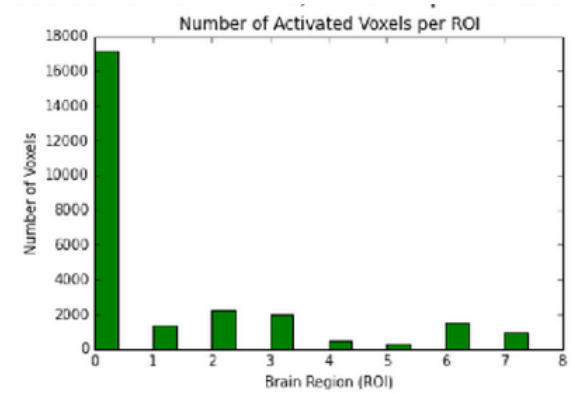


FIGURE 3. A HISTOGRAM OF THE DISTRIBUTION OF VOXELS AMONG REGIONS.

DISTRIBUTION OF VOXEL RESPONSES BY REGION

After removing outliers from the histogram above, it is evident by visual perception that the BOLD responses from voxels of a particular region are approximately distributed as standard normal variables across images. Visualizing the distribution was key in order to set appropriate voxel activity thresholds and label voxels as on or off for each image. Since the data appears to be normally distributed, we were able to directly set thresholds based on voxel activity percentiles.

3.2. DATA WRANGLING & CLEANING

3.2.1. BOLD RESPONSES

The measured BOLD responses provided was not a clean dataset. There existed numerous voxels whose entire set of measurements were NaN values. We filtered those voxels out since there would be no relevant information to predict or learn from such voxels.

For other voxels that contained some NaN values, there were three options to take:

1. Replace the NaN values with 0.0 as the measurement, i.e. resting state.
2. Replace the NaN values with the mean BOLD response level for that particular voxel.
3. Filter out the NaN BOLD responses and leave them out from the training.

We selected option 3 above primarily because we did not want to assume the behavior of individual voxels for particular images.

3.2.2. STIMULUS IMAGES

To train the models, we sought to derive features from the images. At first, our feature matrix consisted of raw pixel values. However, we soon realized that was an overly simplistic approach.

Instead, we researched notable methods of featurizing images and pursued edge detection and Gabor filters. To perform corner detection on the images, we installed the OpenCV package. Specifically, we used the Python wrapper around OpenCV.

Within OpenCV, there were several corner detection algorithms, such as the Harris corner detection method, SIFT, SURF, etc. The first method we attempted was a method

developed by OpenCV's developers, called ORB (Oriented FAST and Rotated BRIEF). Mostly, our choice was due to the numerous patents on the other detection methods.

With the ORB algorithm, we were able to pick out 10 corners of each image. However, it was difficult to match the features of one image to the features of another using the corners method.

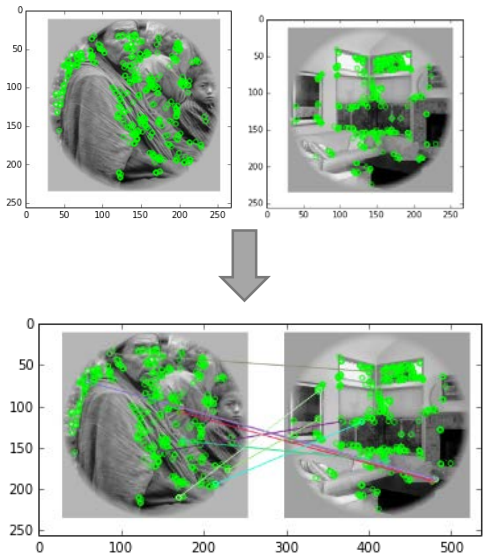


FIGURE 4. THE CORNERS, AS DETECTED BY THE HARRIS CORNER DETECTION ALGORITHM. MATCHING THE FEATURES WERE DIFFICULT.

After some experimentation, we decided to move on to using the skimage package, part of SciPy. In skimage, there are built-in algorithms for various filters to apply onto images. Our next feature to try was edges, so we chose to run skimage's Canny edge detection algorithm on all 1750 training images, generating 1750 new images that highlighted the edges of the original stimuli images. Once transformed, we attempted to draw a correlation between the edges of an image and the voxel BOLD responses but were unable to find anything concrete.

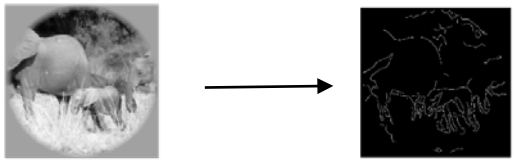


FIGURE 5. EDGE DETECTION USING THE CANNY EDGE DETECTOR.

Alternatively, Gabor filters are a well-known technique to model human perception. Gabor filters can be used to detect both edges and texture. Though Log-Gabor filters are said to outperform Gabor filters, we were unable to find a suitable implementation as skimage did not include one.

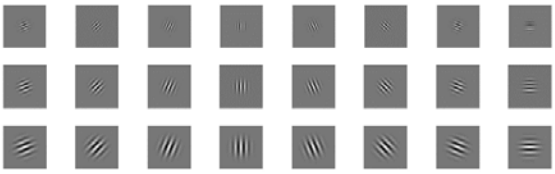


FIGURE 6. SAMPLE GABOR FILTER BANK, WHICH HAS SEVERAL GABOR KERNELS AT DIFFERENT ORIENTATIONS AND SPATIAL FREQUENCIES TO APPLY ON AN IMAGE.

Instead of using the whole response matrix after applying the Gabor kernel, we decided to featurize the images based on various statistics of the response matrix. We built a Gabor filter bank, with Gabor kernels representing various edge orientations and spatial frequencies. The spatial frequencies we used were 1.0, 2.0, 5.0, 6.0, and 12.5. The orientations we used were 0, 22.5, 45, 67.5, 90, 112.5, 135, and 157.5. These numbers were chosen to spread out the span of features we covered and have a larger chance of detecting a significant feature.

Once an image was transformed with a Gabor kernel characterized by a pair of spatial frequency, edge orientation value, we pulled out the following statistics as features from the response matrix:

1. The mean of the matrix values
2. The variance of the matrix values

3. The sum of squares of matrix values (local energy)
4. The sum of absolute values of matrix values (mean amplitude)

Since the output feature matrix still contained several values, using scikit-learn, we ran PCA (principal component analysis) on the feature matrix to eliminate redundant or useless features.

Overall, feature extraction was an expensive and difficult process without domain expertise and more resources in both time and computing. Featurizing with the Gabor kernels took well over 5 hours on 1750 images, so repeated changes in featurization was very slow and inefficient. This significantly hindered the process of making tweaks to features.

3.3. Learning From The Data

During our exploration of the data, we decided to try a variety of different machine learning techniques to extract information from the data.

3.3.1. K-MEANS CLUSTERING

The K-Means clustering technique was the first method we attempted on the dataset. The input was a 1750 by approximately 25,000 matrix, which represented 1750 training images and 25,000 voxels. Each cell in the matrix was a BOLD response.

The clusters, then, would represent images that are clustered by their overall voxel BOLD responses. We tried four different cluster sizes: 29, 50, 100, and 200. Since we didn't know how many clusters (labels) there should actually be for the dataset, we chose multiple cluster sizes to see which model performs best. The cluster size 29 was chosen based on the square root of half of the number of data points, which in this case was 1750 images.

We looked at the inertia values to see how tightly knit the clusters were. In general, the closer they are, the better the cluster; therefore, lower inertia scores is better, to a point. Another internal evaluation metric was silhouette plots. In general, for each cluster in the silhouette plot, we want their widths as close as possible.

Overall, with our tests, the inertia declined with an increase in clusters, which was expected. By this metric, the optimal number of clusters is 200; however, K-Means cannot be judged purely by inertia. To examine the clustering from another angle, we looked at the silhouette plots for each cluster size.

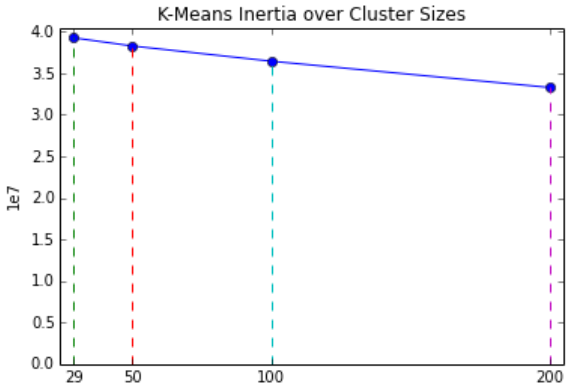
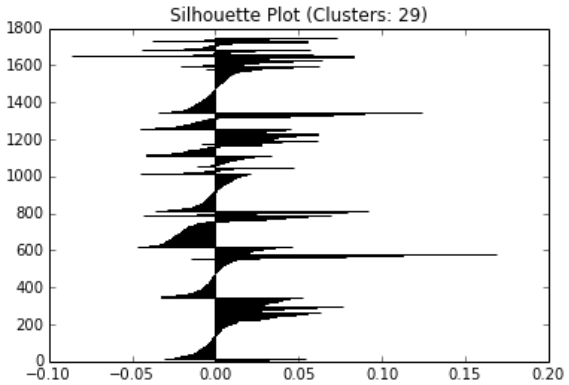


FIGURE 7. THE INERTIAS OF FOUR DIFFERENT CLUSTER SIZES. THE LOWER THE INERTIA, THE BETTER (IN GENERAL). OTHER METRICS SHOULD BE COMBINED TO EVALUATE; ONE CANNOT RELY SOLELY ON INERTIA.



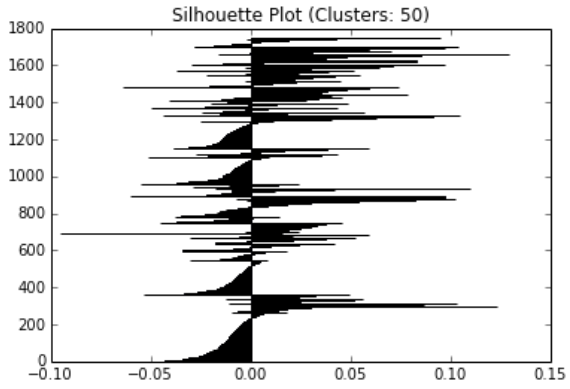


FIGURE 8. THE SILHOUETTE PLOTS FOR TWO CLUSTER SIZES, 29 AND 50. EACH SET OF PEAKS IN THE SILHOUETTE PLOTS REPRESENT A CLUSTER. IN GENERAL, WE WANT EACH CLUSTER TO HAVE SIMILAR VALUES, SO THE "FLATTER" EACH PEAK IS, THE BETTER. THE LOWER THE INERTIAS, THE BETTER. A BALANCE BETWEEN THE TWO IS NECESSARY.

In general, the silhouette scores appeared to be relatively balanced in distance, but still slightly favor their own center over their nearest neighboring cluster. The average size of a cluster was also fairly equally distributed. The $k = 29$ clusters model appears to be a fairly good model, not overfitting while yielding the highest average silhouette score, which indicates a reasonable choice for the value of k.

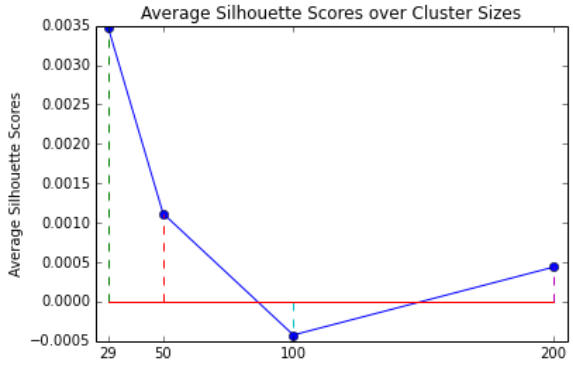


FIGURE 9. THE SILHOUETTE SCORES WERE GENERALLY POSITIVE EXCEPT FOR 100 CLUSTERS, WITH 29 AND 50 CLUSTER MODELS PERFORMING THE BEST.

3.3.2. REGRESSION MODELS

To estimate the BOLD responses for each voxel, we attempted to train a handful of regression models with the Gabor kernel statistics as features for each image. Given an input feature matrix that represents an image, we would output the estimated BOLD response for a voxel.

One main issue was resource consumption: to train regression models to estimate voxel responses, approximately 25000 models would need to be trained, one for each model. Given the time and resource constraints of the EC2 server, we decided to inspect performance on only a handful of voxels. These voxels were selected as voxels with the most BOLD activity (highest mean activation) and had no NaN values.

We tried three different regression models: linear regression, support vector regression, and decision tree regression (using the RBF kernel). We divided our given training set into 5 equal sets, each of size 350. At any time, 4 sets were used for training and 1 set was used for validation. Using cross-validation, we tuned hyperparameters of the model, in particular the penalty weight, and selected the one which resulted in the highest average performance across all validation sets. For each model, we used the default parameters.

Our results were unexpectedly bad. Scores were based on the coefficient of determination R^2. The higher the score, the better. The training score of the linear regression model was close to 0.0, while the support vector and decision tree models scored 0.85 and 0.93 respectively. On the test set, the scores were all negative, though close to 0.0 (approximately 0.001). Since the kernel used for SVR was the RBF kernel, we strongly suspect that the SVR model have been overfitting due to its ability to separate anything with infinite dimensions. Additional tuning of the gamma parameter for the SVR and max number of features/depth for the decision trees did not yield any noticeable improvements.

The conclusion we drew from the regression results was relatively consistent: the regression approach underperformed. Perhaps with a different featurization of the images, the regression models would be able to perform better. Naturally, in the future, we'd like to look into the residual plots and regularization tuning.

3.3.3. CLASSIFIERS

A variety of classifiers were available for our use.

MULTICLASS CLASSIFIERS

The problem we tackled with multiclass classifiers was to predict the ROI with highest mean activity for a given image.

Again, we used the feature matrix derived from the Gabor filter bank. This time, we also provided labels for each image. The label of each image was the ROI which had the highest mean activation level across its voxels.

The first two models we trained were the linear support vector classifier and the C-support vector classifier, which we used the RBF kernel with. We trained on 1400 images and saved 350 images for test, as we did previously. The training accuracies for each were 0.25 and 0.26, respectively. The test accuracies were 0.26 and 0.27. Though the scores of the two classifiers were similar to each other, the linear SVC was extremely inconsistent and would score from a range of 0.10 to 0.27 for its accuracies.

We decided to explore a few more classifiers in scikit-learn: decision tree classifier, random forest classifier, extra trees classifier, and the Ada boost classifier. Mostly, we wanted to compare performance of these classifiers to the original models. Most of the scores after training were similar, though slightly higher.

TABLE 1. THE TRAINING AND TEST ACCURACY SCORES OF EACH CLASSIFIER IS SHOWN BELOW. FOR RANDOM FOREST, EXTRA TREES, AND ADA BOOST, DIFFERENT NUMBER OF ESTIMATORS WERE USED.

Classifier	Training	Test
Decision Tree	1.0	0.25
Random Forest (10)	0.983	0.24
Random Forest (20)	0.998	0.28
Random Forest (30)	1.0	0.27
Extra Trees (20)	1.0	0.24
Ada Boost (20)	0.31	0.29

The performance across the classifiers was still not spectacular, but we were moving in the right direction with higher than 0.0 accuracy scores. Based on our runs with the various classifiers and the produced accuracy scores, the Ada Boost classifier performed best on the test set. A quality metric we wanted to use was the ROC AUC scores, but there was no clear and simple way of obtaining the scores for multiclass classifiers.

Since the training scores were so high in most of the cases but the test scores were much lower, we again suspect overfitting in the models.

BINARY CLASSIFIERS

To simplify the problem further, we decided to train a classifier for each ROI. The image label would be 1 if the most activated ROI for that image matched the ROI of the current classifier, and would be 0 otherwise. Essentially, all the other ROI other than the classifier's ROI were collapsed into one category.

Since these were binary classifiers, we could also evaluate the ROC AUC scores for each model, as well as the precision/recall. No ROC curves or precision/recall curves are shown because for there was no clear "thresholds" to vary over for this problem.

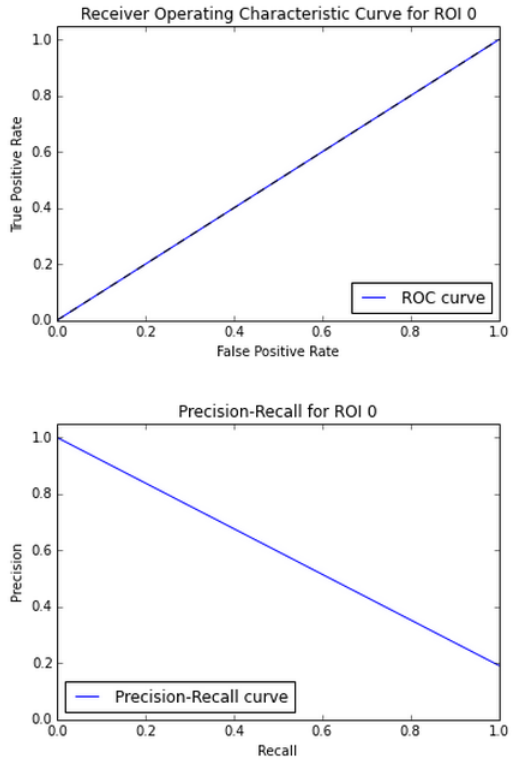


FIGURE 10. WITH ONLY TWO THRESHOLDS FOR BINARY CLASSIFICATION, THE ROC AND PRECISION/RECALL CURVES ARE NOT VERY USEFUL. MOST OF OUR EVALUATION, AS A RESULT, IS BASED ON ACCURACY.

Although we were not part of the graduate class, we decided to also create a baseline model to compare performance. The baseline model for each ROI is one that always assigns 1 or 0, depending on whether the training data had more 1 labels or 0 labels. This yielded relatively high accuracy on the test data.

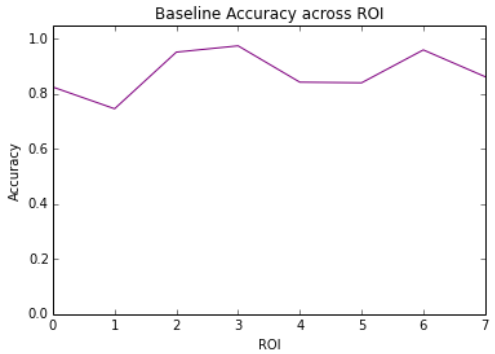


FIGURE 11. BASELINE MODEL TEST ACCURACY FOR EACH ROI WAS RELATIVELY HIGH.

Again, we tried the linear SVC and C-SVC models first. This time, we evaluated the performance of the models over various penalty scores (C values) to have a wider selection of models to explore. Overall, the performance of these two were much higher in the binary classification case. The average cross-validation scores across the linear SVC model was around the 85% accuracy mark, while the C-SVC models all consistently struck a mean cross-validation accuracy of 0.875.

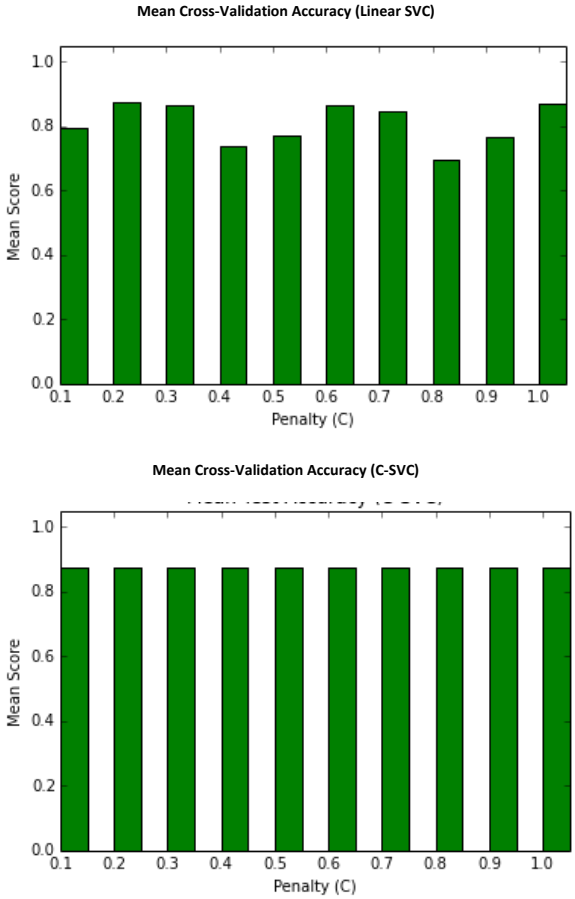


FIGURE 12. A BAR PLOT OF THE AVERAGE CROSS-VALIDATION ACCURACY SCORES ACROSS THE ROI CLASSIFIERS FOR EACH PENALTY SCORE VALUE (C). LINEAR SVC PERFORMED BEST WHEN C = 0.6, WITH AN AVERAGE ACCURACY OF 0.87. C-SVC PERFORMED BEST WITH C = 1.0 . THE BEST LINEAR SVC MODEL COULD HIT UP TO 97% ACCURACY, ON AVERAGE OVER EACH OF THE ROI CLASSIFIERS.

With high scores from these models already, we decided to only try two other classifiers: the decision tree classifier and the random forest classifier. The parameters we adjusted for the decision tree classifier was its max number of features, which could be drawn from the number of features the feature matrix has, the square root of the original number of features, and the log base 2 of the original number of features.

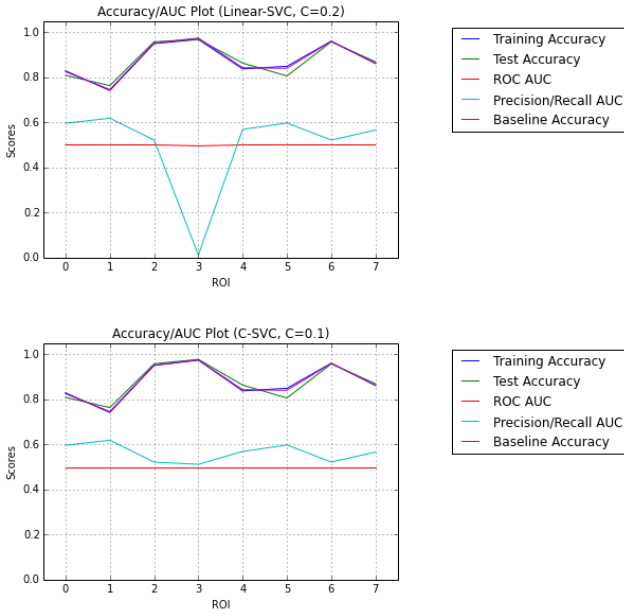


FIGURE 13. THE PLOTS OF VARIOUS QUALITY METRICS FOR THE BEST PERFORMING SVC MODELS. WHILE ACCURACY WAS HIGH FOR BOTH, THE ROC AUC WAS ONLY 0.5, AND PRECISION/RECALL AUC HOVERED JUST SLIGHTLY ABOVE 0.5 FOR THE C-SVC.

For the decision tree classifier, there was no clear winner in terms of the best parameter value to use for max features. All of the mean cross-validation accuracy scores were 0.79 with a deviation of less than 0.1 among the three scores.

Because there was no distinguishable improvement with using decision tree classifiers, we moved forward with the last model: random forest classifiers. The parameter we tuned for the random forest classifier was the number of estimators (features) to use per tree. We varied this from 10 to 45, at steps of 5.

Again, the parameter tuning did not yield any distinguishable results, but the accuracies with the random forest classifier were higher than those of the SVC classifiers by about 0.02.

Overall, the binary classification task was much more successful, albeit at the expense of less information gain than the previous problems. All of the classifiers performed with high accuracies and yielded comparable results to the baseline model.

We did look into the feature weights from the decision tree classifier and found that there was no dominant feature based on the weights. Several features had a weight of 0, and those with a weight had relatively low ones that were on par with other non-zero weights. No weight was significantly higher than the others. This suggests that the model could be a set of weak features combined together. Perhaps from these results, we can draw a conclusion that brain regions are not tuned to specific features like edge detection, but instead survey or observe combinations of features within each region.

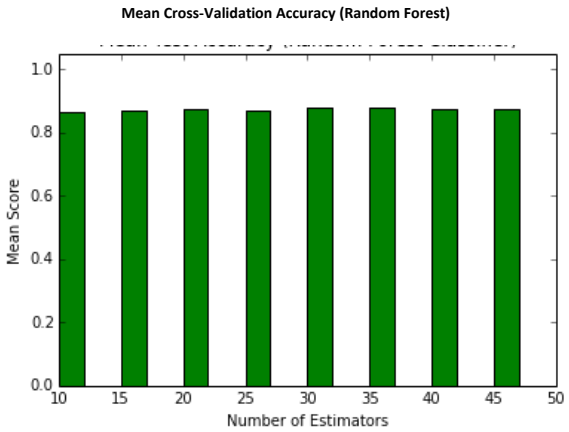


FIGURE 14. AVERAGE CROSS-VALIDATION ACCURACY SCORE FOR THE RANDOM FOREST CLASSIFIERS.

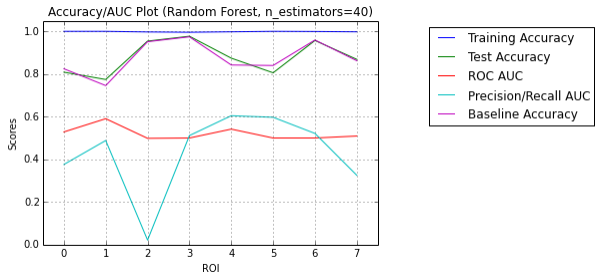


FIGURE 15. THE BEST PERFORMING RANDOM FOREST CLASSIFIER'S QUALITY METRICS.

TABLE 2. THE MEAN CROSS-VALIDATION ACCURACIES OF THE RANDOM FOREST CLASSIFIERS VARIED OVER VARIOUS PARAMETER VALUES ARE SHOWN BELOW. THE ACCURACIES WERE VERY SIMILAR, WITH 30 WINNING OUT SLIGHTLY.

n_estimators	Mean Cross-Validation Accuracy
10	0.8674
15	0.8714
20	0.8743
25	0.8732
30	0.8742
35	0.8735
40	0.8774
45	0.8746

4. TOOLS

4.1. DATA EXPLORATION

A variety of languages presented themselves as options for data exploration. Mostly, we were choosing among Python, R, and Matlab. We reduced down the choices to Python and Matlab because of the format our datasets were in (Matlab files). In the end, we chose to go with Python, using IPython to easily visualize and organize our exploration process in a notebook. Numpy and Bottleneck helped speed up calculations in Python, and we used a variety of scientific and machine learning libraries in Python to explore the data, such as scipy.

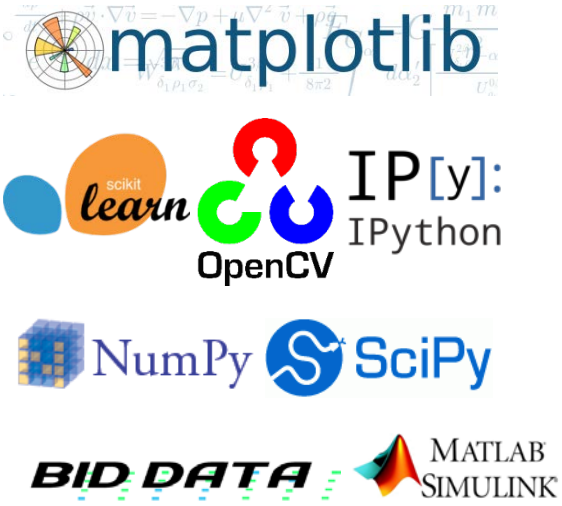


FIGURE 16. THE VARIOUS TOOLS USED THROUGHOUT THE PROJECT.

4.2. DATA WRANGLING

All of the data wrangling was done in Python, which usually only involved applying various filters and feature extraction techniques on the images. We relied on packages like skimage, OpenCV, and Mahotas to explore various ways of featurizing our data. One other package we considered was py-LogGabor, but the documentation was lacking so we decided against it.

4.3. LEARNING FROM DATA

Here, we had to choose between using Python or Scala, scikit-learn or BIDMach. Python was enticing because of its familiarity, and since all our data exploration and wrangling work was done in Python. Scala with BIDMach had an obvious performance advantage, with more than a 3x speedup when we utilized it for SVMs. We tried both, but in the end, we stuck with scikit-learn due to some technical problems with transferring and reading in data from IPython and BIDMach. Additionally, our unfamiliarity with Scala made it difficult to manipulate data with BIDMach.

4.4. DATA VISUALIZATION, RESULTS

For visualizing the data, we relied heavily on matplotlib for charts, images, and graphs. All of

the accuracies and images shown in the paper and used to investigate the data were done through matplotlib.

5. LESSONS, CONCLUDING THOUGHTS

Having domain expertise is usually necessary for successful machine learning on complicated topics. This is due to the difficult feat of feature engineering and extraction – without the domain expertise, we would mostly be drawing up features at random, most of which are not correlated to the problem at hand. Using the original pixels of the images as features was extremely bad, both accuracy-wise and performance-wise. Training on the 500 by 500 images would have taken over 44 hours on scikit-learn for 25,000 linear regression models.

Machine learning also takes a great deal of resources to execute well, in both computing power and space as well as time. With the limited EC2 hours and resources we had, we could not explore all of the voxels for the regression models and try more techniques. With more time, there were other parameters we could tune for the classifiers, as well as combinations of classifiers to look into. Better feature extraction could also have been performed. Overall, regularizing models and preventing overfitting is haard.

6. ROADMAP

A good roadmap for continuing research on this dataset would include:

1. Feature extraction using Log-Gabor filters, which are much better tuned for natural objects.
2. Attempting various non-linear kernels, i.e. polynomial or sigmoid, on the regression and classification models other than the RBF kernel.
3. Training an image classifier (such as Caffe) to gain labels for the grayscale images so that K-Means clustering could be more meaningful during manual evaluation.

7. RELATED WORKS/LINKS

[1] Daugman. (1980), "Two-dimensional spectral analysis of cortical receptive field profiles", *Vision Res.* **20** (10): 847–56, http://www.sciencedirect.com/science/article/pii/0042698980900656

[2] “Gabor filter extraction”, StackOverflow. http://stackoverflow.com/questions/20608458/gabor-feature-extraction

[3] Wang, Hutchinson, Mitchell (2003). “Training fMRI Classifiers to Detect Cognitive States across Multiple Human Subjects”, *NIPS03*. http://www.cs.cmu.edu/~tom/nips03-submitted.pdf

[4] Do, Yang (2014). “A Robust Feature Selection Method for Classification of Cognitive States with fMRI Data”, *Advances in Computer Science and Its Applications*. http://link.springer.com/chapter/10.1007%2F978-3-642-41674-3_11

[5] Pereira, Mitchell, Botvinick (2008). “Machine Learning Classifiers and fMRI: A Tutorial Overview.” https://www.princeton.edu/~fpereira/Presentations/ipam2008.pdf

[6] Mahmoudi, Takerkart, et al. “Multivoxel Pattern Analysis for fMRI Data: A Review.” http://www.hindawi.com/journals/cmmm/2012/961257/