Project - Visual Object Recognition

EEE482 - Computational Neuroscience

Hygerta Imeri - 21603212

Utku Gorkem Erturk - 21502497

Efe Acer - 21602217

Bilkent University, CS

Contents

Abstract

Neuroscience is an emerging field that faces the dilemma whether it is possible or not to decode human brain activity to different tasks it performs. One important tool that can be used while working with fMRI datasets is Multi-Voxel Pattern Analysis (MVPA) which helps in making the data denser by lowering the dimensionality and increasing the model accuracy. Through this paper we intend to show how Visual Object Recognition and decoding can be performed using various classification methods (KNN, SVC, DecisionTree, GaussianNB, LDA, MLP and Logistic Regression) and how the best method is chosen based on the classifier's predictive accuracy. In a comparison of several of the existing analysis we came up with a comprehensive methodology that includes MultiDimensional Scaling (MDS) Analysis, Dimensionality Reduction (PCA and Masking) and Classification steps. MDS is offered as a way to investigate data separability, Dimensionality Reduction is proposed as a way to boost classifier accuracy and decrease runtime. The final step, Classification, is used to decode fMRI data back to the actual stimulus categories. Among the various methods we considered, Masking followed by MLP Classification provided the best predictive performance. MLP is a neural networks approach that explores all probable connections between different predictor variables. Results, therefore, show that there is one optimal data pipeline. The pipeline is: Masking followed by Multi-Layer Perceptron (MLP). This demonstrates that what we proposed as our goal was achieved.

1 Introduction

The emerging complex and multidisciplinary field of neuroscience is constantly running into challenges which are closely related to its primary goal; undertaking and guaranteeing mappings between brain activity images and different tasks performed by the human brain. This being said, challenges arise in the way the analysis is tackled and in the methods used to come up with answers to the proposed questions. Among the analysis methods that use machine learning tools on fMRI datasets, MVPA (Multi-Voxel Pattern Analysis) is being considered [1]. This method allows the analysis of *visual object patterns* which turns out to be one of the most compelling matters of the field. The topic is significant in the way it enables the explanation of 'how the human brain handles and accumulates a visual stimulus'. Moreover, it might be crucial to some other developments in medicine or visual arts (Graphic Design, UIs etc.).

Neural Decoding is one of the most prominent challenges in Computational Neuroscience. Considering that Neural Decoding refers to the process of reconstructing the sensory stimuli based on the information that has already been encoded in the human brain. This method can also be utilized in decoding the category of visual stimuli viewed by a human subject based on their recorded fMRI brain activity.

The key to performing methods such as neural decoding is working with a dense low-dimensional dataset in a high-performance manner. The most common way to solve problems related to brain data is through binary classification, yet it is hard to do so when one is working with high dimensional brain datasets[2]. The classifiers then would be expected to have poor performance.

To come up with solutions to the problems mentioned above this paper proposes testing some of the already existing dimensionality reduction and classification methods. The purpose of this work is to show how Visual Object Recognition is done and how decoding can be performed using various classification methods. Different models are trained to perform the decoding of the brain activity to the stimulus that it was triggered by. After comparing the classifiers' accuracy values, we select the one with the best predictive accuracy as the decoder.

By the end of this work, we expect to have successfully come up with an as-accurate-as-possible analysis method to decode brain responses of visual stimuli.

The succeeding sections of this paper will be as stated below:

Section 2 introduces the methods used, Section 3 gives the results, Section 4 presents a discussion followed up by Section 5 and 6 which correspond to References and Appendix respectively.

2 Methods

2.1 Haxby Dataset

In this project, we examined the Haxby dataset [3]. This dataset[4] contains the experimental fMRI data obtained from a study on the distributed and overlapping representations of faces and objects in human ventral temporal cortex.

6 subjects are involved in the experiment that generated the Haxby dataset. 12 runs are conducted

per subject (9th run for Subject 5 was corrupted); a subject is shown greyscale images of eight object categories. Images from each category are shown to the subjects for a period of 24 seconds and each such period is separated by rest periods of 12 seconds. During the 24 seconds long periods, each image belonging to the corresponding category is presented to the subject for 500 milliseconds followed by a 1500 milliseconds inter-stimulus interval. fMRI data was obtained by recording full brain images with a volume repetition time of 2.5 seconds. Hence, a stimulus block, a block where a single category is shown to the subjects, is approximately covered by 9 fMRI volumes. The image categories included in the experiment are scissors, face, cat, scrambled pixels, bottle, chair, shoe and house.

2.2 Mapping the Volumes to Stimulus Blocks

As it was mentioned in the previous section, the fMRI data points and the stimuli data points given in the Haxby dataset are probed in different time lines. Thus, a mapping was constructed to pair each stimulus block with its corresponding fMRI volumes.

2.3 Dimensionality Reduction

At the final stages of our fMRI analysis, we constructed various classifiers that can decode the fMRI responses back to the input stimulus. To do this, we made use of different machine learning algorithms. A single fMRI volume in the dataset represents a 3 dimensional image of full-brain, which corresponds to 40 × 64 × 64 pixels. This means that each data point has 163840 features (regressors). This number is relatively large so it deteriorates the accuracy and run-time of the machine learning algorithms. For these reasons, PCA and Masking (Region of Interest) are used to overcome the curse of dimensionality.

From a website containing the original dataset of the experiment[3], masking data that was created based on prior biological knowledge on subjects are fetched. Both PCA and Masking is applied to whole response data with whitening. Figure 1 shows the mask of Subject 2 on top of his/her brain anatomy:

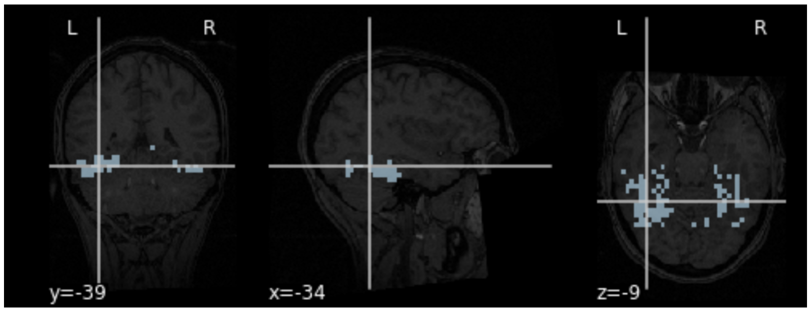


Figure 1: Region of Interest (Mask) for Subject 2 on Top of His/Her Brain Anatomy

To see which reduction method is better, they are compared in terms of classification accuracy using a simple classifier.

2.4 Multidimensional Scaling Analysis

In order to examine whether the eight stimulus categories are discriminable based on the response patterns they elicit across ventral temporal cortex, we computed the response similarity matrices using euclidean, cosine and correlation metrics.

After computing the similarity matrices, we projected the data points into 2 and 3 dimensional space using classical multidimensional scaling algorithm. This way we had the chance to visualize the higher dimensional fMRI response data while preserving the distances (similarities) between the individual data points.

2.5 Classification

Classification stage is the key step to build the decoder that can infer the stimulus category based on the fMRI responses. Since the classification algorithm to use is highly dependent on the specifics of the dataset, we choose to train many classifiers. Here is a list of the classification techniques we examined, together with brief explanations:

- **K-Nearest Neighbors (kNN)**: A non-parametric method that assigns a data point to a class such that the majority of k of its neighbors belong to that class.
- **Support Vector Machine Classifier (SVC)**: A classifier that tries to build a linear boundary which separates class labels by a clear gap (margin) that is as wide as possible. Unseen data points are then assigned to the class label based on which side of the gap they fall.
- **Decision Tree**: A tree structured model where the leaf nodes represent target class labels and branches represent the conjunctions (logical connectives) that lead to the class labels.
- **Gaussian Naive Bayes (GaussianNB)**: A probabilistic method that assigns a data point to the class that maximizes the posterior probability of observing the data point. Makes use of the bayes rule to combine prior assumptions and the maximum likelihood estimations. Assumes that the data associated with each class are Gaussian distributed. In short, performs maximum-a-posteriori (MAP) estimation with a strong assumption of a Gaussian prior.
- **Linear Discriminant Analysis (LDA)**: A classifier that tries to build linear decision boundary on the input space which maximizes the separability among known classes. When the linear decision boundary is being established, the LDA classifier tries to maximize distance between the means of the classes and minimize the variation within each class.
- **Multi-Layer Perceptron (MLP)**: A class of feed-forward artificial neural network. Uses nonlinear activation functions (relu, sigmoid, etc.) in its hidden layer(s). Tunes the weights of its layers by running the backpropagation algorithm. Its output layer performs the classification and maps the final value it receives from the last hidden layer to a class label.
- **Logistic Regression**: A classification technique in which linear regression is applied to the dataset and the values predicted for a data point by the linear model is squeezed into the [0, 1] range using the sigmoid function. The resulting values are interpreted as probabilities and thresholding is applied on the probabilities to choose the appropriate class labels.

All of the methods above work on two classes, meaning we chose binary classification over multi-class classification. This decision is based on the results of the MDS analysis which will be explained in the upcoming sections of our report.

3 Results

3.1 Similarity Matrices

The figure below displays the fMRI response similarity matrices computed using euclidean, cosine and correlation metrics. The similarity matrices are only computed for Subject 2, since other subjects have very similar similarity matrices.

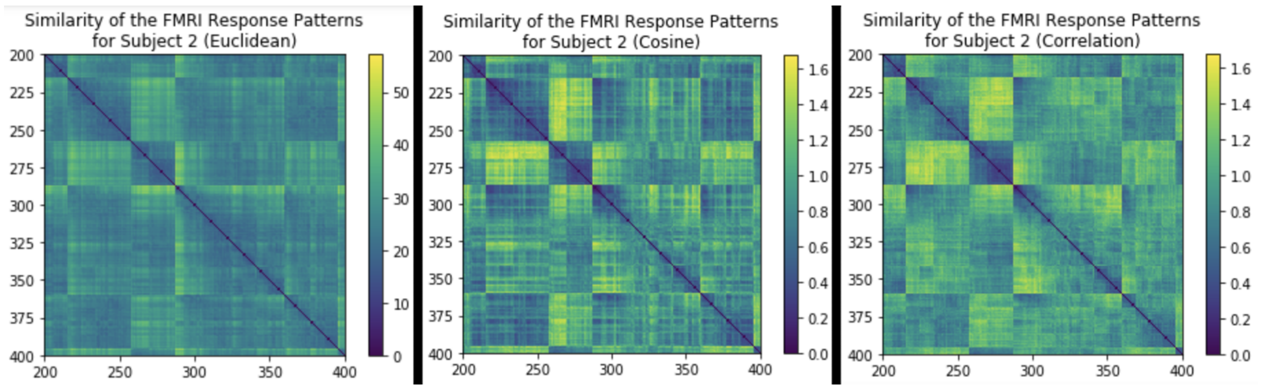


Figure 2: fMRI Response Similarity Matrices for Subject 2

We can see a clear discrimination for the categories in all three similarity matrices. However, the one that uses correlation as its metric provides the best contrast. Hence, we conclude that the correlation metric results in the best discrimination for the categories and proceed with the correlation based similarity matrix in the MDS analysis.

3.2 MDS Results

MDS analysis can give one clues about what type of classification method to use. There are various techniques to perform multidimensional scaling, we choose to implement and use classical multidimensional scaling.

At first, we planned to implement a multi-class decoder that can map an fMRI response to any one of the 8 stimulus categories. We projected the fMRI responses on a 2 dimensional MDS space and labeled the responses according to the actual stimulus category that elicited them. Since the distances between the data points are tried to be preserved by the classical MDS algorithm we expected to see 8 clusters representing responses to each stimulus category, when we scattered the data in the 2 dimensional MDS space. However, we did not observed such a case; indeed we identified a big data cloud of mixed labels. This implied that there is no clear distinction between the categorical responses in the 2 dimensional MDS space. Higher dimensions may provide better distinction but they are harder to visualize. The following figure shows the resulting scatter plot:

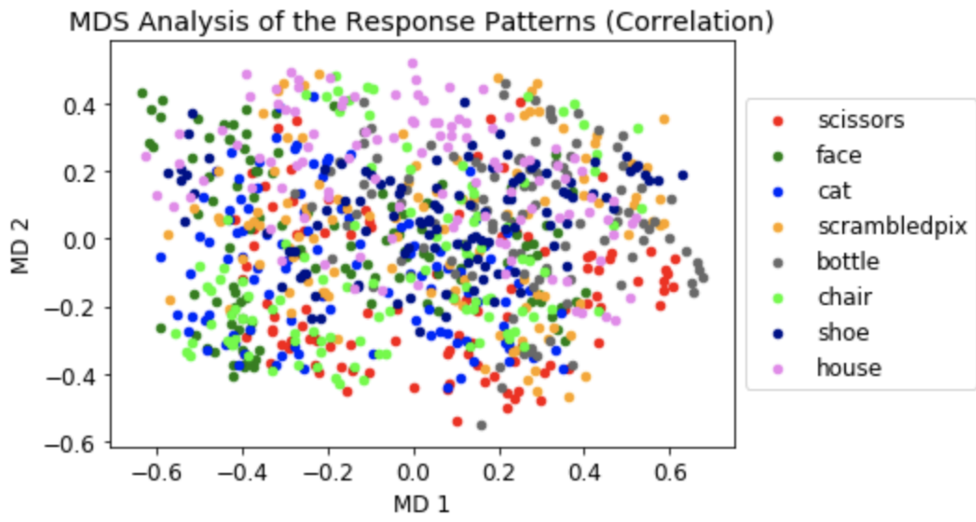


Figure 3: MDS Analysis - Part 1

Considering that a multi-class decoder will have a hard time distinguishing the fMRI responses (at least in the 2D MDS space), we turned our attention to building binary decoders for each category. In order to see whether a binary decoder will perform well or poor, we arbitrarily chose a category, house, and labeled the projected data as house or non-house. Figure 4 displays the resulting scatter plot:

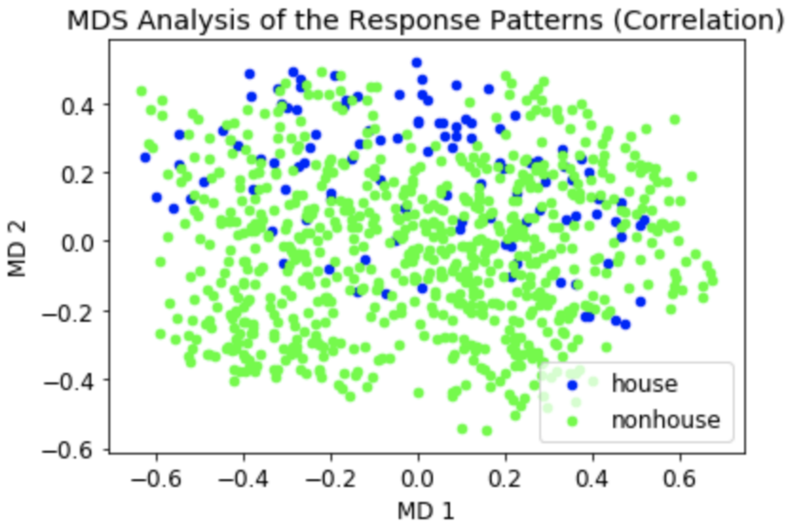


Figure 4: MDS Analysis - Part 2

By inspection on figure 4, we can say that the responses to house stimuli tend to be located above the responses to nonhouse stimuli. There are still no exact clusters but we can see a bit of separation between the data having different labels. This separation is expected to be more visible in a higher dimensional MDS space. Since the best that the human eye can visually interpret is 3, we show the same data in 3 dimensional MDS space. Figure 5 given below presents the resulting scatter plot:

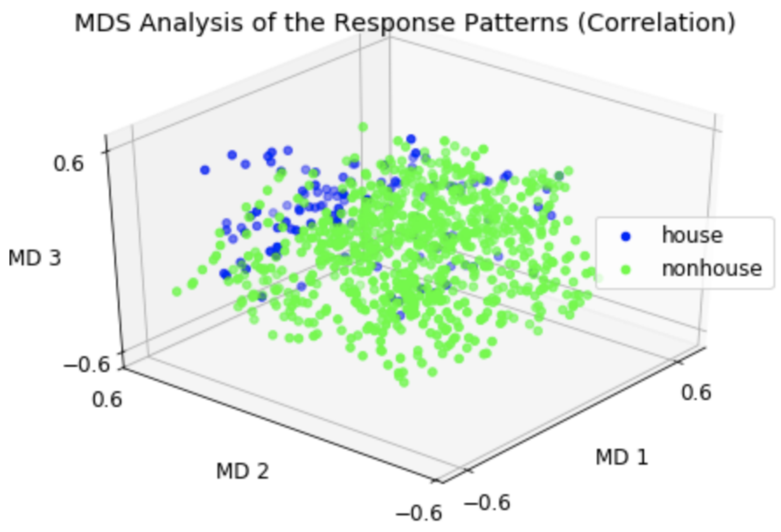


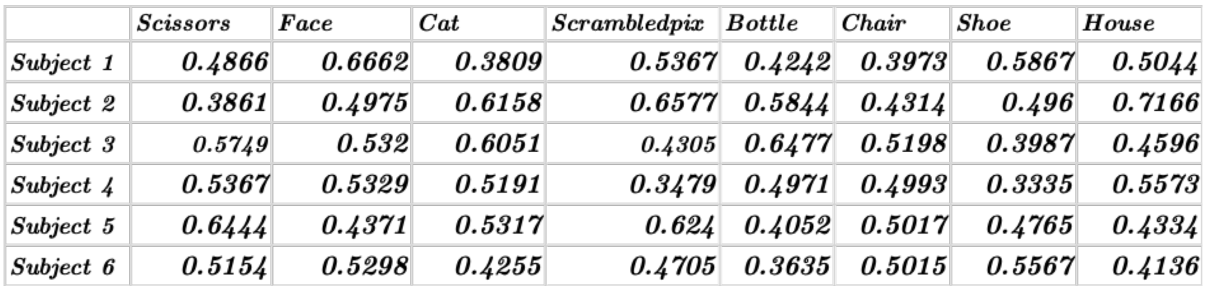
Figure 5: MDS Analysis - Part 3

Now we can see that there exists two separate data clouds for the labels, hence there are certainly two clusters. This implies a binary decoder for each label will probably give high classification accuracies. Thus, we decided to train various binary classifiers for each stimulus category for the proceeding phases of our analysis.

3.3 PCA vs Mask

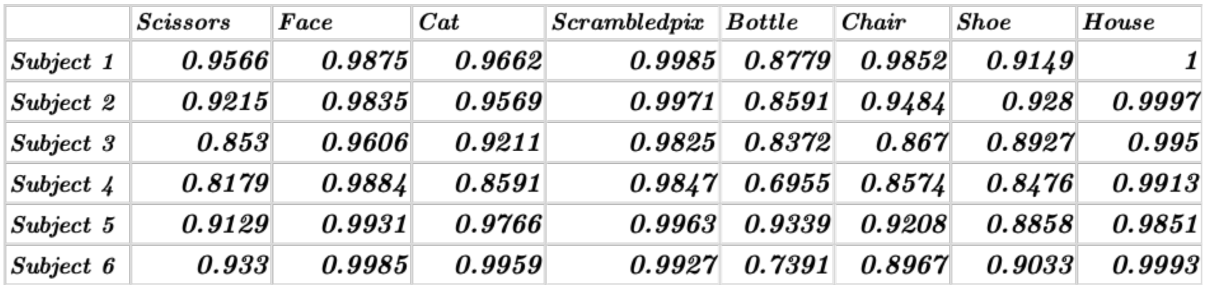
To inspect and classify the recorded brain activity, dimension of the data should be reduced. In order to reduce the dimension of the data, PCA and masking methods are used and they are compared with each other. The comparison is based on the classification accuracy of an arbitrarily chosen classifier which is selected to be Support Vector Machine Classifier (SVC). The raw data is fed into the SVC after its dimension is reduced using PCA or Masking.

Figure 6 shows the SVC accuracies when PCA-reduced data is fed into the classifier. The accuricies are shown for each label, since binary classification is used, and for each subject. Figure 7 shows the same but this time for Masking-reduced data. The data from the tables in figures 6 and 7 are taken and their means are calculated across subjects. The means are than plotted and shown in Figure 8 with error bars showing the standard errors. All of these figures are located on the next page.



	Scissors	Face	Cat	Scrambledpix	Bottle	Chair	Shoe	House
Subject 1	0.4866	0.6662	0.3809	0.5367	0.4242	0.3973	0.5867	0.5044
Subject 2	0.3861	0.4975	0.6158	0.6577	0.5844	0.4314	0.496	0.7166
Subject 3	0.5749	0.532	0.6051	0.4305	0.6477	0.5198	0.3987	0.4596
Subject 4	0.5367	0.5329	0.5191	0.3479	0.4971	0.4993	0.3335	0.5573
Subject 5	0.6444	0.4371	0.5317	0.624	0.4052	0.5017	0.4765	0.4334
Subject 6	0.5154	0.5298	0.4255	0.4705	0.3635	0.5015	0.5567	0.4136

Figure 6: SVC Classification Accuracies for PCA-reduced Data



	Scissors	Face	Cat	Scrambledpix	Bottle	Chair	Shoe	House
Subject 1	0.9566	0.9875	0.9662	0.9985	0.8779	0.9852	0.9149	1
Subject 2	0.9215	0.9835	0.9569	0.9971	0.8591	0.9484	0.928	0.9997
Subject 3	0.853	0.9606	0.9211	0.9825	0.8372	0.867	0.8927	0.995
Subject 4	0.8179	0.9884	0.8591	0.9847	0.6955	0.8574	0.8476	0.9913
Subject 5	0.9129	0.9931	0.9766	0.9963	0.9339	0.9208	0.8858	0.9851
Subject 6	0.933	0.9985	0.9959	0.9927	0.7391	0.8967	0.9033	0.9993

Figure 7: SVC classification Accuracies for Mask-reduced Data

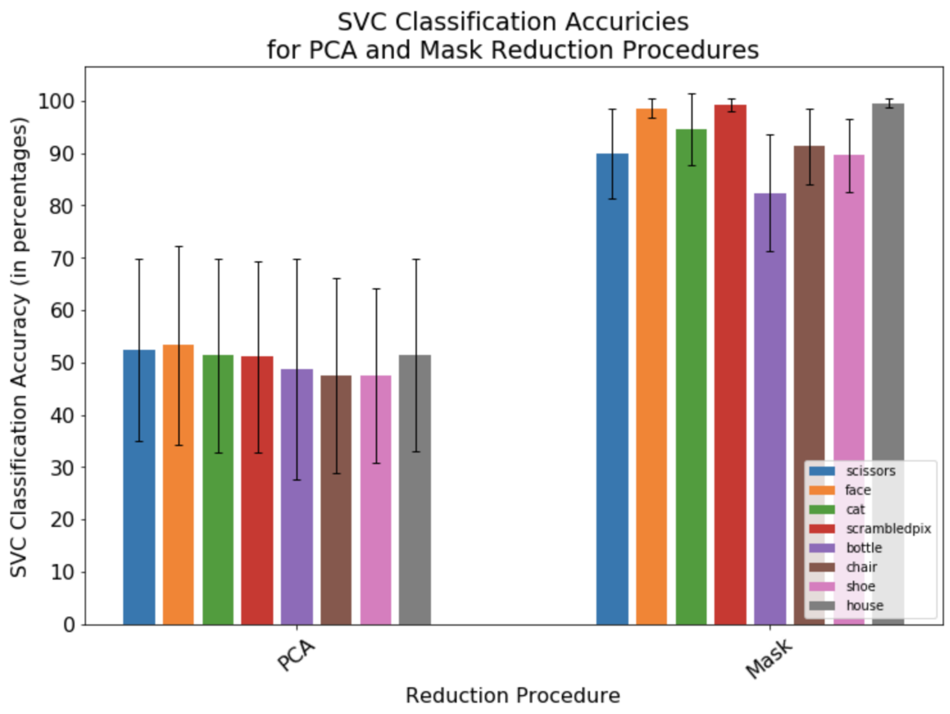


Figure 8: PCA vs Mask SVC Accuracies

SVC with PCA gives approximately 50% classification accuracy, whereas SVC with masking gives around 90% accuracy.

PCA refers to the well known Principal Component Analysis technique, where eigenvalue decomposition is applied to the covariance matrix of the data after a whitening procedure. PCA then selects the k most significant eigenpairs, which are the k largest eigenvalues and their corresponding eigenvectors. After this selection, PCA projects the data onto the space of the selected eigenpairs to obtain a lower dimensional data that still explains the variance structure well enough. The eigenvectors sorted with respect to the descending order of eigenvalues are said to be the Principal Components (PCs).

Masking is a whole different story than PCA. A mask is a transformation that filters the fMRI volumes in a way that the resulting brain images only show the parts of the brain that are the most active throughout the experiment. The masks are specific to the subjects because of the anatomical and biological differences among the subjects' brains. The masks are obtained with the help of domain specific knowledge.

Considering the brief explanations above, it is natural and expected for masking to yield better results than PCA. PCA tries its best to reduce the dimension while preserving the characteristics of the data, however it does not have any information about the dimensions which are the most significant throughout the experiment. As a result of the comparison, masking is chosen as the dimensionality reduction method and the raw data is given to the classifiers after their dimension were reduced using the masks.

3.4 Running Different Classification Algorithms

After dimesion of the dataset is reduced using the masks, the classification algorithms listed in section 2.5 are tested using leave one out cross validation. The accuracy of each binary classifier that is trained to recognize responses to a certain type of stimuli is obtained for each classifier type. The accuracy values are obtained by computing an average of the individual accuracies of all subjects. Standard deviation of the accuracies is computed in a similar manner and the results are displayed in figure 9 shown on the next page.

Figure 9 indicates that the best performing models are MLP, Logistic Regression and SVC. Since SVC performs pretty well on decoding individual stimulus category, we can infer that the data is linearly separable in a higher dimensional space. Logistic Regression is a transform added on top of linear regression that maps the outputs to valid probability values. As Logistic Regression performs well, we can infer that linear regression also performs well to describe the relation between the regressors and outputs. Thus, we can say that there is probably a high order linear relationship between the fMRI responses and the stimuli categories. MLP is in fact the best performing method of all. It performs slightly better than logistic regression and this is an expected result since MLP is some kind of an extension to Logistic Regression. MLP incorporates a neural networks approach to automatically perform feature engineering. In other words, hidden layers in the network play with the features (take weighted linear combinations and apply nonlinearity by means of an activation function) and obtain a better feature set. As a final step, the output layer of the network performs the exact same classification step that is performed by Logistic Regression (a sigmoid activation followed by thresholding).

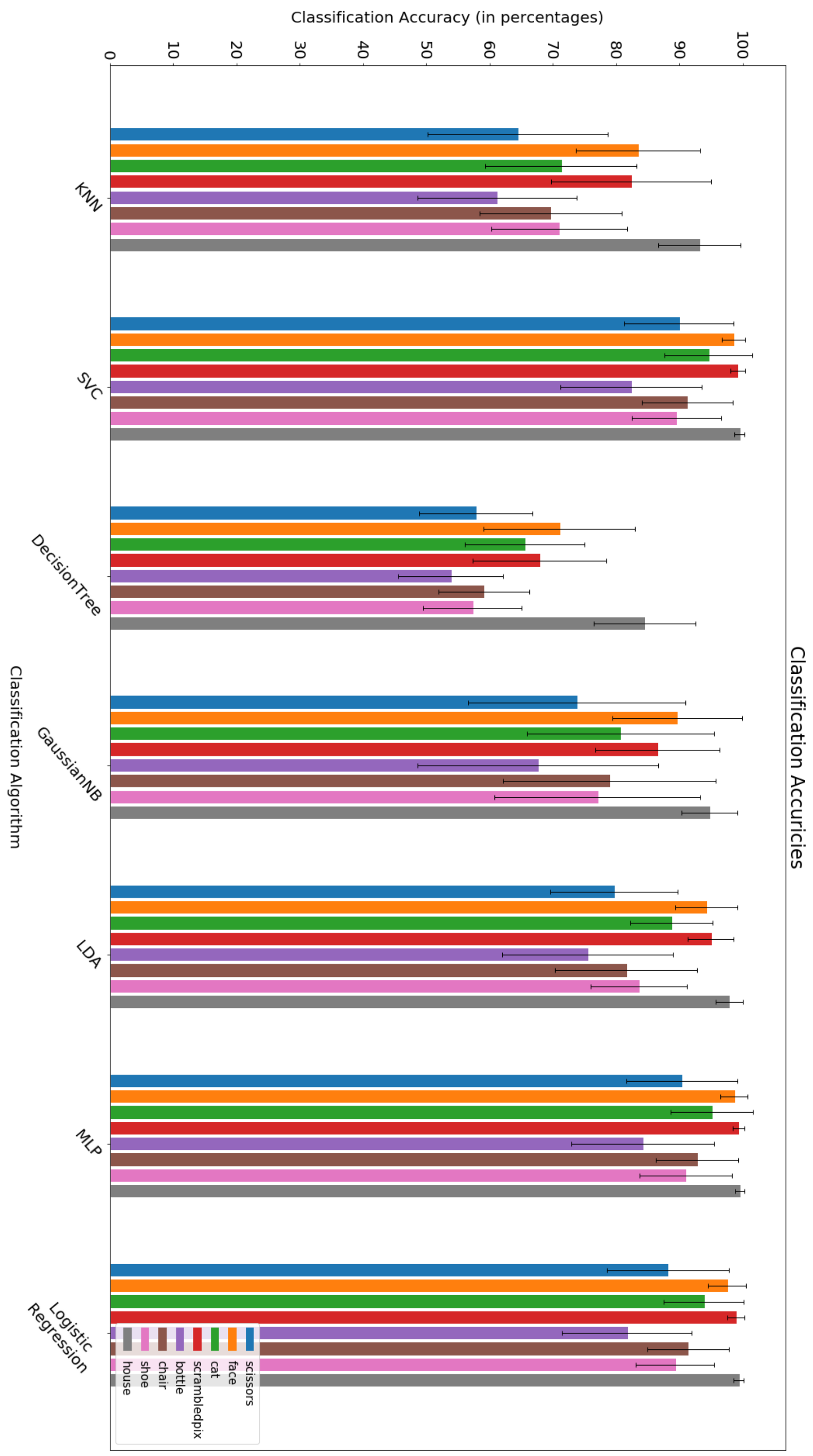


Figure 9: Classifier Accuracies

4 Discussion

Taking into consideration the research that has been done on the same topic [2] [1], we can conclude that we are able to select one optimal classification method among the aforementioned classification methods, namely: KNN, SVC, DecisionTree, GaussianNB, LDA, MLP and Logistic Regression to decode ventral temporal cortex activity. The method that results with the highest accuracy, as mentioned in the previous section is Multi-Layer Perceptron (MLP). MLP does increase the accuracy performance by making use of neural networks. The greatest advantage of using such a machine learning algorithm to implement classification is that it can explore many different combinations (linear or non-linear) of the regressors and come up with a new feature space.

We did not run any new analysis on the current given data (Haxby dataset), nonetheless we compared several of the existing analysis to come up with a comprehensive methodology. We did apply advanced analyses such as MultiDimensional Scaling (MDS) to search for separation between the data points. Among all techniques we have used, only PCA failed drastically in terms of classification accuracy. The rest of the techniques worked successfully.

We believe that the results could be improved by using advanced Deep Learning algorithms. The reason behind our belief is that these methods imitate the human brain, in other words the closer the model gets to the brain the better it stimulates it.

We did not perform any hyper parameter tuning other than the ones performed by default in the SciKit Learn Library. We could have done some additional tuning but it would have minor contributions in the overall classification accuracies.

Overall, the goals set out in the introduction were reached and the desired outcomes were produced.

References

[1] D. Z. M. Yousefnezhad, "Anatomical pattern analysis for decoding visual stimuli in human brains," *Cognitive Computation*, vol. 10, p. 2842950, 2017.

[2] J. V. Haxby, "Distributed and overlapping representations of faces and objects in ventral temporal cortex," *Science*, vol. 293, p. 24252430, 2001.

[3] Haxby, "Haxby dataset."

[4] Haxby, "Openneuro haxby dataset."

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] P. D. A. Library, "pandas."

[7] Nilearn, "Nilearn."

Appendix

[5, 6, 7] The code for the project is written in Python using Jupyter Notebook environment. Each block of the notebook is provided starting below.

```python
import numpy as np
import matplotlib.pyplot as plt
import nibabel as nib
import csv
import pandas as pd
from nibabel.testing import data_path
from nilearn import plotting as nplt
from nilearn.input_data import NiftiMasker
from sklearn.model_selection import LeaveOneGroupOut, cross_val_score
from scipy.spatial.distance import cdist # Python equivalent of pdist2
#from sklearn.metrics import accuracy_score, log_loss
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC
from sklearn.tree import DecisionTreeClassifier
#from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
    GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.decomposition import PCA
#from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from mpl_toolkits.mplot3d import Axes3D
```

```python
LABELS = ['scissors', 'face', 'cat', 'scrambledpix', 'bottle', 'chair', 'shoe', 'house']
```

```python
FMRI_STIMULI_LENGTH = 9
fmri_startpoints = [6, 21, 35, 49, 63, 78, 92, 106]
fmri_indices = []
for fmri_startpoint in fmri_startpoints:
    for offset in range(FMRI_STIMULI_LENGTH):
        fmri_indices.append(fmri_startpoint + offset)
fmri_indices = np.array(fmri_indices)
print('FMRI timpoints corresponding to stimuli:\n', fmri_indices)
print('Number of FMRI timepoints corresponding to stimuli:', fmri_indices.size)
```

```python
NUM_SUBJECTS = 6
NUM_FUNC = 12
STIMULI_LENGTH = 12
NUM_STIMULI = 8
```

```python
%%time
print('Takes a bit of time...')
sub_mask = []
```

```python
sub_anat = []
sub_func = []
sub_func_PCA = []
sub_label = []
flatten = lambda l: [item for sublist in l for item in sublist]
for i in range(1, NUM_SUBJECTS + 1):
    anat_path_str = 'data/sub-' + str(i) + '/anat/sub-' + str(i) + '_T1w.nii.gz'
    anat_data = nib.load(anat_path_str)
    sub_anat.append(anat_data)
    temp_arr = []
    temp_arr_ = []
    temp_arr_PCA = []
    mask_path_str = 'haxby/subj' + str(i) + '/mask4_vt.nii.gz'
    sub_mask.append(nib.load(mask_path_str))
    masker = NiftiMasker(mask_img=mask_path_str,
                         smoothing_fwhm=4,
                         standardize=True,
                         memory='nilearn_cache',
                         memory_level=1)
    pca = PCA(whiten=True)
    for j in range(1, NUM_FUNC + 1):
        if i == 5 and j == 12: # 5th subject has 11 func data
            break
        func_str = '0'
        if j < 10:
            func_str += str(j)
        else:
            func_str = str(j)
        func_path_str = 'data/sub-' + str(i) + '/func/sub-' + str(i) + '_task-
            objectviewing_run-' + func_str + '_bold.nii.gz'
        func_data = nib.load(func_path_str)
        temp_arr.append(pd.DataFrame(masker.fit_transform(func_data).T))
        func_data_PCA_ = np.array(func_data.get_fdata())
        func_data_PCA = func_data_PCA_.reshape(40 * 64 * 64, 121)
        temp_arr_PCA.append(pd.DataFrame(pca.fit_transform(func_data_PCA).T))
        label_path_str = 'data/sub-' + str(i) + '/func/sub-' + str(i) + '_task-
            objectviewing_run-' + func_str + '_events.tsv'
        label_data = pd.read_csv(label_path_str, delimiter='\t')
        filtered_labels = []
        for k in range(NUM_STIMULI):
            filtered_labels.append([label_data['trial_type'][k * STIMULI_LENGTH]] *
                FMRI_STIMULI_LENGTH)
        temp_arr_.append(flatten(filtered_labels))
    sub_func.append(temp_arr)
    sub_func_PCA.append(temp_arr_PCA)
    sub_label.append(temp_arr_)
sub_anat = np.array(sub_anat)
sub_func = np.array(sub_func)
sub_func_PCA = np.array(sub_func_PCA)
sub_label = np.array(sub_label)
```

```python

for i in range(NUM_SUBJECTS):
    for j in range(NUM_FUNC):
        if i == 4 and j == 11:
            break
        sub_func[i][j] = sub_func[i][j][fmri_indices]
        sub_func_PCA[i][j] = sub_func_PCA[i][j][fmri_indices]
```

```python
def stack_data(data, axis_=1):
    """
    Stacks the data together row wise or column wise.
    Args:
        data: Input data
        axis_: Axis to stack (default is 1 (column wise))
    Returns:
        stacked: The stacked data
    """
    stacked = data[0]
    for i in range(1, len(data)):
        stacked = np.append(stacked, data[i], axis=axis_)
    return stacked.T
```

```python
data = []
data_PCA = []
labels = []
for i in range(NUM_SUBJECTS):
    data.append(stack_data(sub_func[i]))
    data_PCA.append(stack_data(sub_func_PCA[i]))
    labels.append(stack_data(sub_label[i], axis_=0))
```

```python
nplt.plot_roi(sub_mask[1], bg_img=sub_anat[1], cmap='Paired')
print('Mask on Top of Brain Anatomy (Subject 2)')
plt.show(block=False)
```

```python
# Compute similarity matrices using different metrics
sim_euclidean = cdist(data[1], data[1], metric='euclidean')
sim_cosine = cdist(data[1], data[1], metric='cosine')
sim_correlation = cdist(data[1], data[1], metric='correlation')
```

```python
figure_num = 1
plt.figure(figure_num); figure_num += 1
plt.imshow(sim_euclidean); plt.colorbar()
plt.xlim(200, 400); plt.ylim(400, 200)
plt.title('Similarity of the FMRI Response Patterns\nfor Subject 2 (Euclidean)')

plt.figure(figure_num); figure_num += 1
plt.imshow(sim_cosine); plt.colorbar()
plt.xlim(200, 400); plt.ylim(400, 200)
plt.title('Similarity of the FMRI Response Patterns\nfor Subject 2 (Cosine)')

plt.figure(figure_num); figure_num += 1
plt.imshow(sim_correlation); plt.colorbar()
plt.xlim(200, 400); plt.ylim(400, 200)
plt.title('Similarity of the FMRI Response Patterns\nfor Subject 2 (Correlation)')
plt.show(block=False)
```

```
def cmdscale(D):
    """
    Implementation of the classical multidimensional scaling (MDS) algorithm.
    Args:
        D: The symmetric matrix containing the distances between n
            objects in p dimensions
    Returns:
        X: Coordinates of n objects in the new space
    """
    N = D.shape[0]
    # Double centering procedure
    J = np.eye(N) - np.ones((N, N)) / N
    B = - J.dot(D ** 2).dot(J) / 2 # = X.X^T
    # Diagonalization
    evals, evecs = np.linalg.eigh(B)
    # Sort eigenpairs according to the descending order of eigenvalues
    idx = np.argsort(evals)[::-1]
    evals = evals[idx]
    evecs = evecs[:, idx]
    # Extract the positive eigenvalues
    pos_idx = np.where(evals > 0)[0]
    L = np.diag(np.sqrt(evals[pos_idx]))
    E = evecs[:, pos_idx]
    X = E.dot(L)
    return X
```

```
# Run classical MDS on correlation based similarity matrix
MDS_correlation = cmdscale(sim_correlation)[:, 0:3]
```

```
categories = sub_label[1]
categories = flatten(categories)

SCISSOR_INDICES = np.array([i for i in range(len(categories)) if categories[i] == '
    scissors'])
FACE_INDICES = np.array([i for i in range(len(categories)) if categories[i] == 'face'])
CAT_INDICES = np.array([i for i in range(len(categories)) if categories[i] == 'cat'])
SCRAMBLEDPIX_INDICES = np.array([i for i in range(len(categories)) if categories[i] == '
    scrambledpix'])
BOTTLE_INDICES = np.array([i for i in range(len(categories)) if categories[i] == 'bottle'
    ])
CHAIR_INDICES = np.array([i for i in range(len(categories)) if categories[i] == 'chair'])
SHOE_INDICES = np.array([i for i in range(len(categories)) if categories[i] == 'shoe'])
HOUSE_INDICES = np.array([i for i in range(len(categories)) if categories[i] == 'house'])
```

```python
plt.figure(figure_num); figure_num += 1
colors_ = ['r', 'g', 'b', 'orange', 'dimgray', 'lime', 'darkblue', 'violet']
plt.scatter(MDS_correlation[SCISSOR_INDICES, 0], MDS_correlation[SCISSOR_INDICES, 1], s=20, marker='o', c=colors_[0])
plt.scatter(MDS_correlation[FACE_INDICES, 0], MDS_correlation[FACE_INDICES, 1], s=20, marker='o', c=colors_[1])
plt.scatter(MDS_correlation[CAT_INDICES, 0], MDS_correlation[CAT_INDICES, 1], s=20, marker='o', c=colors_[2])
plt.scatter(MDS_correlation[SCRAMBLEDPIX_INDICES, 0], MDS_correlation[SCRAMBLEDPIX_INDICES, 1], s=20, marker='o', c=colors_[3])
plt.scatter(MDS_correlation[BOTTLE_INDICES, 0], MDS_correlation[BOTTLE_INDICES, 1], s=20, marker='o', c=colors_[4])
plt.scatter(MDS_correlation[CHAIR_INDICES, 0], MDS_correlation[CHAIR_INDICES, 1], s=20, marker='o', c=colors_[5])
plt.scatter(MDS_correlation[SHOE_INDICES, 0], MDS_correlation[SHOE_INDICES, 1], s=20, marker='o', c=colors_[6])
plt.scatter(MDS_correlation[HOUSE_INDICES, 0], MDS_correlation[HOUSE_INDICES, 1], s=20, marker='o', c=colors_[7])
plt.xlabel('MD 1')
plt.ylabel('MD 2')
plt.title('MDS Analysis of the Response Patterns (Correlation)')
plt.legend(LABELS, loc='center left', bbox_to_anchor=(1, 0.5))
plt.show(block=False)
```

```python
NONHOUSE_INDICES = np.array([i for i in range(len(categories)) if categories[i] != 'house'])
```

```python
plt.figure(figure_num); figure_num += 1
plt.scatter(MDS_correlation[HOUSE_INDICES, 0], MDS_correlation[HOUSE_INDICES, 1], s=20, marker='o', c='b')
plt.scatter(MDS_correlation[NONHOUSE_INDICES, 0], MDS_correlation[NONHOUSE_INDICES, 1], s=20, marker='o', c='lime')
plt.xlabel('MD 1')
plt.ylabel('MD 2')
plt.title('MDS Analysis of the Response Patterns (Correlation)')
plt.legend(['house', 'nonhouse'])
plt.show(block=False)
```

```python
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(MDS_correlation[HOUSE_INDICES, 0],
          MDS_correlation[HOUSE_INDICES, 1],
          MDS_correlation[HOUSE_INDICES, 2], color='b')
ax.scatter(MDS_correlation[NONHOUSE_INDICES, 0],
          MDS_correlation[NONHOUSE_INDICES, 1],
          MDS_correlation[NONHOUSE_INDICES, 2], color='lime')
ax.view_init(elev=35, azim=220)
plt.title('MDS Analysis of the Response Patterns (Correlation)')
ax.set_xlabel('MD 1'); ax.set_ylabel('MD 2'); ax.set_zlabel('MD 3')
ax.set_xticks([-0.6, 0.6]); ax.set_yticks([-0.6, 0.6]); ax.set_zticks([-0.6, 0.6])
```

```
plt.legend(['house', 'nonhouse'], loc='center right')
plt.show(block=False)
```

```
svc = SVC(C=1., kernel='linear')
cv = LeaveOneGroupOut()

cv = LeaveOneGroupOut()

GROUPS = np.repeat([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11], 72)
GROUPS_FOR_SUB5 = np.repeat([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 72)

scores_PCA = dict()
scores_Mask = dict()

for i in range(NUM_SUBJECTS):
    print('\n---SUBJECT %d---' % (i + 1))
    scores_PCA[i] = dict()
    scores_Mask[i] = dict()
    groups_ = GROUPS if i != 4 else GROUPS_FOR_SUB5
    for label in LABELS:
        print('Classification of %s' % label)
        classification_target = (labels[i] == label)
        scores_PCA[i][label] = cross_val_score(svc,
                                               data_PCA[i],
                                               classification_target,
                                               cv=cv,
                                               groups=groups_,
                                               scoring='roc_auc')
        scores_Mask[i][label] = cross_val_score(svc,
                                                data[i],
                                                classification_target,
                                                cv=cv,
                                                groups=groups_,
                                                scoring='roc_auc')
        print('Scores for PCA: %1.4f +- % 1.4f' % (scores_PCA[i][label].mean(), scores_PCA
            [i][label].std()))
        print('Scores for Mask: %1.4f +- % 1.4f' % (scores_Mask[i][label].mean(),
            scores_Mask[i][label].std()))
```

```
reduction_names = ['PCA', 'Mask']
score_means_r = dict()
score_stds_r = dict()
for r_name in reduction_names:
    score_means_r[r_name] = dict()
    score_stds_r[r_name] = dict()
    for label in LABELS:
        sum_mean = 0
        sum_std = 0
        for i in range(NUM_SUBJECTS):
            if r_name == 'PCA':
                sum_mean += scores_PCA[i][label].mean()
                sum_std += scores_PCA[i][label].std()
            elif r_name == 'Mask':
                sum_mean += scores_Mask[i][label].mean()
                sum_std += scores_Mask[i][label].std()
        score_means_r[r_name][label] = sum_mean / NUM_SUBJECTS
        score_stds_r[r_name][label] = sum_std / NUM_SUBJECTS
```

```
reduction_names = ['PCA', 'Mask']
plt.figure(figure_num, figsize=(12, 8)); figure_num += 1
plt.rcParams.update({'font.size': 16})
plt.title('SVC Classification Accuricies\nfor PCA and Mask Reduction Procedures')
plt.xlabel('Reduction Procedure')
plt.ylabel('SVC Classification Accuracy (in percentages)')
plt.xticks([i * NUM_STIMULI * 1.5 + 4 for i in range(len(reduction_names))],
           reduction_names,
           rotation=40)
plt.yticks([i / 10 for i in range(11)], [i * 10 for i in range(11)])
for i, label in enumerate(LABELS):
    plt.bar(x=[j * NUM_STIMULI * 1.5 + i for j in range(len(reduction_names))],
            height=[score_means_r[r_name][label] for r_name in reduction_names],
            yerr=[score_stds_r[r_name][label] for r_name in reduction_names],
            error_kw=dict(lw=1, capsize=3, capthick=1))
plt.rcParams.update({'font.size': 10})
plt.legend(LABELS, loc='lower right')
plt.show(block=False)
```

```
corr_matrix = np.corrcoef(data[1])
print('For subject 2\'s FMRI responses:')
print('Determinant of the correlation matrix:', np.linalg.det(corr_matrix))
print('Condition number of the correlation matrix:', np.linalg.cond(corr_matrix))
```

```
%%time
print('Takes a bit of time...')

classifiers = [
    KNeighborsClassifier(2),
    SVC(C=1., kernel='linear'),
    DecisionTreeClassifier(),
    #RandomForestClassifier(),
    #AdaBoostClassifier(),
    #GradientBoostingClassifier(),
    GaussianNB(),
    LinearDiscriminantAnalysis(),
    #QuadraticDiscriminantAnalysis(), # Variables are collinear
    MLPClassifier(alpha=1, max_iter=1000),
    LogisticRegression(solver='lbfgs')]

classifier_names = ['KNN',
                    'SVC',
                    'DecisionTree',
                    #'RandomForest',
                    #'AdaBoost',
                    #'GradientBoosting',
                    'GaussianNB',
                    'LDA',
                    #'QDA',
                    'MLP',
                    'LogisticRegression']

cv = LeaveOneGroupOut()

scores = dict()

for clf, clf_name in zip(classifiers, classifier_names):
    print('Classification using %s:' % clf_name)
    scores[clf_name] = dict()
    for i in range(NUM_SUBJECTS):
        scores[clf_name][i] = dict()
        groups_ = GROUPS if i != 4 else GROUPS_FOR_SUB5
        for label in LABELS:
            classification_target = (labels[i] == label)
            scores[clf_name][i][label] = cross_val_score(clf,
                                                         data[i],
                                                         classification_target,
                                                         cv=cv,
                                                         groups=groups_,
                                                         scoring='roc_auc')
        print('Subject %d done.' % (i + 1))
```

```
%%time
score_means = dict()
score_stds = dict()
for clf_name in classifier_names:
    score_means[clf_name] = dict()
    score_stds[clf_name] = dict()
    for label in LABELS:
        sum_mean = 0
        sum_std = 0
        for i in range(NUM_SUBJECTS):
            sum_mean += scores[clf_name][i][label].mean()
            sum_std += scores[clf_name][i][label].std()
        score_means[clf_name][label] = sum_mean / NUM_SUBJECTS
        score_stds[clf_name][label] = sum_std / NUM_SUBJECTS
```

```
print('--- SCORES ---\n')
for clf_name in classifier_names:
    print('=== Classifier %s ===\n' % clf_name)
    for label in LABELS:
        print('*** Prediction Accuracy Averaged Over All Subjects for Label %s ***' %
          label)
        print('%1.4f +- %1.4f\n' % (score_means[clf_name][label], score_stds[clf_name][
            label]))
```

```
plt.figure(figure_num, figsize=(32, 16)); figure_num += 1
plt.rcParams.update({'font.size': 20})
plt.title('Classification Accuricies')
plt.xlabel('Classification Algorithm')
plt.ylabel('Classification Accuracy (in percentages)')
plt.xticks([i * NUM_STIMULI * 1.5 + 4 for i in range(len(classifier_names))],
          classifier_names[0:(len(classifier_names) - 1)] + ['Logistic\nRegression'],
          rotation=40)
plt.yticks([i / 10 for i in range(11)], [i * 10 for i in range(11)])
for i, label in enumerate(LABELS):
    plt.bar(x=[j * NUM_STIMULI * 1.5 + i for j in range(len(classifier_names))],
            height=[score_means[clf_name][label] for clf_name in classifier_names],
            yerr=[score_stds[clf_name][label] for clf_name in classifier_names],
            error_kw=dict(lw=1, capsize=3, capthick=1))
plt.rcParams.update({'font.size': 15})
plt.legend(LABELS, loc='lower right')
#plt.savefig('ClassificationResults.png')
plt.show(block=False)
```