Visual Object Recognition Project

M. Ufuk Özdemir *21803596*
E. Ahsen Çakmakci *21802620*
Derin Çayır *21802558*
B. Arda Düzgün *21802633*

***Abstract*—This project uses three different feature selection methods and three different classification methods for solving the problem of visual object recognition. Pipelines investigate FMRI images of human subjects' brain activity corresponding to eight different types of objects (face, cat, house, bottle, shoes, chair, scissors and nonsense pictures). For feature selection of the data, Ventral Temporal Cortex Masking, Analysis of Variance (ANOVA) and Recursive Feature Elimination (RFE) are used. Then, methods Support Vector Classifier (SVM), Logistic Regression (LR) and Linear Discriminant Analysis (LDA) are applied for classifiers. All of the classifiers are identified as successful for classifying FMRI images with proper regularization. Also, SVD feature reduction is used for the LDA classifier. To compare and understand the accuracy of feature selections and classifiers, confusion matrices are computed. These matrices along with accuracy scores of the classifier pipelines showed that a similar prediction accuracy can be achieved using RFE without using brain masks. ANOVA did not work well for feature selection compared to two other methods.**

***Index Terms*—Multi-Voxel Pattern Analysis, FMRI**

I. Introduction

This project focuses on the recorded brain activities of the human brain. The aim is to use multi-voxel pattern analysis to determine which type of stimuli are present. To do so, methods of feature selection and classification are chosen which are given in the proceeding sections.

A. Possible Applications of Multi-Voxel Pattern Analysis

Multi-voxel pattern analysis from FMRI images can be extended to provide communication methods for fully paralyzed individuals. By classifying the visual stimuli of the human brain, those individuals can easily imagine an abstract object which can be tracked. Therefore, this project becomes crucial when it comes to overcoming communication obstacles. If this method can be generalized to understand the thoughts of people, it will be easier to communicate with paralyzed people. Additionally, it is observed that decoding subjects' brain activity becomes useful when identifying whether a player is lying about the card that they are playing with [1]. This concept can be generalized to detect crimes, as well. For instance, during courts, the suspects' brains can be examined to understand the truth.

B. Overview of the Methodology and Similar Work in Literature

The methodology used in this project is summarized in Fig. 1. The images of the brain response to 8 different

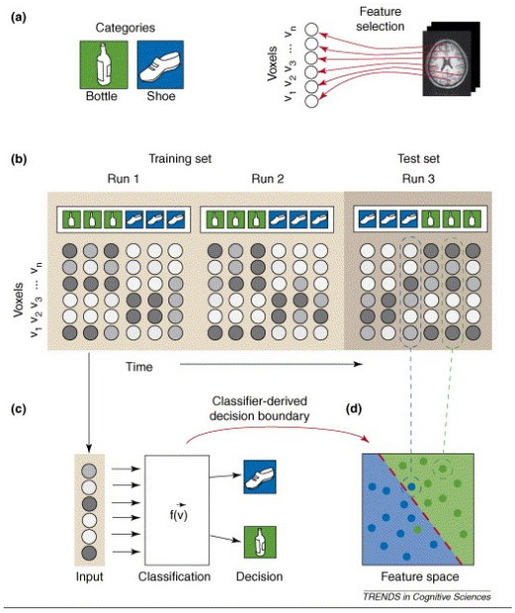

Fig. 1. The overview of the methodology used in this project. [1] First, the experiments are performed and the response of the brain to different images are recorded. The high dimensional FMRI data caanot be used on its own. A feature selection method is used to determine more "informative" voxels in the data. After that, the chosen features along with the category of the image is used to train a classification algorithm. This model can be used to illustrate the effect of different images on brain or can be used to predict the image seen by the subject from new data.

categories of images (face, shoe, cat, chair, bottle, scrambled images, scissors, house) are recorded by Haxby et al. [2]. The responses of the brain can be classified to predict the unseen FMRI results. The time-series data of the FMRI is decomposed into eight groups using the information of different image categories. The first challenge is to perform this classification task to identify the voxels in FMRI images that carry significant information for our task. A typical FMRI output in our dataset consists of $40 \times 64 \times 64 = 163840$ voxels. Only a minor portion of these voxels are likely to carry significant information for classifying the image categories. Therefore, the data dimension must be reduced. Haxby et al. used domain knowledge to select these voxels [2]. They identified the anatomical brain regions which are related to high-level visual processing. The ventral temporal region of the brain is chosen and the "mask" that identifies the related regions is applied to all FMRI measurements to limit the dimension used in the classification. This way only 464 of the initial 163840 voxels are used for classification. So, both

the implementation time of the algorithms are reduced and the widely-known effect of high dimension ("the curse of dimensionality") in classification success is bypassed. However, these masks are unique for each subject since each subject's brain is different. This method reduces the applicability of the method for different people. Also, the principle of the human brain is not fully studied yet. So, if this algorithm is intended to be used on another application for which the responsible brain region is not fully known, this method will not be useful. Therefore, the feature selection part of the pipeline mentioned in Figure 1 is also tried to be achieved by using unsupervised pre-processing methods. These methods are introduced in the methods section and the success of these other methods is compared to the masking method in the results section. After selecting the important features in the data set, the images should be classified using these features. The classification task is achieved using three different methods which are described in the methods section as well. Classification methods are Support Vector Classifier (SVM), Logistic Regression (LR) and Linear Discriminant Analysis (LDA). Also, two different norms (l1 and l2) are used to calculate the penalty in LR and SVM. Using the feature selection methods mentioned above and these classification methods $3 \times 5 = 15$ pipelines are formed and their classification performance is compared in the results section using confusion matrices and accuracy scores. It is expected that some of the proposed methods will be able to predict the category of unseen brain images with a high success rate.

II. Methods

A. Description of the Data Set

The FMRI images collected by Haxby et al. [2] are used in this project. The brain images of six different (five female, one male) subjects are collected through 12 different 5-minute runs. In this project, only one subject (subject 2) is used. If the model is trained in one subject, it cannot be used on another so the analysis will be observing only one subject. Different subjects can be used to reproduce similar results as well. Subject 2 is chosen for this project because its MR images are cleaner for the first evaluation. In each run, the time horizon gets divided into 150 equal parts. One category gets chosen randomly and an image of this category is shown to the subject for 0.5 seconds in 12 of these equal intervals. Since there are eight categories of images, $12 \times 8 = 96$ time intervals are used to show the image to the subject. The remaining 34 time interval in each run is used to capture the resting state of the brain by showing no image. These are labelled as "rest" in the data set. The FMRI outputs are recorded every 2.5 seconds so the timing of the FMRI images is not compatible with the labels. For this reason, the FMRI images are mapped to their labels before starting the implementation of the algorithms below. Also, the images that are labelled as 'rest' are used to predict the resting state response of the subjects. The mean of these images is calculated and subtracted from others to neutralize the effect of the resting state response of the human brain.

B. Feature Selection

Feature selection is an important aspect of MVPA as it is mentioned in the introduction section. The dimension of the output of FMRI is very high hence there are lots of unnecessary variables among the features. Considering the "curse of dimensionality", which is explained in the following section, these features must be reduced to make better predictions and obtain a better model for the system. Masking is a standard way to select more "informative" voxels. However, it does not involve our analysis of the data. The previously found anatomically information-bearing voxels are selected and used. Therefore, if we are required to use classification algorithms in a new topic which is not studied before, unsupervised pre-processing methods should be used. We wanted to implement and compare such methods with Ventral Temporal Masks. For this reason, one univariate and one multivariate feature selection method are chosen and implemented. ANOVA is a univariate method and computes voxel-wise f-scores. After that, the voxels are sorted according to their scores and some with the highest scores are chosen to be used as input to the classifiers. However, this type of feature selection is prone to neglect the voxels which are more informative when considered together. This concern is addressed by multivariate methods but they are computationally expensive to implement due to the combinatorial explosion of the possible selections. The efficient implementation can be achieved by adding one new voxel to the selected set at a time. Our multivariate method is recursive feature elimination (RFE) which can solve the problem of eliminating the collaboratively informative voxels. The details of these methods are explained in the following sections.

a) Curse of Dimensionality: Curse of dimensionality is a term used for the problems that occur when the dataset is high dimensional. The most common problems of high dimensionality are data sparsity and distance concentration. To analyze data better, we need to have all the possible outcomes in the dataset. For high dimensional data, the samples cannot cover all the possibilities and this inability is called data sparsity. Analyzing with data sparsity would cause overfitting. And distance concentration happens when samples' pairwise distances converge to the same value due to the high dimensionality. This problem affects models that are based on calculating the distance such as the method LDA which we have used in this project. So, to prevent the curse of dimensionality, one must use feature selection methods to decrease the dimension of the data. It is observed that non-linear feature selection methods do not overperform linear feature selection methods although they are computationally expensive compared to linear methods [1] [2].

b) Ventral Temporal Cortex Masking: The Ventral Temporal region of the brain can be masked and its voxels can be used as a feature selection method. This region is identified and used by Haxby et al. [2]. To observe the activation of the human brain in response to visual stimuli, the model must focus on the activity on the ventral object vision pathway. Studies about the ventral temporal cortex revealed that different types of objects activate different pathways and differently

localized areas [2]. One of the proposed models that explain the specialization is object topography. This is the model the mask procedure considers. This model predicts that different objects would arouse different and unique response patterns in the ventral temporal cortex and allows acquiring sufficient and overt representations for limitless categories. [2] The ventral temporal region masks for 6 subjects are obtained from Hanke et al.'s work [3].

c) Analysis of Variance (ANOVA): ANOVA method reduces dimension via deciding on which samples are more statistically significant concerning their F values. The F value gives the ratio between explained variance and unexplained variance. It can be calculated as the equation down below;

$$\frac{\frac{RSS_1 - RSS_2}{k_2 - k_1}}{\frac{RSS_2}{n - K_2}} \tag{1}$$

Where n: number of data samples, RSS_1 :sum of squares of residual errors after subset 1, RSS_2 :sum of squares of residual errors after subset 2, k_1 : number of parameters of subset 1, k_2: number of parameters of subset 2 and $k_2 > k_1$. F statistic compares the joint effect of all the samples together. So, it must be considered along with the p-value. ANOVA and t-test are similar to each other since they decide on significance based on the mean comparison between two sample groups. However, unlike the t-test ANOVA method can be used for testing more than three samples. Dimension reduction problem naturally works with more than three samples therefore ANOVA method is a nice way of seeing the significance of feature [4]. Also it another method that does not need domain knowledge so it can be compared to the RFE method fairly.

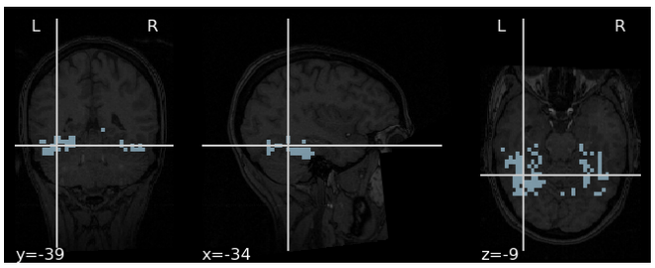

Fig. 2. Example of Ventral Temporal Mask (Subject Two)

d) Recursive Feature Elimination (RFE): When we need to remove numerous features to reduce dimension, the methods that eliminate one feature at a time are not optimal because they ignore the eliminated features' collaborative effect over the data. This problem is solved via RFE because it cares about the feature subset ranking [5]. The recursive feature elimination (RFE) method is an iterative method that uses backwards elimination of the features. This method starts with all of the features and then in each iteration cycle eliminates the subset that gives the least information. The RFE ranking does not necessarily give the nested feature subsets with individually most informative samples. Instead, it gives the nested feature subsets that contain complementary significant samples. This is a computationally expensive method but it is useful since it works better for the correlated samples [6]. The importance score is based on the coefficients of the features in our implementation in this project. Different metrics to evaluate the importance of the features can be chosen or developed for different problem instances.

C. Classification

When choosing the most suitable classifier method, the importance of voxels is considered. It is known that detecting the voxels has a crucial role in detecting the different categories of images. When the classifier is chosen as a linear one, the effect of the voxels on the classifier performance can be tracked easily. The linear models can be inverted easily so it is easier to detect the voxels which are important in the classification of images on the brain. By illustrating them on brain MR images we can gain knowledge on the processes in the human brain. Hence, linear classifiers are chosen. Furthermore, if interpreting the model is not required and the image classification performance is the only concern, a non-linear transformation can be done prior to the linear classifiers described below to obtain similar performance to non-linear methods. This way these methods may be adapted to non-linear data with proper transformations.

a) Linear Discriminant Analysis (LDA): In a classification problem, there are predictors (which corresponds to the FMRI voxels in our case) and response which has discrete values. In our example, there are eight ($K = 8$) classes. Linear Discriminant Analysis assumes that each category has its own Gaussian distribution and predictors are sampled from the Gaussian of the corresponding category [7]. The multivariate Gaussian of k^{th} category can be written as:

$$f_k(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)} \tag{2}$$

$$\mathbb{P}(C = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{i=1}^{K} f_i(x)\pi_i} \tag{3}$$

where d is the number of predictors and $\sum_{i=1}^{K} \pi_i = 1$. Also, the parameters of the Gaussian are intentionally written as μ_k and Σ because LDA assumes that the categories have different mean but the covariance matrices are the same for all categories [7]. Therefore, the probability of one data point belonging to the category k can be compared to another category l using the following log-odds [8]:

$$\log \frac{\mathbb{P}\left(C = k|X = x\right)}{\mathbb{P}\left(C = k|X = x\right)} = \log \frac{f_k(x)}{f_l(x)} + \log \frac{\pi_k}{\pi_l} \tag{4}$$

The decision boundary between the categories k and l is linear in x according to this log-odds function [7]. Since this is true for all other classes as well, the decision boundaries are linear in LDA. If the parameters are known detecting the category of a given data point is very easy except in some special cases where the data is very close to the decision boundary. However, the parameters of Gaussians are not known in practice. These parameters should be estimated to construct a model. This model can be used to predict the class of new data or the model can be inspected on its own to characterize the system which creates the data. The estimates are as follows [7]:

$$\hat{\pi}_k = N_k/N \tag{5}$$

$$\hat{\mu}_k = \sum_{i:c_i=k} \frac{\mathbf{x}_i}{N_k} \tag{6}$$

$$\hat{\Sigma} = \frac{\sum_{k=1}^{K} \sum_{i:c_i=k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T}{N - K} \tag{7}$$

where N_k is the number of data instances that belongs to category k. The decision rule can be written as linear discriminant functions by replacing model parameters with these estimates.

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \tag{8}$$

and

$$C(x) = arg\, max_k \delta_k(x) \tag{9}$$

These definitions and equations can be used to form decision boundaries for data and new unseen data can be predicted using them. The "LinearDiscriminantAnalysis()" function of the Python library "sklearn" is used in the implementation.

b) Support Vector Classifier/Machine (SVM): The algorithm referred to by SVM is a linear support vector machine in this project. Separating two linearly separable groups can be achieved by finding one hyperplane which separates them. However, this separation can generally be achieved using infinitely many hyperplanes. A classifier that finds random one of such hyperplanes can be very sensitive to individual observations [9]. To minimize the sensitivity of the classifier to observed data an optimal hyperplane among all possible hyperplanes should be found. Support Vector Machines find this hyperplane which is equidistant to the nearest element of each class. The solution of the Support Vector Classifier can be found by solving the following optimization problem:

$$maximize_{\beta_0 \ldots \beta_d, \epsilon_0 \ldots \epsilon_n} M \tag{10}$$

subject to

$$\sum_{j=1}^{p} \beta_j^2, \tag{11}$$

$$c_i(\beta_0 + \beta_1 x_1 ... + \beta_d x_d) \geq M(1 - \epsilon_i) \tag{12}$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq Constant \tag{13}$$

where $Constant$ is a non-negative tuning parameter, c_i is the class of the i^{th} data point, x are predictors, d is the dimension (feature count) and n is the number of the data and M is the width of the margin described above. As mentioned earlier in this section we want this margin to be as high as possible. $\epsilon_1 ... \epsilon_n$ are slack variables that allow corresponding data points to be classified wrong [10]. It is important to highlight that only data points which lie on the margin or violates it affects the position of the margin and they are called the support vectors. The observations which are correctly classified do not affect SVC. The parameter C can be used to control the bias-variance trade-off of the classifier. We can treat it like the inverse of the parameter λ which is described in the regularization subsection below. The "LinearSVC()" function of Python library "sklearn" is used in this project to implement support vector classifiers.

c) Logistic Regression (LR): Logistic regression models the probability of a data point (x) belongs to a category as a linear function of x. Therefore, this function of x should sum up to one for all values of the response and remain between $[0, 1]$. The following canonical model is used to achieve this effect for K number of categories [11]:

$$\log \left(\frac{\mathbb{P}(C = 1|X = x)}{\mathbb{P}(C = K|X = x)} \right) = \beta_{10} + \boldsymbol{\beta}_1^T x \tag{14}$$

$$\log \left(\frac{\mathbb{P}(C = 2|X = x)}{\mathbb{P}(C = K|X = x)} \right) = \beta_{10} + \boldsymbol{\beta}_1^T x \tag{15}$$

...

$$\log \left(\frac{\mathbb{P}(C = K - 1|X = x)}{\mathbb{P}(C = K|X = x)} \right) = \beta_{10} + \boldsymbol{\beta}_1^T x \tag{16}$$

This model specifies the probabilities using K-1 logit functions and satisfies the property of summing up to one. By manipulating the logit functions above, the following probability calculations for category $l \in [K - 1]$ can be obtained:

$$\mathbb{P}(C = l|X = x) = \frac{exp(\beta_{l0} + \boldsymbol{\beta}_l^T x)}{1 + \sum_{t=1}^{K-1} exp(\beta_{t0} + \boldsymbol{\beta}_t^T x)} \tag{17}$$

$$\mathbb{P}(C = K|X = x) = \frac{1}{1 + \sum_{t=1}^{K-1} exp(\beta_{t0} + \boldsymbol{\beta}_t^T x)} \tag{18}$$

Clearly, they sum to one. Depending on the number of categories (K) and the dimension of the data (d), this model requires $(d + 1) \times (K - 1)$ parameters. If we denote the parameter set as $\theta = \{\beta_{10}, \beta_1 ... \beta_{K-1}, \beta_{K-1}\}$. The probabilities of the classes can be written as:

$$\mathbb{P}(C = l|X = x) = p_l(x; \theta) \tag{19}$$

This model with $K = 8$ is used to classify our data when the algorithm labelled as LR or logistic regression is used. The model can be trained by writing the likelihood function and maximizing it. When $K = 8$ and N is the number of data samples, the likelihood function can be written as:

$$L(\theta) = \prod_{i=1}^{N} \left(\prod_{l=1}^{K} (p_l(x; \theta))^{\mathbb{I}(c_i = l)} \right) \tag{20}$$

After taking the logarithm of the likelihood and further calculations, the following log-likelihood function is found:

$$\ell(\theta) = \sum_{i=1}^{N} \left(\sum_{l=1}^{K-1} \mathbb{I}(c_i = l) \boldsymbol{\beta}_l^T \mathbf{x}_i - \log \left(1 + \sum_{j=1}^{K-1} e^{\beta_l^T \mathbf{x}} \right) \right) \tag{21}$$

Since the above equation cannot be solved for a global optimum. It is solved using heuristic algorithms such as the Newton-Raphson method in practice. In our implementation "liblinear" library is used. These algorithms are guaranteed to converge to a local optimum but the converged point may not be the global optimum. The regression coefficients can be regularized using penalty factors. The penalty types indicated by 'l1' and 'l2' correspond to the regularization technique.

Regularization is achieved by penalizing high coefficients and optimizing the following function:

$$\ell_{l1}(\theta) = \ell(\theta) - \lambda \sum_{j=1}^{d} |\beta_j| \tag{22}$$

$$\ell_{l2}(\theta) = \ell(\theta) - \lambda \sum_{j=1}^{d} \beta_j^2 \tag{23}$$

The regularization parameter which is denoted as λ in the equation above should be tuned and its optimal value should be used for high performance. The optimal regularization parameter ($\lambda_\star$) is determined by cross-validation in this project.

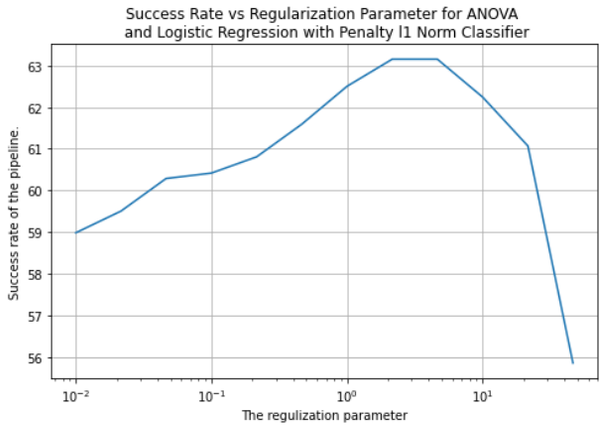

Fig. 3. The validation curve of ANOVA and Logistic Regression with l1 regularization. Different penalty factors are used to train the model. The obtained models are used to predict the validation data in each run. Due to limited number of data in data set, 12-fold cross validation is used. The fold number is chosen as 12 because there are 12 experiments runs in the data set. Some other number can be chosen as well.

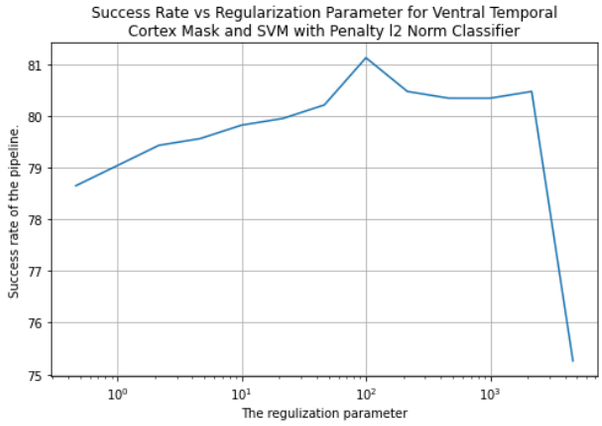

Fig. 4. The validation curve of Ventral Temporal Masking and Support Vector Classifier with l2 regularization. Different penalty factors are used to train the model. The obtained models are used to predict the validation data in each run. Due to limited number of data in data set, 12-fold cross validation is used. The fold number is chosen as 12 because there are 12 experiments runs in the data set. Some other number can be chosen as well.

d) Regularization: The measurements of biological systems are very prone to noise. This has two major reasons. Firstly, the biological responses are not completely deterministic. For example, a neuron can give different responses to the same input. Secondly, the measurement devices are not perfect even when the latest technology is used. In our dataset, some measurements are distorted. One major cause of these distortions is the head movement of the subjects. The subjects are asked to stand still for at least 5 minutes and this process is repeated 12 times. It is very likely that the quality of some measurements is not very good. These errors and the randomness of the system are modelled as noise in our models. All input predictors have high noise. The models try to minimize the loss in the observed data which includes this high noise. If our model perfectly fits the currently observed data, it will likely fit the noise as well. This is called overfitting. The easiest and most common way to deal with this noise in our algorithms is to regularize the model by placing a penalty on high coefficients. By using this method the model does not over-fit the data and does not capture the noise in it. The regularized model is a biased estimate of the real model but according to the bias-variance trade-off, it gives a lower error value. However, the regularization parameter should be tuned and the best value must be found to balance the trade-off between bias and variance. Cross-validation is used for this purpose and accuracy of the same model with different regularization parameters is found. After that, the best parameter is found and it is used in the tests. Two examples of these graphs are in Fig. 3 and Fig. 4. Others are added to the appendix.

D. Metrics to Evaluate Statistical Significance of the Results

a) Significance Testing: Significance testing can be done using various methods, in this section, we summarize our method to evaluate the statistical significance of the results of this project. The null hypothesis is chosen as the event that the classifier pipelines uniformly classifies an instance. This classification corresponds to the performance of the Dummy Classifier whose performance is shown with our pipelines in Fig. 17. After that, the event that this classification method leads to correct classifications is modelled as a binomial. Then central limit theorem is used to predict the probability that our event happens under the null hypothesis. The Z-score of this event is used to predict the probability. Lastly, this probability is inspected to decide whether the null hypothesis could be rejected. If the null hypothesis is rejected our methods perform significantly better than the dummy classifier. A similar approach can be used to evaluate the performance difference between our implemented pipelines. The dummy classifier in this method can be changed to another classifier.

b) Effect Size: Effect size measures the relationship strength between two variables from the same population. We used this metric to show the meaningfulness of our results along with significance. It shows the correctness of a statistical claim. The following formula is used to calculate effect size:

$$\frac{\mu_1 - \mu_2}{\sigma} \tag{24}$$

In this formula, μ_1 corresponds to the correct number of classification of our method. μ_2 is the expected number of correct classification of uniform class assignments (dummy classifier). σ is the standard deviation of the correct assignments of the

dummy classifier. The Dummy classifier's correct assignment is a binomial with $\frac{1}{n}$ probability and the number of trials is the number of data points so this value can be found for all classifiers to evaluate the significance of our results. Also, the same metric can be used to compare two classifiers. μ values can be found by success rate and σ is the standard deviation of either of them.

III. Results

In this paper in total 15 pipelines are compared to each other with respect to their ability to classify FMRI images. To compare different pipelines each of their confusion charts of the data over 12-fold cross-validation are drawn. To save space only some of the confusion charts are shown in this chapter. However, confusion charts for all 15 pipelines can be found in the appendix.

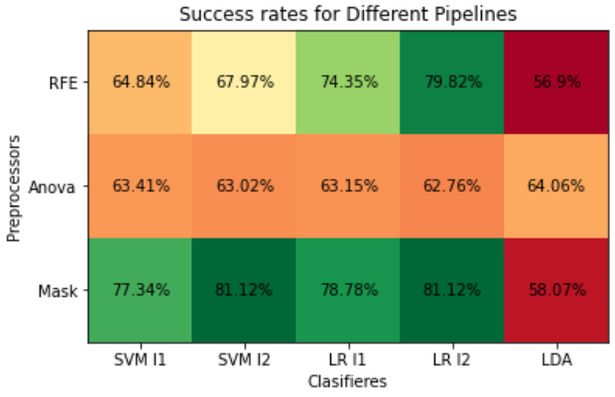

Fig. 5. The chart shows the success rate of the predictions models trained by the corresponding pipelines. Rows show the feature selection method, columns show the classifier method. An overview of the performance of the methods used can be seen in this chart. More detailed explanation are made in the following paragraphs and some confusion matrices are included to give better intuition of the performance of the methods.

A. Ventral Temporal Cortex Mask:

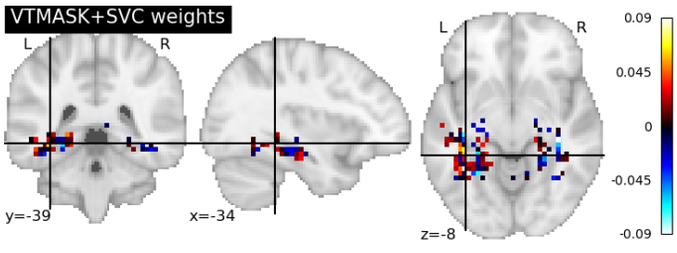

Fig. 6. The important voxels which are found using Ventral Temporal Mask and SVM classifier pipeline. Only "face" and "cat" classes are used for this training so these voxels show the neurons which identify faces and cats.

The results of the ventral temporal cortex mask with l2 norm SVM classifier are shown in Fig. 7. The success rate of this pipeline is 81.12%. Mask pre-processing gives high success rates for all the classifiers other than LDA, as it can be seen in Fig 5. These high success rates prove that the ventral temporal cortex mask is successful at lowering the dimensionality of the data and preparing it for successful classification. One of the

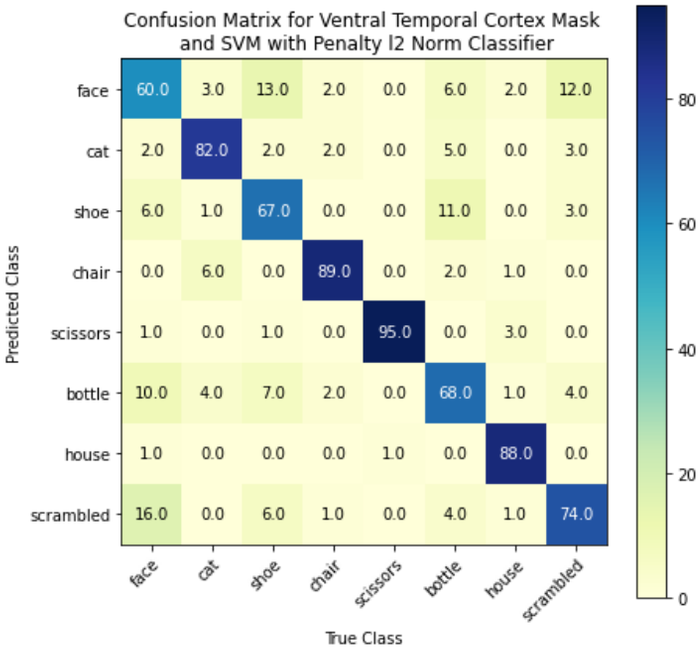

Fig. 7. confusion chart for mask pre-processing and SVM classification with penalty l2 norm for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. For the null hypothesis of a dummy classifier, the significance value of this result is less than 10^{-325} with an effect size of 57.50.

most confused class pairs are shoes and bottles. These object pairs are visually similar too and a human might confuse one for the other in a glance. So, the errors are not random instead they stem from the similar responses similar-looking objects create in the ventral temporal cortex of the brain. The ventral temporal cortex for the second subject can be seen in Fig. 6.

B. Recursive Feature Elimination:

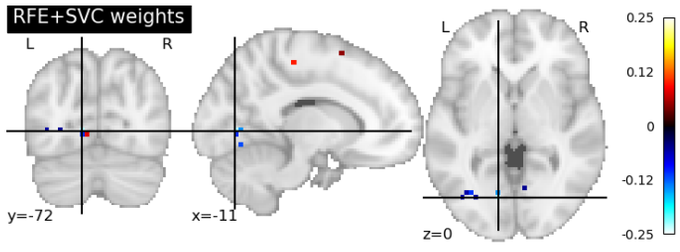

Fig. 8. The important voxels which are found using RFE pre-processing and SVM classifier pipeline. Only "face" and "cat" classes are used for this training so these voxels show the neurons which identify faces and cats.

The success rate of RFE is 67.97% if it is used with SVM with l2 norm. This is lower than the success of SVM for mask pre-processing. However, the success rate of RFE for logistic regression with l2 norm penalty (79.82%) is very close to the success rate of ventral temporal cortex mask for logistic regression with l2 norm penalty (81.12%). These can be seen in 5. These results illustrate that RFE can be as successful as the ventral temporal cortex mask if it is used with the right classifier. So, RFE is a dimension reduction method that can prepare FRMI data for successful classification. The

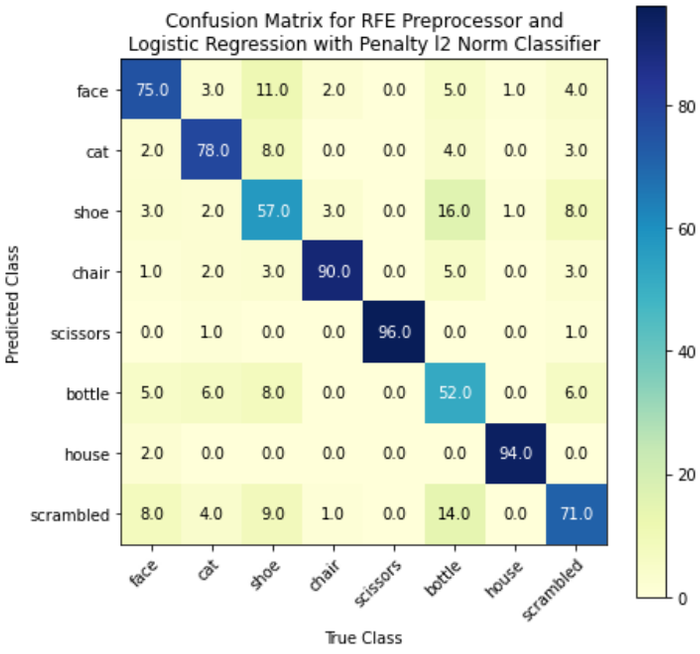

Fig. 9. confusion chart for RFE pre-processing and logistic regression classification with penalty l2 norm for the data over 12 fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 56.41.

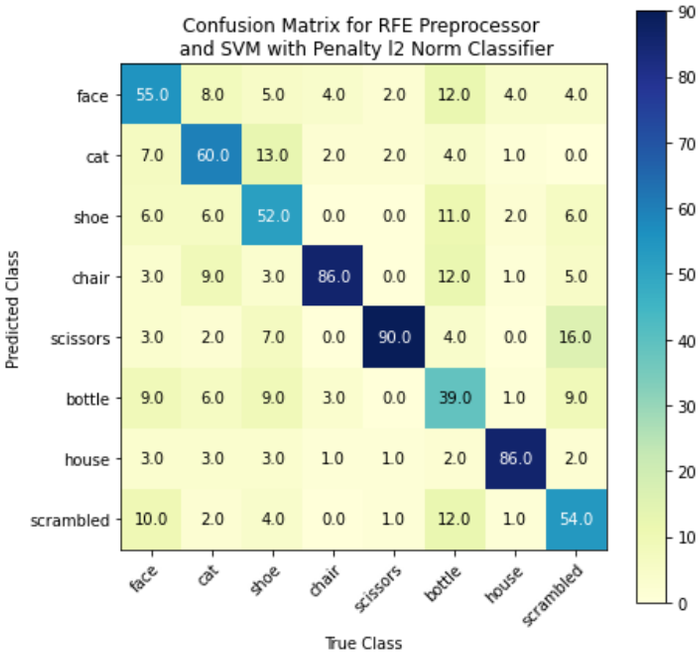

Fig. 10. confusion chart for RFE pre-processing and SVM classification with penalty l2 norm for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 46.48.

higher error on the SVM classifier could occur because the dimension reduction done by the RFE is making the data less linearly separable. This is problematic for SVM because linear SVM is being used. If Fig. 9 is examined the most commonly confused objects are bottles and shoes or faces and shoes images. Bottles and shoes resemble each other and mixing them is a common occurrence in all the pipelines as they can be seen in the appendix. However, if Fig. 10 is examined there are very strange confusions such as mixing bottles as chairs or bottles as scrambled images. These newly mixed object pairs have no visual similarity and their mixing can be explained by the effect of RFE instead of actual similar effects of these objects on the brain. Also comparing the important voxels chosen by the RFE and the important voxels chosen by the ventral temporal cortex mask which are respectively given in Fig. 8. and Fig. 6, reveal voxels chosen by RFE mostly fall on the ventral temporal cortex. So RFE can identify the high-level visual processing region of the brain from full brain scans. This is the reason why RFE pre-processing can give as accurate results as ventral temporal cortex mask pre-processing which uses domain knowledge.

C. ANOVA

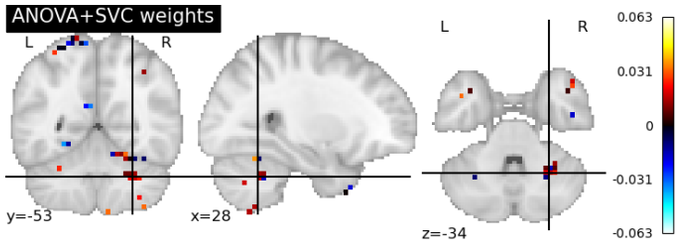

Fig. 11. The important voxels which are found using ANOVA pre-processing and SVM classifier pipeline. Only ”face” and ”cat” classes are used for this training so these voxels show the neurons which identify faces and cats.

The confusion chart in Fig. 12 has an accuracy rate of 63.02%. ANOVA pre-processing has a less than 65% accuracy rate for all classifiers. This accuracy is low. It is giving such bad results because ANOVA only picks voxels with high variance so it is picking some relatively random voxels with high variance in the brain. As it can be seen in Fig. 11 these voxels can be very far from the visual cortex and those voxels which are far can have no correlation to the seen images. Understandably this is decreasing the success rate of the used classifiers. ANOVA pre-processing gives its highest success rate which is 64.06%, for the LDA classifier. This happens because ANOVA reduces the dimensionality to 50 without using the classification method. Why this is important will be further elaborated later. Finally, examining further the Fig. 12, reveals confusion between objects which are visually similar is still very prevalent. However, there are more confusions that seem arbitrary such as confusing cats with bottles. These confusions occur due to the seemingly random voxels ANOVA chooses as important.

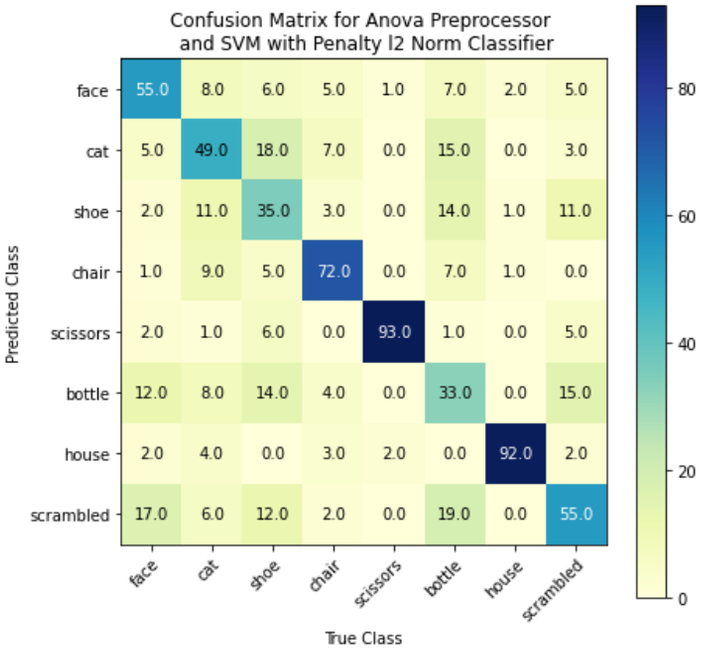

Fig. 12. confusion chart for ANOVA pre-processing and SVM classification with penalty l2 norm for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 42.33.

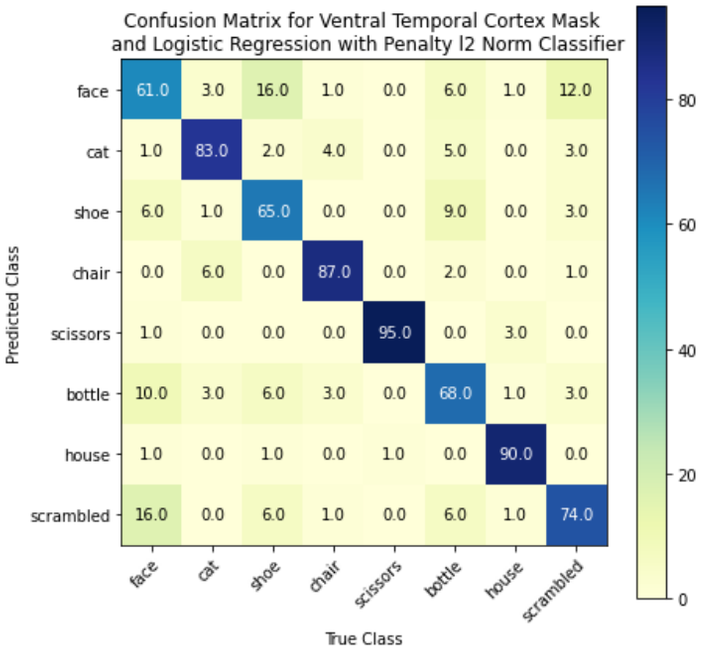

Fig. 13. confusion chart for mask pre-processing and logistic regression classification with penalty l2 norm for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 57.50.

D. Support Vector Machine:

For almost all pre-processing methods either l2 norm penalty is better than l1 norm penalty or their success rates are almost the same. These results can be seen in 5. The highest success rate for SVM is for l2 norm penalty after ventral temporal cortex mask. The results of the l2 norm regularization SVM classifier after ventral temporal cortex mask are illustrated in Fig 7. The success rate of this pipeline is 81.12%. SVM is the most common classification method used in the classification of FRMI data, and as expected it is giving a significantly high success rate.

E. Logistic Regression:

For Logistic regression, l2 norm penalty gives a better result than l1 norm penalty gives for all pre-processing methods. The success rates for all these pipelines can be seen in 5. Logistic regression gives a significantly high success rate for both mask and RFE pre-processing but it gives the highest success rate for mask pre-processing. The result of this pipeline is given in Fig. 13 and it has a success rate of 81.12% which demonstrates logistic regression is great for classifying FMRI data.

F. Linear Discriminant Analysis:

The success rate of the LDA classifier for mask pre-processing and RFE pre-processing are both less than 60%. However, the success rate of the LDA classifier for ANOVA pre-processing is 64.06%. This is very interesting as this is the highest success rate for any pipeline which involves an ANOVA pre-processing or LDA classifier and this pipeline's confusion chart is given in Fig. 14. ANOVA and LDA give

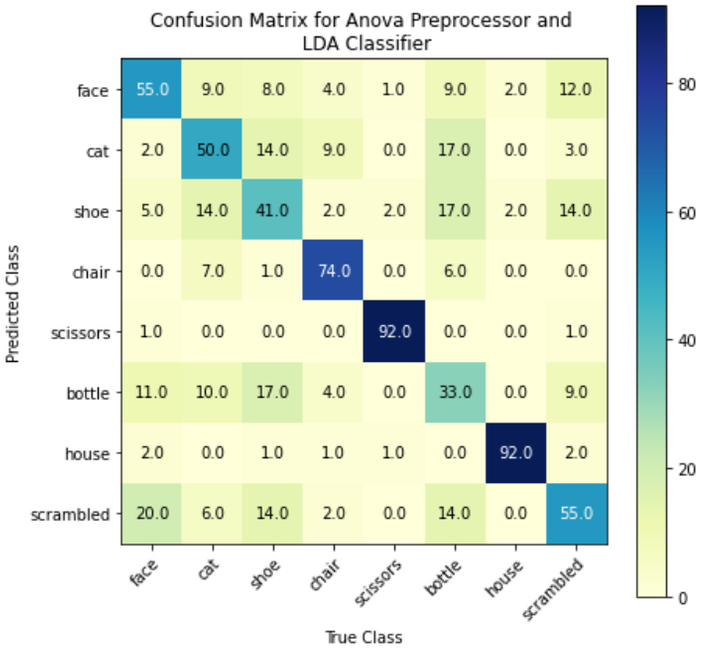

Fig. 14. confusion chart for ANOVA pre-processing and LDA classification for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 43.21.

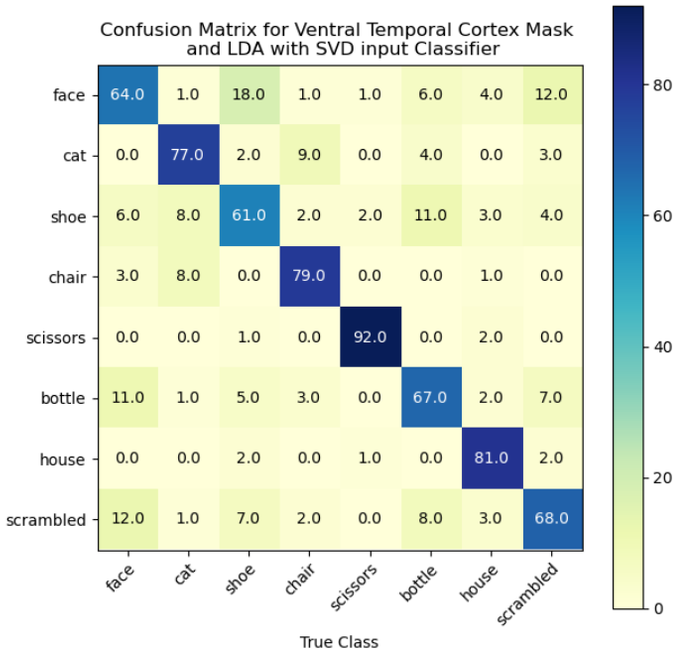

Fig. 15. Confusion chart for Ventral Temporal Masking and Singular Value Decomposition pre-processing and LDA classification for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325}. The performance of LDA is increased by using the singular value decomposition of the design matrix.

each other the highest success because ANOVA is decreasing the dimensionality to 50 and LDA works best for a low dimensinoal data. LDA has significantly lower performance compared to the other classification methods. However its success rate can be improved by using Singular Value Decomposition prior to LDA. To prove this SVD is used on the mask pre-processed FMRI data and this increased LDA's performance to 74.91%. The improvement on the LDA classifier can be seen by comparing Fig. 15 and Fig. 14. The potential reasons behind this observation are discussed in the following section.

IV. Discussion

A. Comparison and Importance of Results

In the end, all pipelines that were tested showed a better classification accuracy than that of the dummy classifier and the statistical significance of even the lowest success rate result is lower than 10^{-300}. So, it is clear that all the pre-processing and classification method pairs that were tested are able to recognize the class of the object seen by the subject from the FMRI scan. However, as it can be seen in Fig. 5 all the pre-processing and classification methods are not equally successful. The highest success rate which is 81.12%, this is achieved by ventral temporal cortex mask pre-processing and either SVM or logistic regression with penalty l2 norm. The best success rate getting shared between these two classification methods indicate that SVM with penalty l2 norm and logistic regression with penalty l2 norm are equally successful in classifying FMRI results. However, logistic regression is also very successful on RFE pre-processed data and this is its superiority compared to SVM. LDA might seem like a significantly worse classifier than SVM and logistic regression, However, LDA is not strictly worse than SVM or logistic regression, it only requires a more specific pre-processing. LDA requires low-dimensional data to function properly. The success rate in the temporal cortex mask pre-processing with LDA increases when SVD is applied to the masked data because SVD can reduce the dimension of the data which is classified by LDA and allows it to work properly [12]. In RFE in the first iteration LDA is used with over 36 000 dimensions and after that LDA is used with thousands of dimensions repeatedly. Since LDA is used for such high-dimensional data, this pipeline has the lowest success rate. ANOVA pre-processing chooses some very irrelevant voxels; however, it still lowers the dimension of the data to 50 and because of this it gave the highest success rate out of all pre-processing methods from the initial test. In the end, LDA is not a bad classifier it only requires the data to be low dimensional. RFE is not a commonly used feature selection method. However, RFE with logistic regression with l2 norm penalty was able to achieve a success rate of 79.82% which is only 1.30% less than the highest achieved success rate in the experiment. The highest result was achieved with ventral temporal cortex mask pre-processing, ventral temporal cortex of every subject is unique and it requires domain knowledge. RFE's ability to achieve a success rate which very close to the one achieved with the ventral temporal cortex mask and RFE's chosen important voxels being mostly in the ventral temporal cortex are two very significant results. The ANOVA pre-processing was chosen to highlight the difficulty of pre-processing FMRI data without using domain knowledge. ANOVA pre-processing was not able to choose most of its important voxels in the ventral temporal cortex and because of this, it gave success rates less than 65% for all the classifier methods. RFE is a very time-consuming algorithm and because of this it is rarely used however, with the new improvements in hardware technology, we believe it has become a contender for mask pre-processing. Because RFE pre-processing does not require domain knowledge, different subjects' full brain FMRI scan analysis can be quickly done with powerful enough hardware. Also, if an analysis is wanted to be done on a previously not studied problem. The brain-specific anatomical region can be found automatically by our RFE based pipelines. The analysis was successful in proving feature selection methods which do not need domain knowledge can be as successful as feature selection methods which use domain knowledge. However, the biggest obstacle against FMRI scan technology becoming widely used in daily life still stands. The effects of what a person sees appear delayed in brain FMRI scans. Because of this FMRI scans can not be used in any setting where the results are required quickly. An example of unimplementable technology could be, a headband which automatically gives the turn signals in a car. Also because of the delay, it is not

intuitive and fun to use. Because of this, it is not used in smart home solutions the same way voice commands are used. Until the problem related to this delay is slowed FMRI scans seem unlikely to be widely used in daily life.

B. Potential Improvements

The FMRI data used in the analysis have measurement noise in them due to the movements of the subjects. Because of these corrupted data, none of the success rates from the analysis reached the 90% others researchers achieved. These corrupted data could be fixed using a motion correction algorithm on the data before the pre-processing step [13]. The regularization parameter of the SVM and logistic regression for RFE pre-processing could not be chosen the same way other hyperparameters were chosen for other pre-processing methods. This is because of the long run time of the RFE algorithm. Instead of testing logarithmically spaced regularization parameters, Gaussian process optimization can be used for efficient hyperparameter tuning because our algorithm involving RFE is computationally expensive [14]. This way a smaller number of trials will be required to find a good regularization parameter for pipelines that use an RFE pre-processing and will be able to be done in a reasonable time frame. Using the gaussian process for hyperparameter tuning would also allow us to choose more accurate regularization parameters for all pipelines which would, in turn, increase the success rate of those pipelines. Because every human's brain is slightly different a model fit for a subject could not be used on the other subjects. By using a method such as Talairach coordinates FMRI data from different subjects can be transformed into some standard anatomical space [15]. This way a model could be trained using the data from all six subjects.

References

[1] K. A. Norman, S. M. Polyn, G. J. Detre, J. V. Haxby, "Beyond mind-reading: multi-voxel pattern analysis of FMRI data." Trends in Cognitive Sciences, 10(9), pp. 424—430, 2006. https://doi.org/10.1016/j.tics.2006.07.005

[2] J. V. Haxby, M. I. Gobbini, M. L. Furey, A. Ishai, J. L. Schouten, and P. Pietrini,Distributed and overlapping representations of faces and objects in ventral temporal cortex., Science, vol. 293, no. 5539, pp. 2425–2430, Sep. 2001.

[3] M. Hanke, Y.O.Halchenko, P.B.Sederberg, E. Olivetti, I. Fründ, J.W. Rieger, C.S. Herrmann, J.V. Haxby, S. Hanson and S. Pollmann (2009). PyMVPA: a unifying approach to the analysis of neuroscientific data. Frontiers in Neuroinformatics, 3:3.

[4] H. Y. Kim, Analysis of variance (ANOVA) comparing means of more than two groups., pp. 557-581, 2006.) http://dx.doi.org/10.5395/rde.2014.39.1.74

[5] L. Pessoa and S. Padmala, Decoding Near-Threshold Perception of Fear from Distributed Single-Trial Brain Activation., Cerebral Cortex, 3, pp. 691-701, March 2007. https://doi.org/10.1093/cercor/bhk020

[6] I. Guyon, J. Weston, S. Barnhill and V. Vapnik, Gene Selection for Cancer Classification using Support Vector Machines., Machine Learning, 46, pp. 389-422, 2002. https://doi.org/10.1023/A:1012487302797

[7] T. Hastie, R. Tibshirani, and J. H. Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. New York: Springer, pp. 106-111, 2001.

[8] G. James, D. Witten, T. Hastie, R. Tibshirani. An introduction to statistical learning : with applications in R. New York :Springer, pp. 139-148 2013.

[9] G. James, D. Witten, T. Hastie, R. Tibshirani. An introduction to statistical learning : with applications in R. New York :Springer, pp. 346-360, 2013.

[10] T. Hastie, R. Tibshirani, and J. H. Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. New York: Springer, pp. 436-446, 2001.

[11] T. Hastie, R. Tibshirani, and J. H. Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. New York: Springer, pp. 119-125, 2001.

[12] S. J. Hanson , T. Matsuka, J. V. Haxby Combinatorial codes in ventral temporal lobe for object recognition: Haxby (2001) revisited: is there a "face" area? Neuroimage. pp. 156-66. 2004. doi:10.1016/j.neuroimage.2004.05.020. PMID: 15325362.

[13] M. Zaitsev, B. Akin and B. Knowles, "Prospective Motion Correction in Functional MRI," NeuroImage, vol. 154, pp. 33-42, 1 July 2017.

[14] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," arXiv preprint arXiv:0912.3995, 2009

[15] J. Talairach and P. Tournoux. Co-planar stereotaxic atlas of the human brain. Thieme, New York, 1988.

Appendix

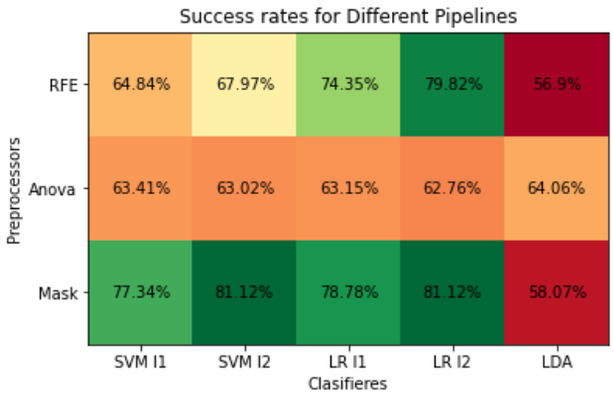

Fig. 16. The chart shows the success rate of the predictions models trained by the corresponding pipelines. Rows show the feature selection method, columns show the classifier method. An overview of the performance of the methods used can be seen in this chart.

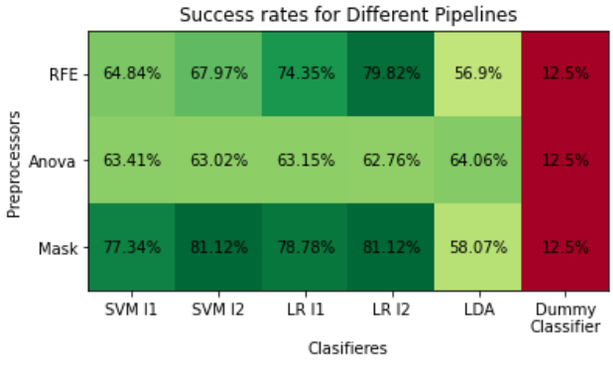

Fig. 17. The chart shows the success rate of the predictions models trained by the corresponding pipelines (Dummy Classifier included). Rows show the feature selection method, columns show the classifier method. An overview of the performance of the methods used can be seen in this chart.

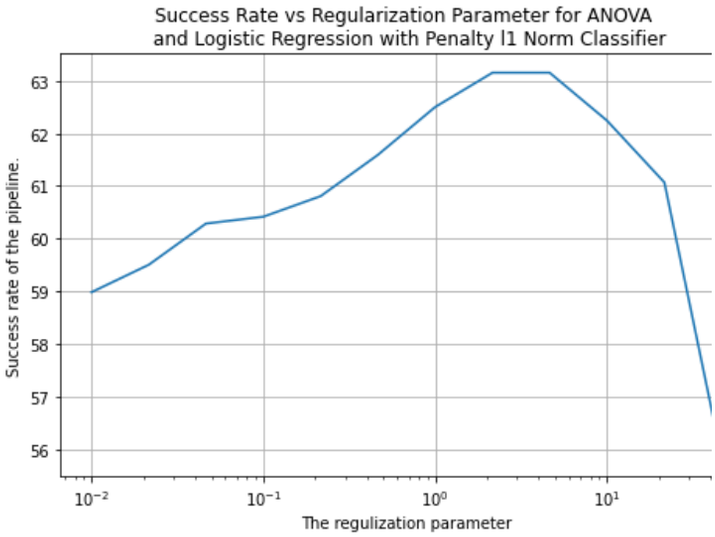

Fig. 18. Success rate vs. The regularization parameter of the pipeline plot for ANOVA and Logistic Regression with Penalty l1 Norm Classifier

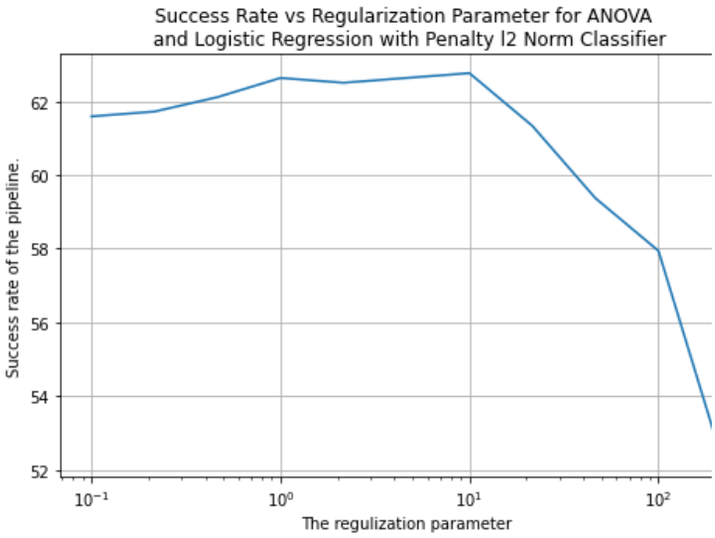

Fig. 19. Success rate vs. The regularization parameter of the pipeline plot for ANOVA and Logistic Regression with Penalty l2 Norm Classifier

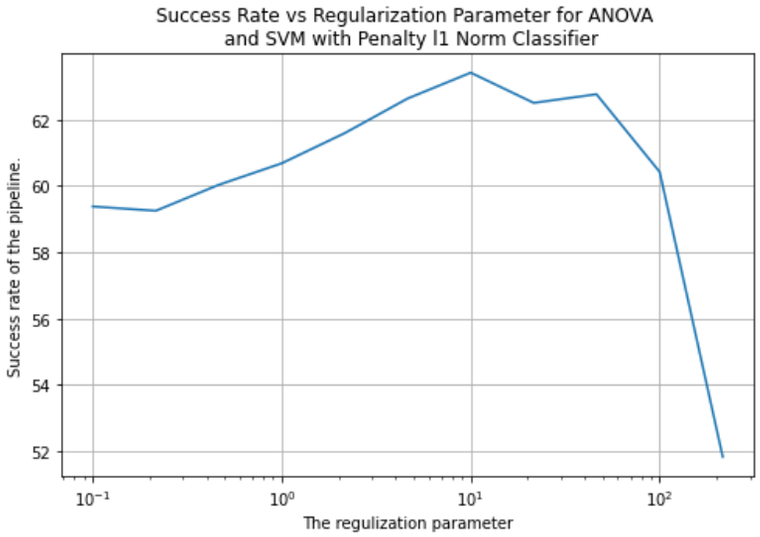

Fig. 20. Success rate vs. The regularization parameter of the pipeline plot for ANOVA and SVM with Penalty l1 Norm Classifier

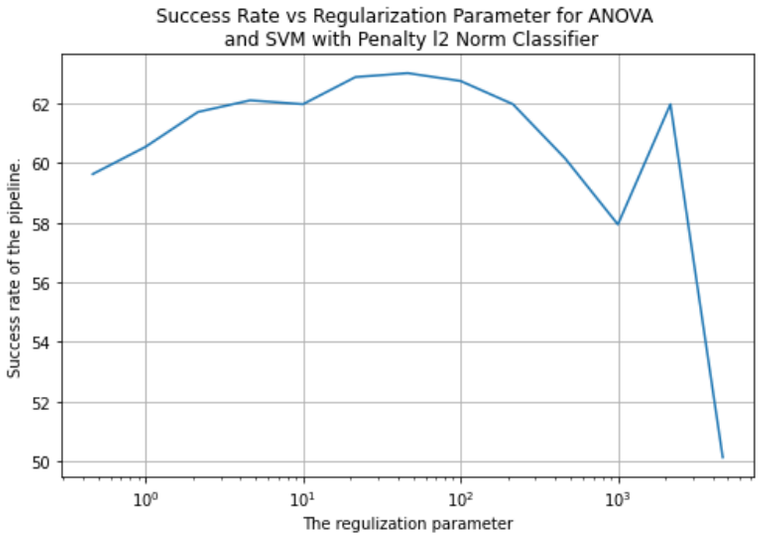

Fig. 21. Success rate vs. The regularization parameter of the pipeline plot for ANOVA and SVM with Penalty l2 Norm Classifier

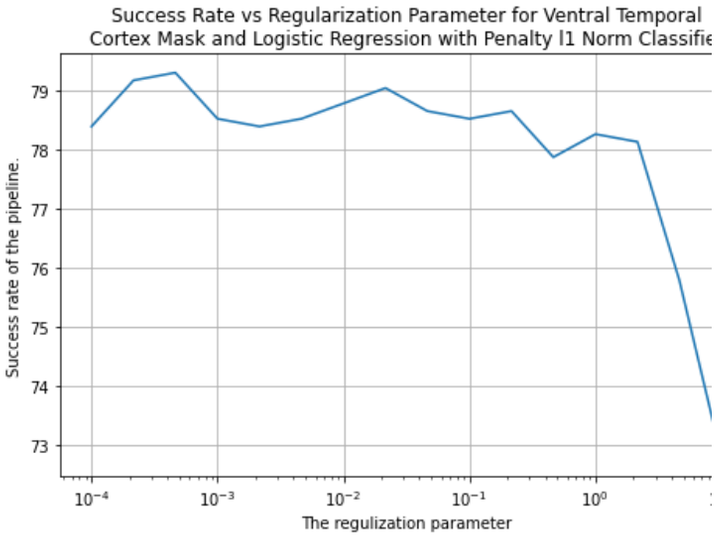

Fig. 22. Success rate vs. The regularization parameter of the pipeline plot for Ventral Cortex Mask and Logistic Regression with Penalty l1 Norm Classifier

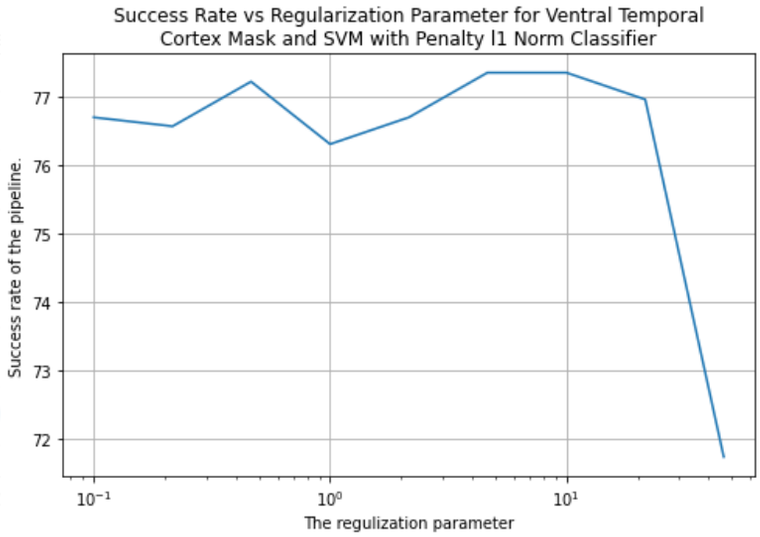

Fig. 24. Success rate vs. The regularization parameter of the pipeline plot for Ventral Cortex Mask and SVM with Penalty l1 Norm Classifier

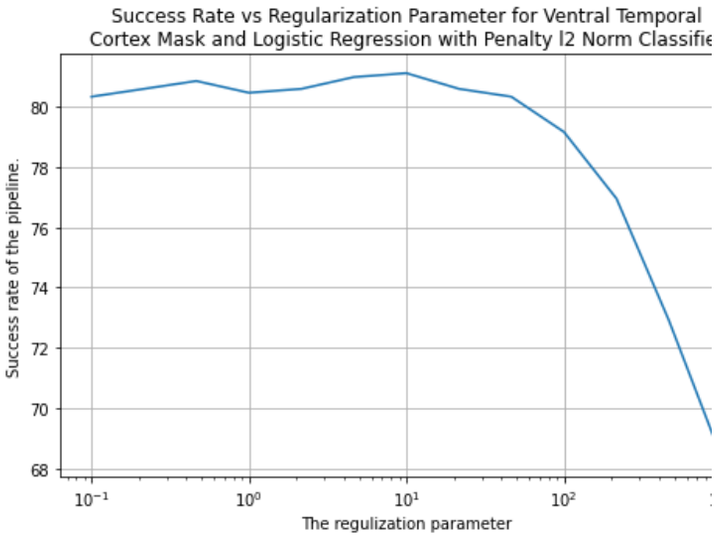

Fig. 23. Success rate vs. The regularization parameter of the pipeline plot for Ventral Cortex Mask and Logistic Regression with Penalty l2 Norm Classifier

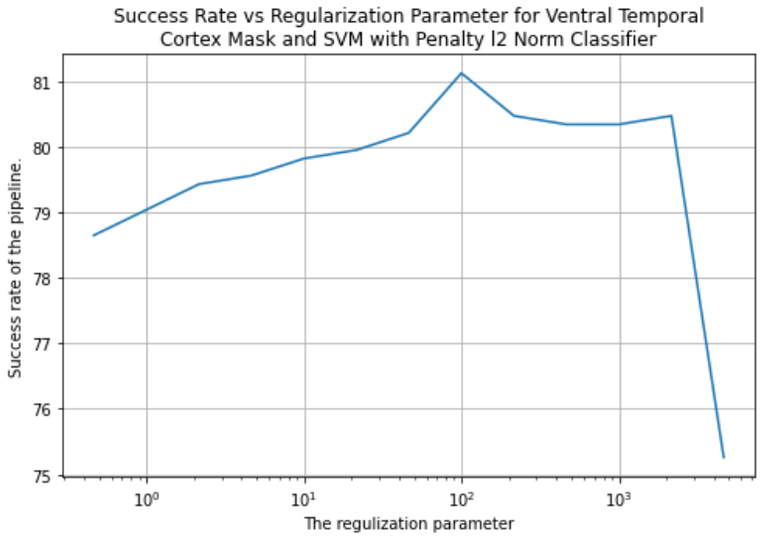

Fig. 25. Success rate vs. The regularization parameter of the pipeline plot for Ventral Cortex Mask and SVM with Penalty l2 Norm Classifier

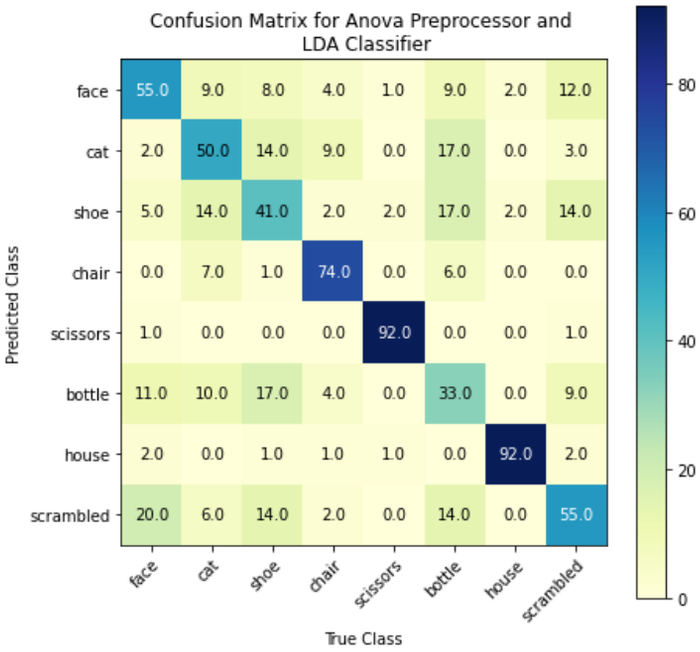

Fig. 26. Confusion chart for ANOVA pre-processing and LDA classification for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 43.21.

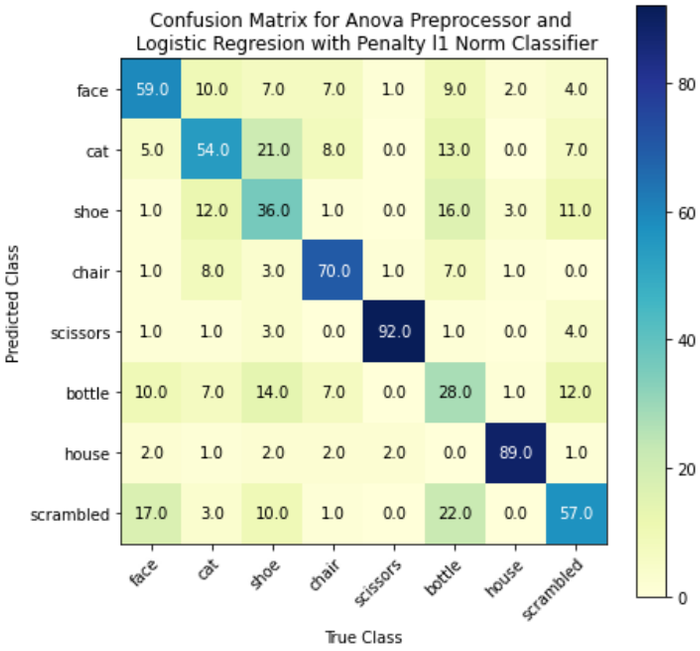

Fig. 27. Confusion chart for ANOVA pre-processing and Logistic regression l1 for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 42.44.

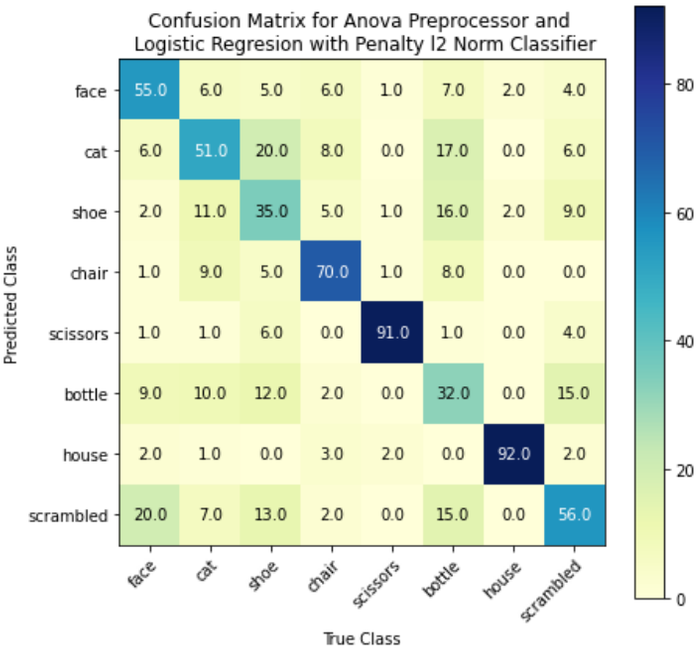

Fig. 28. Confusion chart for ANOVA pre-processing and Logistic regression l2 for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 42.12.

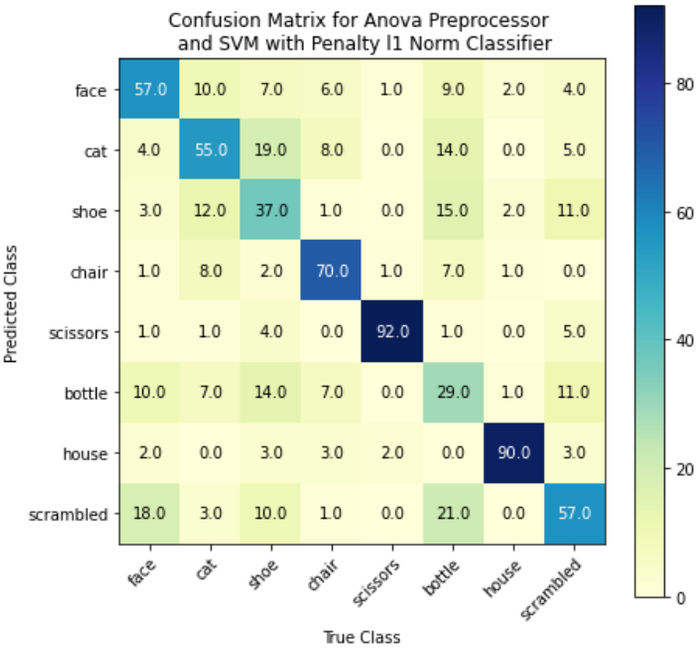

Fig. 29. Confusion chart for ANOVA pre-processing and SWM with Penalty l1 for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 42.66.

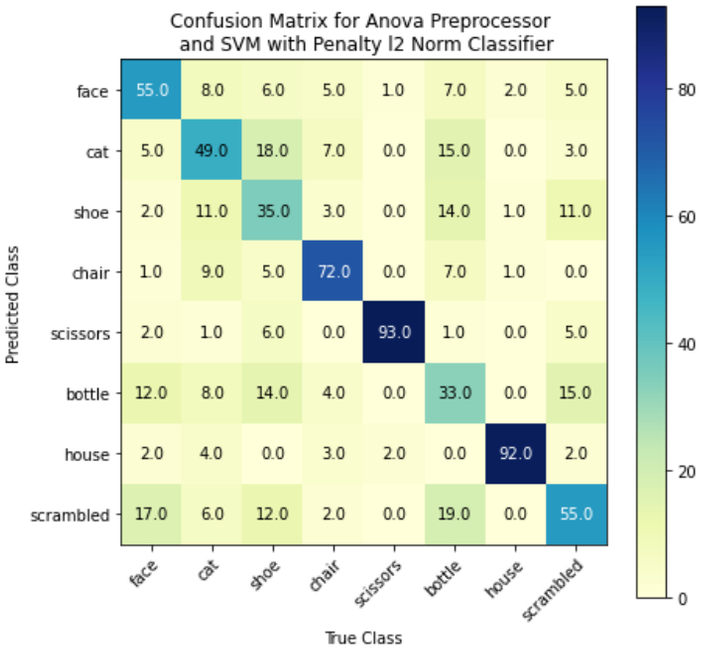

Fig. 30. Confusion chart for ANOVA pre-processing and SWM with Penalty l2 for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 42.33.

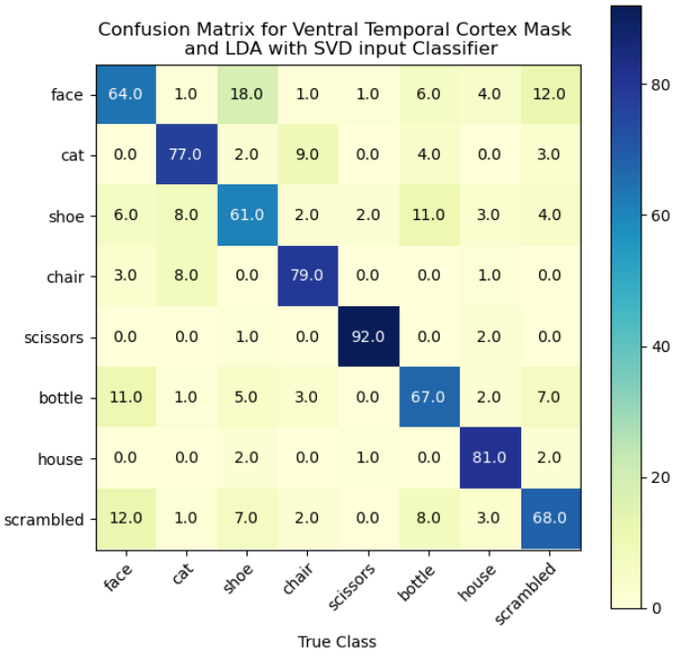

Fig. 31. Confusion chart for Mask and LDA with SVD input classifier for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325}.

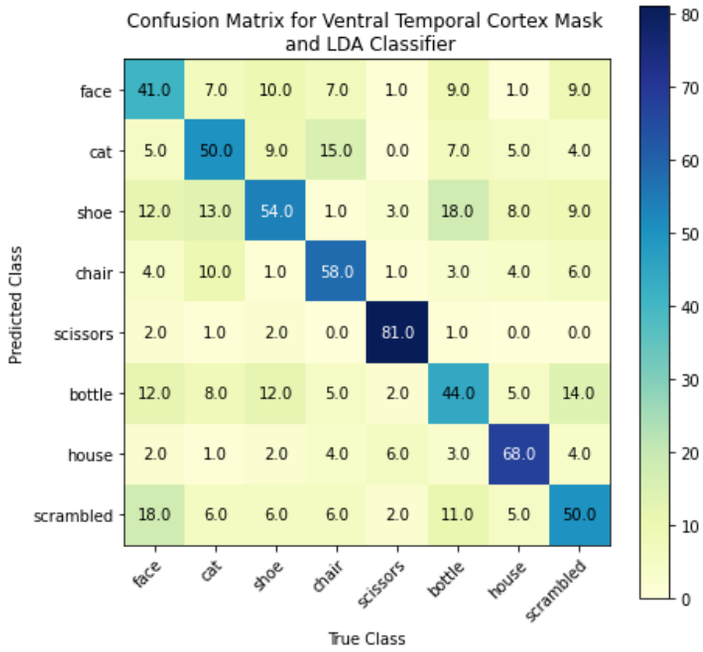

Fig. 32. Confusion chart for Mask and LDA for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is $5.32 \bullet 10^{-318}$ with an effect size of 38.19.

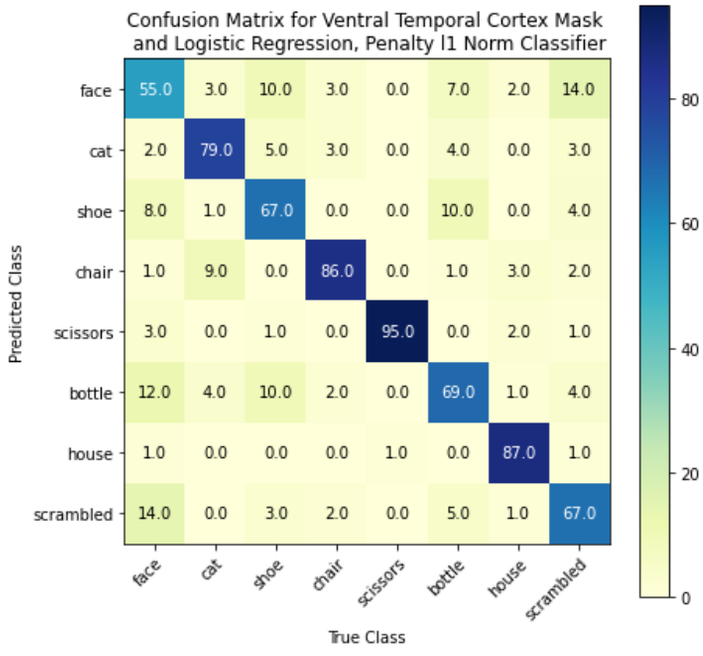

Fig. 33. Confusion chart for Mask and Logistic Regression, Penalty l1 Norm Classifier for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 55.54.

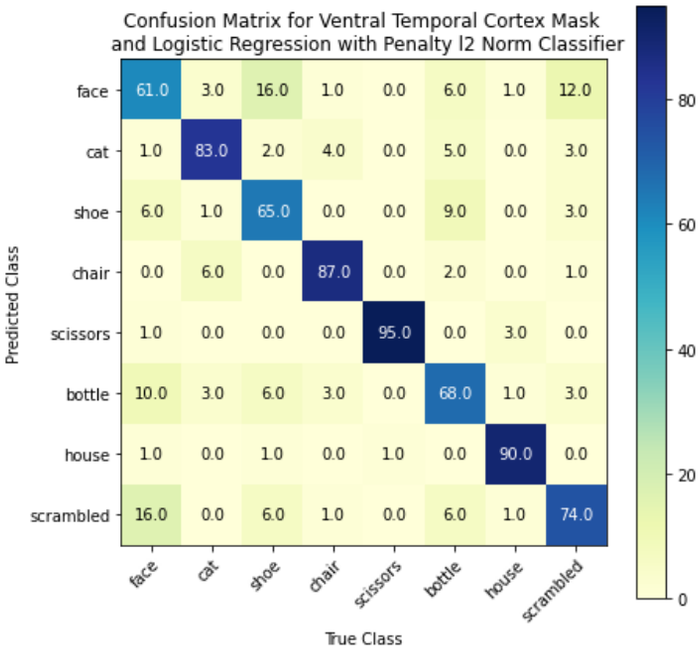

Fig. 34. Confusion chart for Mask and Logistic Regression, Penalty l2 Norm Classifier for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 57.50.

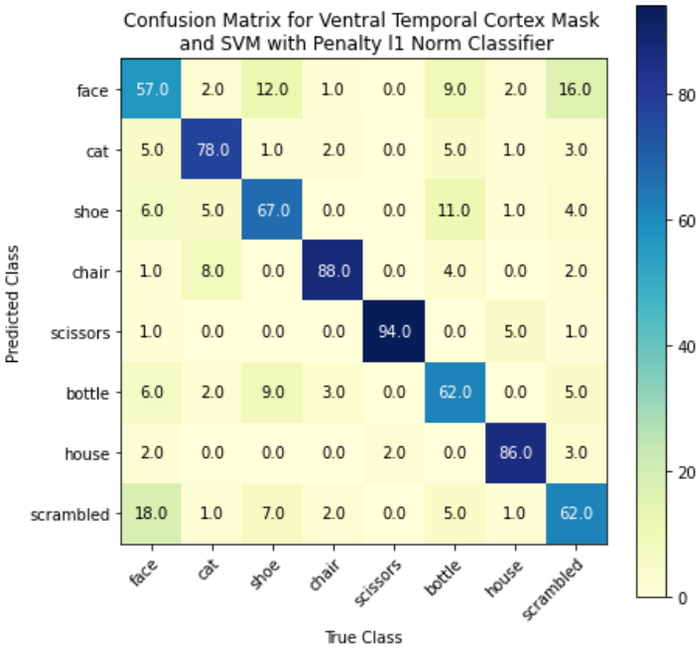

Fig. 35. Confusion chart for Mask and SWM, Penalty l1 Norm Classifier for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 54.34.

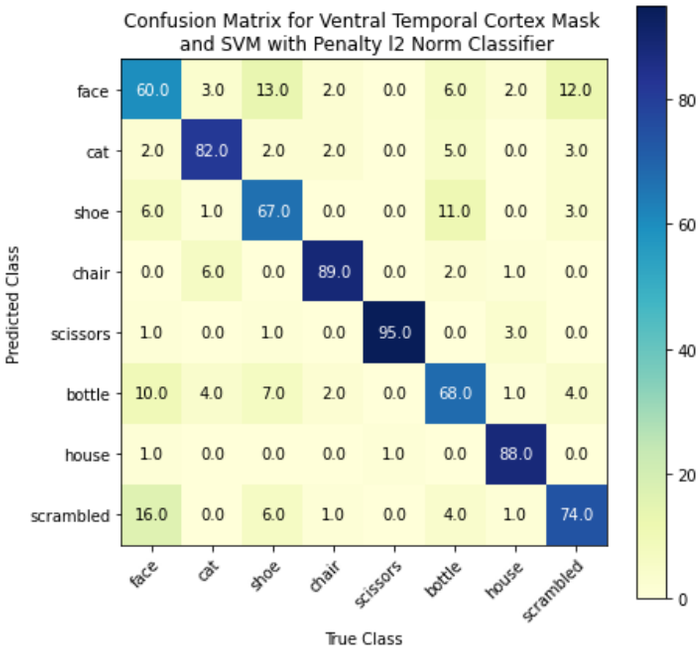

Fig. 36. Confusion chart for Mask and SWM, Penalty l2 Norm Classifier for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 57.50.

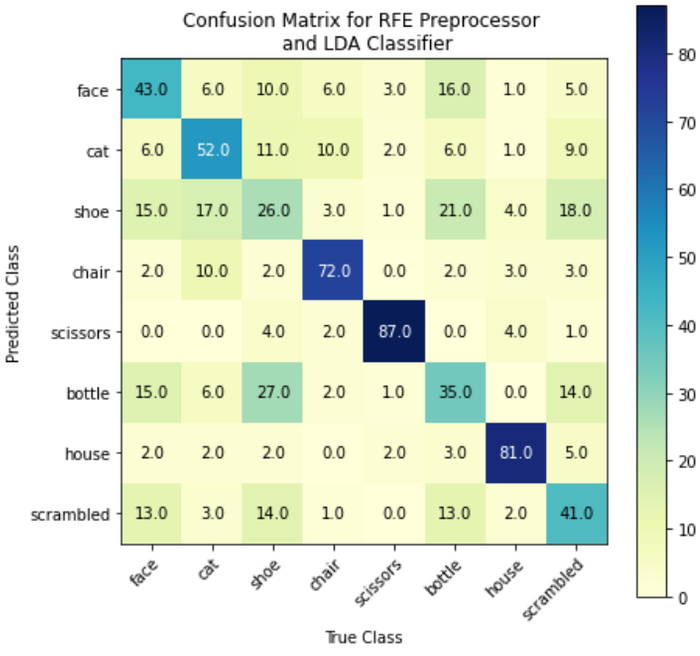

Fig. 37. Confusion chart for RFE and LDA Classifier for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is $2.71 \bullet 10^{-303}$ with an effect size of 37.21.

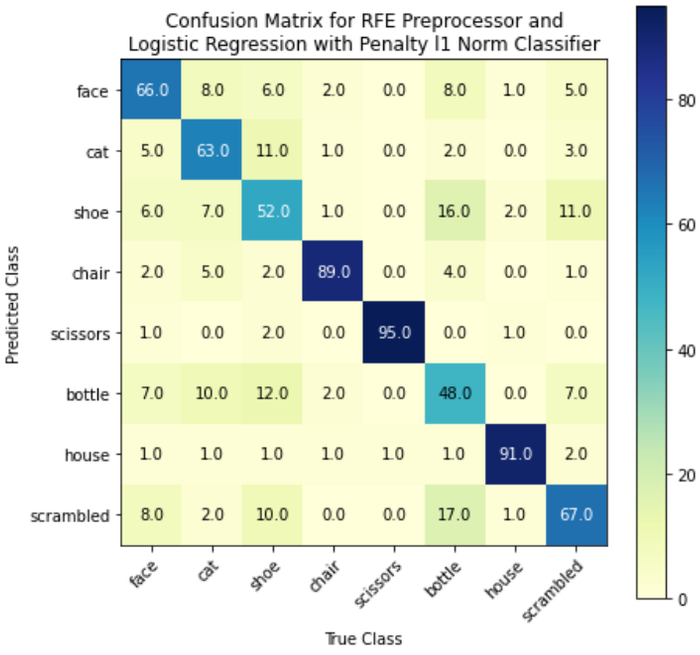

Fig. 38. Confusion chart for RFE and Logistic regression with Penalty l1 Norm Classifier for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 51.83.

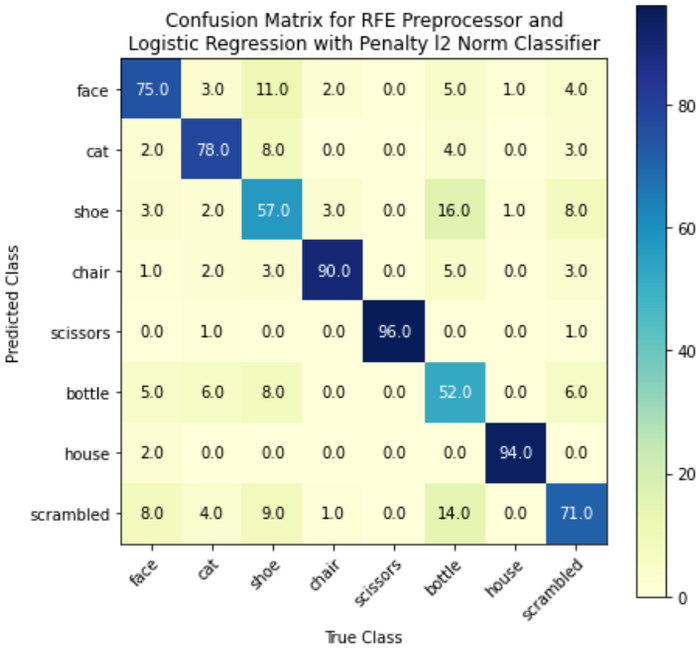

Fig. 39. Confusion chart for RFE and Logistic regression with Penalty l1 Norm Classifier for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 56.41.

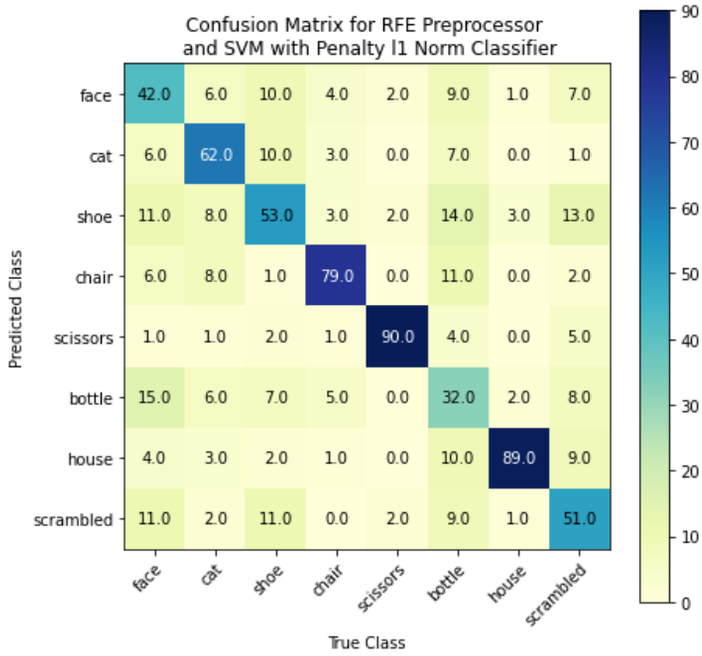

Fig. 40. Confusion chart for RFE and SWM with Penalty l1 Norm Classifier for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 43.86.

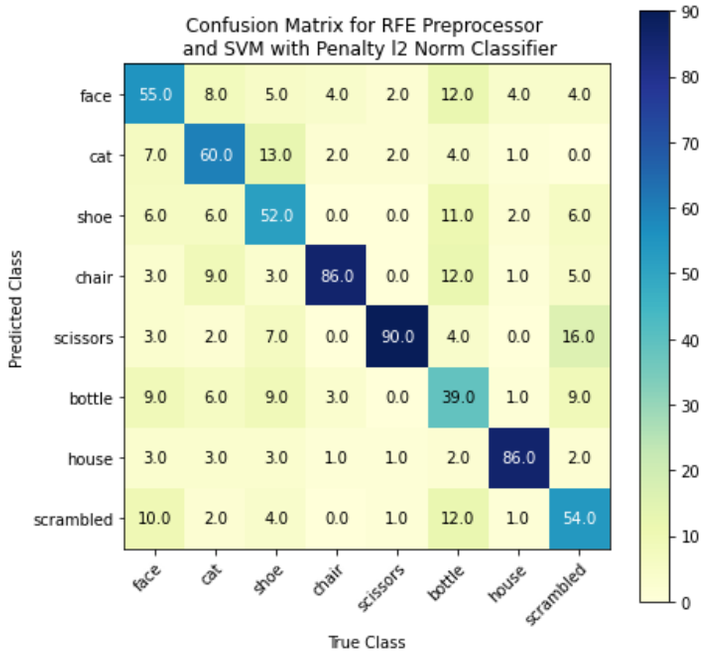

Fig. 41. Confusion chart for RFE and SWM with Penalty l2 Norm Classifier for the data over 12-fold cross validation. The classes are given in the diagonal axis and their predictions are given on the vertical axis. Significance value of this result for the null hypothesis of a dummy classifier is less than 10^{-325} with an effect size of 46.48.

CODE APPENDIX

A. Importing Downloaded Dataset to Python and Visulizing Subject 2 Mask Code

```
import os
import numpy as np
import pandas as pd

from nilearn.image import index_img
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker
'''
Write the filepath here by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')

haxbydir = 'C:/Users/mehme/EEE482_data/'
haxbysubjects = os.listdir(haxbydir)
timehorizon = np.arange(0,302.5,2.5)

#reference arrays for dictionaryies
haxbysubjects = haxbysubjects[4:10]
inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']


#Parsing directory into dictionary for easier access
haxby = {}
for i in haxbysubjects:
    directory = os.listdir(haxbydir+i)
    haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                'fun': os.listdir(haxbydir + i + '/'+directory[1])}
for i in (haxbysubjects):
    haxby[i]['data'] = {}
    for j in range(len(haxby[i]['fun'])):
        if haxby[i]['fun'][j].find('.tsv') == -1:
            continue
        else:
            behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
            conditions = np.array(['rest']*121)
            for abc in range(8):
                conditions[(((12+36*abc)<timehorizon) & (timehorizon <= (12 + 36*abc + 22.5)))] =
                behavior ['trial_type'][abc*12]
            haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

    for j in range(len(haxby[i]['fun'])):
        if haxby[i]['fun'][j].find('.tsv') == -1:
            target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j]) #index_img(haxbydir + i
            + '/'+'func/'+haxby[i]['fun'][j], haxby[i]['data'][str(int((j+2)/2))+'_behavioral'])
            mask = masking.compute_brain_mask(target_img, threshold=0.5,
            connected=True, opening=2, verbose=0)
            masker = NiftiMasker(mask_img=mask,
                     smoothing_fwhm=4, standardize=True,
                     memory="nilearn_cache", memory_level=1)
            haxby[i]['data'][str(int((j+1)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
haxby.pop('sub-5')
haxbysubjects.pop(4)

'''
Now visualise the mask in Haxby dataset on anatomical barin image in the same dataset.
'''
from nilearn import plotting
for i in haxbysubjects:
    plotting.plot_img(haxby[i]['anat'])
    plotting.plot_roi(haxby[i]['mask'], bg_img=haxby[i]['anat'], cmap = 'Paired')
```

B. Coefficients and Brain Voxels Illustrator Code

```
import os
import numpy as np
import pandas as pd
from nilearn import plotting
from nilearn.image import index_img
from nilearn.image import load_img
```

```
from nilearn import masking
from nilearn.input_data import NiftiMasker




'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = 'C:/Users/mehme/EEE482_data/'
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+
                    '/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <=
                    (15 + 36*abc + 20)))] = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j])
                #index_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j],
                haxby[i]['data'][str(int((j+2)/2))+'_behavioral'])
                mask = masking.compute_brain_mask(target_img,
                threshold=0.5, connected=True,
                opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                          smoothing_fwhm=4, standardize=True,
                          memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.model_selection import cross_validate
from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut

fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)
```

```
from sklearn.feature_selection import RFE
feature_selection = RFE(LinearSVC(), 50, step=0.25) #LinearSVC(), 50, step=0.25
rfe_svc = Pipeline([('rfe', feature_selection), ('svc', LinearSVC())])
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label = np.concatenate((session_label, group2))
condition_mask = (conditions == 'face') | (conditions == 'cat') #conditions != 'rest'
conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]
resting = fmri_masked[~condition_mask]
resting = np.mean(resting, axis = 0)
fmri_masked2 = fmri_masked2 - resting

fitted_pipeline = cross_validate(rfe_svc, fmri_masked2, conditions2,cv=cv,
                                 groups=session_label2, return_estimator=True)
print("\n\n\n\n\nRFE+SVC test score: {:.3f}".format(fitted_pipeline["test_score"].mean()))


second_pipeline = fitted_pipeline["estimator"][0]
svc_coef = second_pipeline.named_steps['svc'].coef_
print("After feature selection with RFE, the SVC is trained only on {} features".format(
    svc_coef.shape[1]))

# We invert the feature selection step to put these coefs in the right 2D place
full_coef = second_pipeline.named_steps['rfe'].inverse_transform(svc_coef)

print("After inverting feature selection, we have {} features back".format(
    full_coef.shape[1]))

# We apply the inverse of masking on these to make a 4D image that we can plot
from nilearn.plotting import plot_stat_map
weight_img = masker.inverse_transform(full_coef)
plot_stat_map(weight_img, title='RFE+SVC weights')

import os
import numpy as np
import pandas as pd
#from nilearn import plotting
#from nilearn.image import index_img
from nilearn.image import load_img
#from nilearn import masking
from nilearn.input_data import NiftiMasker




'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser(a):
    haxbydir = 'C:/Users/mehme/EEE482_data/'
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
```

```
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+
                    directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((a + 36*abc)<timehorizon) & (timehorizon <= (a + 36*abc + 20)))]
                    = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j]) #index_img(haxbydir +
                i + '/'+'func/'+haxby[i]['fun'][j], haxby[i]['data'][str(int((j+2)/2))+'_behavioral'])
                mask = haxby['sub-2']['mask']
                masker = NiftiMasker(mask_img=mask,
                          smoothing_fwhm=4, standardize=True,
                          memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker
#from sklearn.pipeline import Pipeline
#from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.model_selection import cross_validate
from sklearn.svm import LinearSVC #, SVC
from sklearn.model_selection import LeaveOneGroupOut
#from sklearn.feature_selection import RFE


haxby, haxbysubjects, inputs, masker = parser(15)



fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)


#feature_selection = RFE(LinearSVC(), 50, step=0.25) #LinearSVC(), 50, step=0.25
#rfe_svc = Pipeline([('rfe', feature_selection), ('svc', LinearSVC())])
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))
condition_mask = (conditions  == 'face') | (conditions  == 'cat') #conditions != 'rest'
conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]

fitted_pipeline = cross_validate(LinearSVC(), fmri_masked2, conditions2,
cv=cv,groups=session_label2, return_estimator=True)
print("\n\n\n\nRFE+SVC test score: {:.3f}".format(fitted_pipeline["test_score"].mean()))



second_pipeline = fitted_pipeline["estimator"][0]
full_coef = second_pipeline.coef_
from nilearn.plotting import plot_stat_map
weight_img = masker.inverse_transform(full_coef)
```

```
plot_stat_map(weight_img, title='VTMASK+SVC weights')


import os
import numpy as np
import pandas as pd
from nilearn import plotting
from nilearn.image import index_img
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker




'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = 'C:/Users/mehme/EEE482_data/'
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'
                    +(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((14.5 + 36*abc)<timehorizon) & (timehorizon <= (14.5 + 36*abc + 22.5)))] =
                    behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'
                +haxby[i]['fun'][j]) #index_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], haxby[i]['data'][str(
                mask = masking.compute_brain_mask(target_img, threshold=0.5, connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                          smoothing_fwhm=4, standardize=True,
                          memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.model_selection import cross_validate
```

```
from sklearn.svm import LinearSVC
from sklearn.model_selection import LeaveOneGroupOut

fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

feature_selection = SelectPercentile(f_classif, percentile=0.15)
anova_svc = Pipeline([('anova', feature_selection), ('svc', LinearSVC())])
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))
condition_mask = (conditions  == 'face') | (conditions  == 'cat') #conditions != 'rest'
conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]

fitted_pipeline = cross_validate(anova_svc, fmri_masked2, conditions2,
cv=cv,groups=session_label2, return_estimator=True)
print("ANOVA+SVC test score: {:.3f}".format(fitted_pipeline["test_score"].mean()))


second_pipeline = fitted_pipeline["estimator"][0]
svc_coef = second_pipeline.named_steps['svc'].coef_
print("After feature selection with RFE, the SVC is trained only on {} features".format(
    svc_coef.shape[1]))

# We invert the feature selection step to put these coefs in the right 2D place
full_coef = second_pipeline.named_steps['anova'].inverse_transform(svc_coef)

print("After inverting feature selection, we have {} features back".format(
    full_coef.shape[1]))

# We apply the inverse of masking on these to make a 4D image that we can plot
from nilearn.plotting import plot_stat_map
weight_img = masker.inverse_transform(full_coef)
plot_stat_map(weight_img, title='ANOVA+SVC weights')
```

C. Regularization Parameter Validation

```
import os
import numpy as np
import pandas as pd
from nilearn import plotting
from nilearn.image import index_img
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python

"""
Anova, SVM l1, Regularization Parameter
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
```

```
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i
                   + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <=
                    (15 + 36*abc + 20)))] = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+
                'func/'+haxby[i]['fun'][j]) #index_img(haxbydir + i + '/'+'func/'+
                haxby[i]['fun'][j], haxby[i]['data'][str(int((j+2)/2))+'_behavioral'])
                mask = masking.compute_brain_mask(target_img,
                threshold=0.5,
                connected=True,
                opening=2,
                verbose=0)
                masker = NiftiMasker(mask_img=mask,
                          smoothing_fwhm=4, standardize=True,

                          memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.model_selection import cross_validate
from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.feature_selection import SelectPercentile, f_classif


scores_SVM_l1_ANOVA = []

for j in ( 0.00464 , 0.01 , 0.0216 , 0.0464 , 0.1 , 0.216 , 0.464 ,  1 , 2.16 , 2.16 , 4.64 , 10):


    fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
    conditions = haxby['sub-2']['data']['1_behavioral']
    session_label = np.zeros(121)

    from sklearn.feature_selection import RFE
    feature_selection = SelectPercentile(f_classif, percentile=0.15)
    # method
    rfe_svc = OneVsRestClassifier( Pipeline([('Anova', feature_selection),
    ('svc', LinearSVC(penalty='l1', C=j, dual=False))]) )
    fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
    cv = LeaveOneGroupOut()

    for i in range(2,13):
        fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
```

```
        fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
        conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
        group2 = np.ones(121)*(i-1)
        session_label  = np.concatenate((session_label, group2))

    condition_mask = conditions != 'rest'

    conditions2 = conditions[condition_mask]
    session_label2 = session_label[condition_mask]
    fmri_masked2 = fmri_masked[condition_mask]


    from sklearn.metrics import confusion_matrix
    from nilearn.plotting import plot_matrix, show



    Confused = np.zeros((8, 8))

    for i in range(12):


        print("\n\n\n")
        print(i) # for understanding run time

        rfe_svc.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

        y_pred_ovo = rfe_svc.predict(fmri_masked2[session_label2 == i])

        temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

        Confused += temp_matrix


    succes_count = 0

    for i in range(8):

        succes_count += Confused[i, i]


    scores_SVM_l1_ANOVA.append( succes_count / 7.68)



plt.figure(figsize=(8, 5))
plt.plot( (1 / np.array(( 0.00464 , 0.01 , 0.0216 ,
0.0464 , 0.1 , 0.216 , 0.464 ,
1 , 2.16 , 2.16 , 4.64 , 10)) ) ,
(np.array(scores_SVM_l1_ANOVA))
)
plt.xlabel('The regulization parameter')
plt.ylabel('Success rate of the pipeline.')
plt.xscale('log')
plt.title("Success Rate vs Regularization Parameter for ANOVA \n and SVM with Penalty l1 Norm Classifier")
plt.grid()
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python
"""
Anova, LR l2, Regularization Parameter
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')
```

```
def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+
                    (os.listdir(haxbydir+i+'/'+directory[0])[0]),

                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <=
                    (15 + 36*abc + 20)))] =
                    behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i
                + '/'+'func/'+haxby[i]['fun'][j])

                ['data'][str(int((j+2)/2))+'_behavioral'])
                mask = masking.compute_brain_mask(target_img, threshold=0.5,
                connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                          smoothing_fwhm=4, standardize=True,
                          memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectPercentile, f_classif

from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression


scores_LR_l2_ANOVA = []

for j in ( 0.00464 , 0.01 , 0.0216 , 0.0464 , 0.1 , 0.216 , 0.464 ,  1 , 2.16, 4.64 , 10):


    fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
    conditions = haxby['sub-2']['data']['1_behavioral']
    session_label = np.zeros(121)

    feature_selection = SelectPercentile(f_classif, percentile=0.15) # method
    rfe_svc = OneVsRestClassifier( Pipeline([('Anova', feature_selection), ('lr',
    LogisticRegression(penalty =
    'l2', C = j ,  solver = 'liblinear'))]) )
    fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
```

```
    cv = LeaveOneGroupOut()

    for i in range(2,13):
        fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
        fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
        conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
        group2 = np.ones(121)*(i-1)
        session_label  = np.concatenate((session_label, group2))

    condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat')
    #conditions != 'rest'

    conditions2 = conditions[condition_mask]
    session_label2 = session_label[condition_mask]
    fmri_masked2 = fmri_masked[condition_mask]


    from sklearn.metrics import confusion_matrix



    Confused = np.zeros((8, 8))

    for i in range(12):


        print("\n\n\n")
        print(i) # for understanding run time

        rfe_svc.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

        y_pred_ovo = rfe_svc.predict(fmri_masked2[session_label2 == i])

        temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

        Confused += temp_matrix


    succes_count = 0

    for i in range(8):

        succes_count += Confused[i, i]


    scores_LR_l2_ANOVA.append( succes_count / 7.68)



plt.figure(figsize=(8, 5))
plt.plot( (1 / np.array(( 0.00464 , 0.01 , 0.0216 , 0.0464 , 0.1 , 0.216 , 0.464 ,
1 , 2.16, 4.64 , 10)) ) , (np.array(scores_LR_l2_ANOVA)) )
plt.xlabel('The regulization parameter')
plt.ylabel('Success rate of the pipeline.')
plt.xscale('log')
plt.title("Success Rate vs Regularization Parameter for ANOVA \n and
Logistic Regression with Penalty l2 Norm Classifier")
plt.grid()
```

```
import os
import numpy as np
import pandas as pd
from nilearn import plotting
from nilearn.image import index_img
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python
"""
Anova, SVM l1, Regularization Parameter
"""

'''
Write the filepaths by changing \ to /
'''
```

```
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')


def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+
                    directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <= (15 + 36*abc
                    + 20)))]
                    = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j]) #index_img(haxbydir + i + '/'+'
                mask = masking.compute_brain_mask(target_img, threshold=0.5,
                connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                          smoothing_fwhm=4, standardize=True,
                          memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.model_selection import cross_validate
from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.feature_selection import SelectPercentile, f_classif


scores_SVM_l1_ANOVA = []

for j in ( 0.00464 , 0.01 , 0.0216 , 0.0464 , 0.1 , 0.216 , 0.464 ,  1 , 2.16 , 2.16 , 4.64 , 10):


    fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
    conditions = haxby['sub-2']['data']['1_behavioral']
    session_label = np.zeros(121)

    from sklearn.feature_selection import RFE
    feature_selection = SelectPercentile(f_classif, percentile=0.15) # method
    rfe_svc = OneVsRestClassifier( Pipeline([('Anova', feature_selection),
    ('svc', LinearSVC(penalty='l1', C=j, dual=False))]) )
```

```
    fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
    cv = LeaveOneGroupOut()

    for i in range(2,13):
        fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
        fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
        conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
        group2 = np.ones(121)*(i-1)
        session_label  = np.concatenate((session_label, group2))

    condition_mask = conditions != 'rest'
    #(conditions  == 'face') | (conditions  == 'cat') #conditions != 'rest'

    conditions2 = conditions[condition_mask]
    session_label2 = session_label[condition_mask]
    fmri_masked2 = fmri_masked[condition_mask]


    from sklearn.metrics import confusion_matrix
    from nilearn.plotting import plot_matrix, show



    Confused = np.zeros((8, 8))

    for i in range(12):


        print("\n\n\n")
        print(i) # for understanding run time

        rfe_svc.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

        y_pred_ovo = rfe_svc.predict(fmri_masked2[session_label2 == i])

        temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

        Confused += temp_matrix


    succes_count = 0

    for i in range(8):

        succes_count += Confused[i, i]


    scores_SVM_l1_ANOVA.append( succes_count / 7.68)



plt.figure(figsize=(8, 5))
plt.plot( (1 / np.array(( 0.00464 , 0.01 , 0.0216 , 0.0464 , 0.1 ,
0.216 , 0.464 ,  1 , 2.16 , 2.16 , 4.64 , 10)) ) ,
(np.array(scores_SVM_l1_ANOVA)) )
plt.xlabel('The regulization parameter')
plt.ylabel('Success rate of the pipeline.')
plt.xscale('log')
plt.title("Success Rate vs Regularization Parameter for ANOVA \n and SVM with Penalty l1 Norm Classifier")
plt.grid()
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python

"""
Anova, SVM l2, Regularization Parameter
"""

'''
Write the filepaths by changing \ to /
```

```
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+
                    '/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) &
                    (timehorizon <= (15 + 36*abc + 20)))] = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]
                ['fun'][j]) #index_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j],
                haxby[i]['data'][str(int((j+2)/2))+'_behavioral'])
                mask = masking.compute_brain_mask(target_img, threshold=0.5,
                connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                          smoothing_fwhm=4, standardize=True,
                          memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier


scores_SVM_l2_ANOVA = []

for j in ( 0.000216 , 0.00464 , 0.001 , 0.00216 , 0.00464 ,
0.01 , 0.0216 , 0.0464 , 0.1 , 0.216 , 0.464 ,  1 , 2.16):


    fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
    conditions = haxby['sub-2']['data']['1_behavioral']
    session_label = np.zeros(121)

    feature_selection = SelectPercentile(f_classif,
    percentile=0.15) # method
    rfe_svc = OneVsRestClassifier( Pipeline([('Anova', feature_selection),
```

```
    ('svc', LinearSVC(penalty='l2', C=j, dual=False))]) )
    fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
    cv = LeaveOneGroupOut()

    for i in range(2,13):
        fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
        fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
        conditions = np.concatenate((conditions,
        haxby['sub-2']['data'][str(i) + '_behavioral']))
        group2 = np.ones(121)*(i-1)
        session_label  = np.concatenate((session_label, group2))

    condition_mask = conditions != 'rest' #(conditions  == 'face') |(conditions  == 'cat')
    #conditions != 'rest'

    conditions2 = conditions[condition_mask]
    session_label2 = session_label[condition_mask]
    fmri_masked2 = fmri_masked[condition_mask]


    from sklearn.metrics import confusion_matrix



    Confused = np.zeros((8, 8))

    for i in range(12):


        print("\n\n\n")
        print(i) # for understanding run time

        rfe_svc.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

        y_pred_ovo = rfe_svc.predict(fmri_masked2[session_label2 == i])

        temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

        Confused += temp_matrix


    succes_count = 0

    for i in range(8):

        succes_count += Confused[i, i]


    scores_SVM_l2_ANOVA.append( succes_count / 7.68)



plt.figure(figsize=(8, 5))
plt.plot( (1 / np.array(( 0.000216 , 0.000464 , 0.001 , 0.00216 , 0.00464 ,
0.01 , 0.0216 , 0.0464 , 0.1 , 0.216 ,
0.464 ,  1 , 2.16)) ) ,
(np.array(scores_SVM_l2_ANOVA)) )
plt.xlabel('The regulization parameter')
plt.ylabel('Success rate of the pipeline.')
plt.xscale('log')
plt.title("Success Rate vs Regularization Parameter for ANOVA \n and SVM with Penalty l2 Norm Classifier")
plt.grid()
```

```
import os
import numpy as np
import pandas as pd
from nilearn import plotting
from nilearn.image import index_img
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker



"""
MASK, LR l1 norm, parameter tuning
```

```
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+
                    '/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <=
                    (15 + 36*abc + 20)))] = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j]) #
                index_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], haxby[i]['data'][str(int((j+2)/2))+'_behavi
                mask = haxby['sub-2']['mask']
                #masking.compute_brain_mask(target_img, threshold=0.5, connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                          smoothing_fwhm=4, standardize=True,
                          memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt # this brings nice loking plots to python
from sklearn.linear_model import LogisticRegression

scores_LR_l1_MASK = []

for j in ( 0.1 , 0.216 , 0.464 ,  1 , 2.16, 4.64 , 10 , 21.6, 46.4 ,
100 , 216, 464 , 1000 , 2160, 4640 , 10000):


    fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
    conditions = haxby['sub-2']['data']['1_behavioral']
    session_label = np.zeros(121)
```

```
    #feature_selection = RFE(LinearSVC(penalty='l2', C=j , dual=False), 50, step=0.25)
    #LinearSVC(), 50, step=0.25
    classifier =OneVsRestClassifier( LogisticRegression(penalty = 'l1', C = j ,
    solver = 'liblinear'))
    #rfe_svc = OneVsRestClassifier( Pipeline([('rfe', feature_selection),
    ('svc', LinearSVC())]) )
    fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
    cv = LeaveOneGroupOut()

    for i in range(2,13):
        fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
        fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
        conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
        group2 = np.ones(121)*(i-1)
        session_label  = np.concatenate((session_label, group2))

    condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat')
    #conditions != 'rest'

    conditions2 = conditions[condition_mask]
    session_label2 = session_label[condition_mask]
    fmri_masked2 = fmri_masked[condition_mask]




    Confused = np.zeros((8, 8))

    for i in range(12):


        print("\n\n\n")
        print(i) # for understanding run time

        classifier.fit(fmri_masked2[session_label2 != i] ,
        conditions2[session_label2 != i] )

        y_pred_ovo = classifier.predict(fmri_masked2[session_label2 == i])

        temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

        Confused += temp_matrix


    succes_count = 0

    for i in range(8):

        succes_count += Confused[i, i]


    scores_LR_l1_MASK.append( succes_count / 7.68)



plt.figure(figsize=(8, 5))
plt.plot( (1 / np.array(( 0.1 , 0.216 , 0.464 ,  1 , 2.16, 4.64 ,
10 , 21.6, 46.4 , 100 , 216, 464 , 1000 , 2160, 4640 , 10000)) ) ,
(np.array(scores_LR_l1_MASK)) )
plt.xlabel('The regulization parameter')
plt.ylabel('Success rate of the pipeline.')
plt.xscale('log')
plt.title("Success Rate vs Regularization Parameter for Ventral Temporal\nCortex Mask and Logistic Regression with P
plt.grid()
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn.input_data import NiftiMasker



"""
```

```
MASK, LR l2 norm, parameter tuning
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+
                    (os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+
                haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <=
                    (15 + 36*abc + 20)))] = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j])
                #index_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j],
                #haxby[i]['data'][str(int((j+2)/2))+'_behavioral'])
                mask = haxby['sub-2']['mask']
                #masking.compute_brain_mask(target_img,
                #threshold=0.5, connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                          smoothing_fwhm=4, standardize=True,
                          memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt # this brings nice loking plots to python
from sklearn.linear_model import LogisticRegression

scores_LR_l2_MASK = []

for j in ( 0.001 , 0.00216 , 0.00464 , 0.01 , 0.0216 , 0.0464 ,
0.1 , 0.216 , 0.464 ,  1 , 2.16, 4.64 , 10):


    fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
```

```
    conditions = haxby['sub-2']['data']['1_behavioral']
    session_label = np.zeros(121)


    #feature_selection = RFE(LinearSVC(penalty='l2', C=j , dual=False), 50, step=0.25)
    #LinearSVC(), 50, step=0.25
    classifier =OneVsRestClassifier( LogisticRegression(penalty = 'l2', C = j ,
    solver = 'liblinear'))
    #rfe_svc = OneVsRestClassifier( Pipeline([('rfe', feature_selection), ('svc', LinearSVC())]) )
    fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
    cv = LeaveOneGroupOut()

    for i in range(2,13):
        fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
        fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
        conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
        group2 = np.ones(121)*(i-1)
        session_label  = np.concatenate((session_label, group2))

    condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat')
    #conditions != 'rest'

    conditions2 = conditions[condition_mask]
    session_label2 = session_label[condition_mask]
    fmri_masked2 = fmri_masked[condition_mask]




    Confused = np.zeros((8, 8))

    for i in range(12):


        print("\n\n\n")
        print(i) # for understanding run time

        classifier.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

        y_pred_ovo = classifier.predict(fmri_masked2[session_label2 == i])

        temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

        Confused += temp_matrix


    succes_count = 0

    for i in range(8):

        succes_count += Confused[i, i]


    scores_LR_l2_MASK.append( succes_count / 7.68)



plt.figure(figsize=(8, 5))
plt.plot( (1 / np.array(( 0.001 , 0.00216 , 0.00464 , 0.01 , 0.0216 ,
0.0464 , 0.1 , 0.216 , 0.464 ,  1 , 2.16, 4.64 , 10)) ) ,
(np.array(scores_LR_l2_MASK)) )
plt.xlabel('The␣regulization␣parameter')
plt.ylabel('Success␣rate␣of␣the␣pipeline.')
plt.xscale('log')
plt.title("Success␣Rate␣vs␣Regularization␣Parameter␣for␣Ventral␣Temporal\nCortex␣Mask␣and␣Logistic␣Regression␣with␣P
plt.grid()
```

```
import os
import numpy as np
import pandas as pd
from nilearn import plotting
from nilearn.image import index_img
from nilearn.image import load_img
from nilearn import masking
```

```
from nilearn.input_data import NiftiMasker


"""
MASK, SVM l1 norm, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+
                    (os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) &
                    (timehorizon <= (15 + 36*abc + 20)))] = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j])
                #index_img(haxbydir + i + '/'+'func/'+
                haxby[i]['fun'][j], haxby[i]['data'][str(int((j+2)/2))+'_behavioral'])
                mask = haxby['sub-2']['mask']
                #masking.compute_brain_mask(target_img, threshold=0.5,
                connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                         smoothing_fwhm=4, standardize=True,
                         memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =
                masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.model_selection import cross_validate
from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt # this brings nice loking plots to python
```

```
scores_SVM_l1 = []

for j in ( 0.0216 , 0.0464 , 0.1 , 0.216 , 0.464 ,  1 , 2.16 , 4.64, 10):


    fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
    conditions = haxby['sub-2']['data']['1_behavioral']
    session_label = np.zeros(121)

    from sklearn.feature_selection import RFE
    feature_selection = RFE(LinearSVC(penalty='l1', C=j , dual=False), 50, step=0.25)
    #LinearSVC(), 50, step=0.25
    classifier =OneVsRestClassifier( LinearSVC(penalty='l1', C=j , dual=False))
    #rfe_svc = OneVsRestClassifier( Pipeline([('rfe', feature_selection),
    ('svc', LinearSVC())]) )
    fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
    cv = LeaveOneGroupOut()

    for i in range(2,13):
        fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
        fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
        conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
        group2 = np.ones(121)*(i-1)
        session_label  = np.concatenate((session_label, group2))

    condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat')
    #conditions != 'rest'

    conditions2 = conditions[condition_mask]
    session_label2 = session_label[condition_mask]
    fmri_masked2 = fmri_masked[condition_mask]




    Confused = np.zeros((8, 8))

    for i in range(12):


        print("\n\n\n")
        print(i) # for understanding run time

        classifier.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

        y_pred_ovo = classifier.predict(fmri_masked2[session_label2 == i])

        temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

        Confused += temp_matrix


    succes_count = 0

    for i in range(8):

        succes_count += Confused[i, i]


    scores_SVM_l1.append( succes_count / 7.68)




plt.figure(figsize=(8, 5))
plt.plot( (1 / np.array(( 0.0216 , 0.0464 , 0.1 , 0.216 ,
0.464 ,  1 , 2.16 , 4.64, 10)) ) ,
(np.array(scores_SVM_l1)) )
plt.xlabel('The␣regulization␣parameter')
plt.ylabel('Success␣rate␣of␣the␣pipeline.')
plt.xscale('log')
plt.title("Success␣Rate␣vs␣Regularization␣Parameter␣for␣Ventral␣Temporal\nCortex␣Mask␣and␣SVM␣with␣Penalty␣l1␣Norm␣C
```

```
plt.grid()
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn.input_data import NiftiMasker



"""
MASK, SVM l2 norm, parameter tuning
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+
                    '/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <= (15 + 36*abc + 20)))]
                    = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j])
                #index_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], haxby[i]['data'][str(int((j+2)/2))+'_behav
                mask = haxby['sub-2']['mask']
                #masking.compute_brain_mask(target_img, threshold=0.5,
                #connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                           smoothing_fwhm=4, standardize=True,
                           memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
```

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt # this brings nice loking plots to python

scores_SVM_l2 = []

for j in ( 0.000216 , 0.00464 , 0.001 , 0.00216 , 0.00464 , 0.01 , 0.0216 ,
0.0464 , 0.1 , 0.216 , 0.464 , 1 , 2.16):


    fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
    conditions = haxby['sub-2']['data']['1_behavioral']
    session_label = np.zeros(121)

    from sklearn.feature_selection import RFE
    feature_selection = RFE(LinearSVC(penalty='l2', C=j , dual=False), 50, step=0.25)
    #LinearSVC(), 50, step=0.25
    classifier =OneVsRestClassifier( LinearSVC(penalty='l2', C=j , dual=False))
    #rfe_svc = OneVsRestClassifier( Pipeline([('rfe', feature_selection), ('svc',
    LinearSVC())]) )
    fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
    cv = LeaveOneGroupOut()

    for i in range(2,13):
        fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
        fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
        conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
        group2 = np.ones(121)*(i-1)
        session_label  = np.concatenate((session_label, group2))

    condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat')
    #conditions != 'rest'

    conditions2 = conditions[condition_mask]
    session_label2 = session_label[condition_mask]
    fmri_masked2 = fmri_masked[condition_mask]




    Confused = np.zeros((8, 8))

    for i in range(12):


        print("\n\n\n")
        print(i) # for understanding run time

        classifier.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

        y_pred_ovo = classifier.predict(fmri_masked2[session_label2 == i])

        temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

        Confused += temp_matrix


    succes_count = 0

    for i in range(8):

        succes_count += Confused[i, i]


    scores_SVM_l2.append( succes_count / 7.68)



plt.figure(figsize=(8, 5))
plt.plot( (1 / np.array(( 0.000216 , 0.000464 , 0.001 , 0.00216 ,
0.00464 , 0.01 , 0.0216 , 0.0464 , 0.1 , 0.216 , 0.464 ,
1 , 2.16)) ) ,
(np.array(scores_SVM_l2)) )
plt.xlabel('The regulization parameter')
plt.ylabel('Success rate of the pipeline.')
plt.xscale('log')
```

```
plt.title("Success Rate vs Regularization Parameter for Ventral Temporal\nCortex Mask and SVM with Penalty 12 Norm C
plt.grid()
```

D. Confusion Matrix Drawing

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt
# this brings nice loking plots to python

"""
Anova, LDA, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i +
                    '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+
                'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) &
                    (timehorizon <= (15 + 36*abc + 20)))] =
                    behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j])

                #index_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j],
                #haxby[i]['data'][str(int((j+2)/2))+'_behavioral'])
                mask = masking.compute_brain_mask(target_img, threshold=0.5,
                connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                          smoothing_fwhm=4, standardize=True,
                          memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)
```

```
    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis


fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

feature_selection = SelectPercentile(f_classif, percentile=0.15)
rfe_svc = OneVsRestClassifier( Pipeline([('Anova', feature_selection),
('svc', LinearDiscriminantAnalysis() )]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions,
    haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat')
#conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]


from sklearn.metrics import confusion_matrix



Confused = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    rfe_svc.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = rfe_svc.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused += temp_matrix

"""
The code for drawing a nice plot.
"""


fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused, cmap='YlGnBu' )

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")
```

```
plt.title("Confusion Matrix for Anova Preprocessor and \n LDA Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')


# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)

for i in range(8):
    for j in range(8):
        if Confused[i, j] > 50:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python

"""
Anova, Logistic Regresion l1, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'
                    +directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'
                +'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <=
                    (15 + 36*abc + 20)))] = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j])
                #index_img(haxbydir + i
                + '/'+'func/'+haxby[i]['fun'][j], haxby[i]['data']
                [str(int((j+2)/2))+'_behavioral'])
                mask = masking.compute_brain_mask(target_img, threshold=0.5,
                connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
```

```
                        smoothing_fwhm=4, standardize=True,
                        memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =
                masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression


fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

feature_selection = SelectPercentile(f_classif, percentile=0.15)
rfe_svc = OneVsRestClassifier( Pipeline([('Anova', feature_selection),
('svc', LogisticRegression(penalty = 'l1', C = 0.33 , solver = 'liblinear') )]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') |
(conditions  == 'cat') #conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]


from sklearn.metrics import confusion_matrix



Confused_ANOVA_LR_l1 = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    rfe_svc.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = rfe_svc.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused_ANOVA_LR_l1 += temp_matrix

"""
The code for drawing a nice plot.
"""

fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused_ANOVA_LR_l1, cmap='YlGnBu' )

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
# ... and label them with the respective list entries
```

```
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for Anova Preprocessor and \n Logistic Regresion with Penalty l1 Norm Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')


# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)

for i in range(8):
    for j in range(8):
        if Confused_ANOVA_LR_l1[i, j] > 50:
            text = ax.text(j, i, Confused_ANOVA_LR_l1[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused_ANOVA_LR_l1[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python

"""
Anova, Logistic Regresion l1, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) &
                    (timehorizon <= (15 + 36*abc + 20)))] = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j])
                #index_img(haxbydir +     i + '/'+'func/'+haxby[i]['fun'][j],
                haxby[i]['data'][str(int((j+2)/2))+'_behavioral'])
```

```
                mask = masking.compute_brain_mask(target_img, threshold=0.5,
                connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                         smoothing_fwhm=4, standardize=True,
                         memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression


fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

feature_selection = SelectPercentile(f_classif, percentile=0.15)
rfe_svc = OneVsRestClassifier( Pipeline([('Anova', feature_selection),
('svc', LogisticRegression(penalty = 'l1', C = 0.33 , solver = 'liblinear') )]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat') #conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]


from sklearn.metrics import confusion_matrix



Confused_ANOVA_LR_l1 = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    rfe_svc.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = rfe_svc.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused_ANOVA_LR_l1 += temp_matrix

"""
The code for drawing a nice plot.
"""

fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused_ANOVA_LR_l1, cmap='YlGnBu' )

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
```

```
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for Anova Preprocessor and \n Logistic Regresion with Penalty l1 Norm Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')


# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)

for i in range(8):
    for j in range(8):
        if Confused_ANOVA_LR_l1[i, j] > 50:
            text = ax.text(j, i, Confused_ANOVA_LR_l1[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused_ANOVA_LR_l1[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python

"""
Anova, Logistic Regresion l2, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <= (15 + 36*abc + 20)))]
                    = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j])
                #index_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j],
```

```
                #haxby[i]['data'][str(int((j+2)/2))+'_behavioral'])
                mask = masking.compute_brain_mask(target_img, threshold=0.5,
                connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                         smoothing_fwhm=4, standardize=True,
                         memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression


fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

feature_selection = SelectPercentile(f_classif, percentile=0.15)
rfe_svc = OneVsRestClassifier( Pipeline([('Anova', feature_selection), ('svc',
LogisticRegression(penalty = 'l2', C = 0.1 ,
solver = 'liblinear') )]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat') #conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]


from sklearn.metrics import confusion_matrix



Confused_ANOVA_LR_l2 = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    rfe_svc.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = rfe_svc.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused_ANOVA_LR_l2 += temp_matrix

"""
The code for drawing a nice plot.
"""

fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused_ANOVA_LR_l2, cmap='YlGnBu' )

# We want to show all ticks...
```

```
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for Anova Preprocessor and \n Logistic Regresion with Penalty l2 Norm Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')


# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)

for i in range(8):
    for j in range(8):
        if Confused_ANOVA_LR_l2[i, j] > 50:
            text = ax.text(j, i, Confused_ANOVA_LR_l2[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused_ANOVA_LR_l2[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python

"""
Anova, SVM l1, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) &
                    (timehorizon <= (15 + 36*abc + 20)))] = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
```

```
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j])
                #index_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], haxby[i]['data'][str(int((j+2)/2))+'_behav
                mask = masking.compute_brain_mask(target_img, threshold=0.5,
                connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                          smoothing_fwhm=4, standardize=True,
                          memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier

fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

feature_selection = SelectPercentile(f_classif, percentile=0.15) # method
rfe_svc = OneVsRestClassifier( Pipeline([('Anova', feature_selection),
('svc', LinearSVC(penalty='l1', C = 0.1 , dual=False))]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat')
#conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]


from sklearn.metrics import confusion_matrix



Confused_ANOVA_SVM_l1 = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    rfe_svc.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = rfe_svc.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused_ANOVA_SVM_l1 += temp_matrix

"""
The code for drawing a nice plot.
"""

fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused_ANOVA_SVM_l1, cmap='YlGnBu' )

# We want to show all ticks...
```

```
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for Anova Preprocessor \n and SVM with Penalty l1 Norm Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')


# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)


for i in range(8):
    for j in range(8):
        if Confused_ANOVA_SVM_l1[i, j] > 50:
            text = ax.text(j, i, Confused_ANOVA_SVM_l1[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused_ANOVA_SVM_l1[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python

"""
Anova, SVM l2, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+
                    (os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) &
                    (timehorizon <= (15 + 36*abc + 20)))] = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions
```

```
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j])
                #index_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j],
                #haxby[i]['data'][str(int((j+2)/2))+'_behavioral'])
                mask = masking.compute_brain_mask(target_img,
                threshold=0.5, connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                         smoothing_fwhm=4, standardize=True,
                         memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier

fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

feature_selection = SelectPercentile(f_classif, percentile=0.15) # method
rfe_svc = OneVsRestClassifier( Pipeline([('Anova', feature_selection),
('svc', LinearSVC(penalty='l2', C = 0.0216 , dual=False))]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat')
#conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]


from sklearn.metrics import confusion_matrix



Confused_ANOVA_SVM_l2 = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    rfe_svc.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = rfe_svc.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused_ANOVA_SVM_l2 += temp_matrix

"""
The code for drawing a nice plot.
"""

fig, ax = plt.subplots(figsize=(7,7))
```

```
im = ax.imshow(Confused_ANOVA_SVM_l2, cmap='YlGnBu' )

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for Anova Preprocessor \n and SVM with Penalty l2 Norm Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')


# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)

for i in range(8):
    for j in range(8):
        if Confused_ANOVA_SVM_l2[i, j] > 50:
            text = ax.text(j, i, Confused_ANOVA_SVM_l2[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused_ANOVA_SVM_l2[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn.input_data import NiftiMasker



"""
MASK, LDA, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon
                    <= (15 + 36*abc + 20)))] = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions
```

```
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j])
                #index_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j],
                #haxby[i]['data'][str(int((j+2)/2))+'_behavioral'])
                mask = haxby['sub-2']['mask']
                #masking.compute_brain_mask(target_img, threshold=0.5, connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                         smoothing_fwhm=4, standardize=True,
                         memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()

from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis


fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)


#feature_selection = RFE(LinearDiscriminantAnalysis(), 50, step=0.25) #LinearSVC(), 50, step=0.25
classifier =OneVsRestClassifier(LinearDiscriminantAnalysis())
#rfe_svc = OneVsRestClassifier( Pipeline([('rfe', feature_selection), ('svc', LinearSVC())]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat') #conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]


from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt # this brings nice loking plots to python



Confused = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    classifier.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = classifier.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused += temp_matrix



fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused, cmap='YlGnBu' )

# We want to show all ticks...
```

```
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for Ventral Temporal
Cortex Mask \n and LDA Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')

# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)


for i in range(8):
    for j in range(8):
        if Confused[i, j] > 50:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn.input_data import NiftiMasker



"""
MASK, Logistic Regression l1, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) &
                    (timehorizon <= (15 + 36*abc + 20)))] = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
```

```
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j])
                #index_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j],
                #haxby[i]['data'][str(int((j+2)/2))+'_behavioral'])
                mask = haxby['sub-2']['mask']
                #masking.compute_brain_mask(target_img, threshold=0.5,
                #connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                         smoothing_fwhm=4, standardize=True,
                         memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression


fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

#feature_selection = RFE(LogisticRegression(penalty = 'l1', solver = 'liblinear'),360, step=0.2)
classifier =OneVsRestClassifier( LogisticRegression(penalty = 'l1', C = 2160 , solver = 'liblinear'))
#rfe_svc = OneVsRestClassifier( Pipeline([('rfe', feature_selection), ('svc', LinearSVC())]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat') #conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]


from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt # this brings nice loking plots to python



Confused_Mask_LR_l1 = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    classifier.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = classifier.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused_Mask_LR_l1 += temp_matrix


fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused_Mask_LR_l1, cmap='YlGnBu')

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
```

```
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for Ventral Temporal Cortex Mask \n and Logistic Regression , Penalty l1 Norm Classifier"
plt.xlabel('True Class')
plt.ylabel('Predicted Class')

# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)

for i in range(8):
    for j in range(8):
        if Confused_Mask_LR_l1[i, j] > 50:
            text = ax.text(j, i, Confused_Mask_LR_l1[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused_Mask_LR_l1[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn.input_data import NiftiMasker



"""
MASK, Logistic Regression l2, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+
                    '/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+
                'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) &
                    (timehorizon <= (15 + 36*abc + 20)))] =
                    behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j])
```

```
                #index_img(haxbydir + i + '/'+'func/'
                +haxby[i]['fun'][j], haxby[i]['data'][str(int((j+2)/2))+'_behavioral'])

                mask = haxby['sub-2']['mask']#masking.compute_brain_mask(target_img, threshold=0.5,
                connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                          smoothing_fwhm=4, standardize=True,
                          memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression


fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

#feature_selection = RFE(LogisticRegression(penalty = 'l1', solver = 'liblinear'),360, step=0.2)
classifier =OneVsRestClassifier( LogisticRegression(penalty = 'l2', C = 0.1 , solver = 'liblinear'))
#rfe_svc = OneVsRestClassifier( Pipeline([('rfe', feature_selection),
('svc', LinearSVC())]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat') #conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]


from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt # this brings nice loking plots to python



Confused_Mask_LR_l2 = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    classifier.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = classifier.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused_Mask_LR_l2 += temp_matrix


fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused_Mask_LR_l2, cmap='YlGnBu')

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
```

```
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for Ventral Temporal Cortex Mask \n and Logistic Regression with Penalty l2 Norm Classif
plt.xlabel('True Class')
plt.ylabel('Predicted Class')

# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)

for i in range(8):
    for j in range(8):
        if Confused_Mask_LR_l2[i, j] > 50:
            text = ax.text(j, i, Confused_Mask_LR_l2[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused_Mask_LR_l2[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn.input_data import NiftiMasker



"""
MASK, SVM l1 norm, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors'
    , 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) &
                    (timehorizon <= (15 + 36*abc + 20)))] =
                    behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'
                +haxby[i]['fun'][j]) #index_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], haxby[i]['data'][str(
```

```
                mask = haxby['sub-2']['mask']
                #masking.compute_brain_mask(target_img, threshold=0.5, connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                         smoothing_fwhm=4, standardize=True,
                         memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()



from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier

fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

from sklearn.feature_selection import RFE
feature_selection = RFE(LinearSVC(penalty='l1', C = 0.1 , dual=False), 50
, step=0.25) #LinearSVC(), 50, step=0.25
classifier =OneVsRestClassifier( LinearSVC(penalty='l1', C = 0.1 , dual=False))
#rfe_svc = OneVsRestClassifier( Pipeline([('rfe', feature_selection),
('svc', LinearSVC())]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat') #conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]


from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt # this brings nice loking plots to python



Confused_Mask_SVM_l1 = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    classifier.fit(fmri_masked2[session_label2 != i] ,
    conditions2[session_label2 != i] )

    y_pred_ovo = classifier.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused_Mask_SVM_l1 += temp_matrix



fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused_Mask_SVM_l1, cmap='YlGnBu' )

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
```

```
ax.set_yticks(np.arange(len(inputs)))
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for Ventral Temporal Cortex Mask \n and SVM with
Penalty l1 Norm Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')

# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)


for i in range(8):
    for j in range(8):
        if Confused_Mask_SVM_l1[i, j] > 50:
            text = ax.text(j, i, Confused_Mask_SVM_l1[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused_Mask_SVM_l1[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn.input_data import NiftiMasker



"""
MASK, SVM l2 norm, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+
                    (os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) &
                    (timehorizon <= (15 + 36*abc + 20)))] = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
```

```
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j]) #index_img(haxbydir + i + '/'+'
                mask = haxby['sub-2']['mask']
                #masking.compute_brain_mask(target_img, threshold=0.5, connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                         smoothing_fwhm=4, standardize=True,
                         memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()



from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier

fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

#feature_selection = RFE(LinearSVC(penalty='l2', C = 0.0085
, dual=False), 50, step=0.25) #LinearSVC(), 50, step=0.25
classifier =OneVsRestClassifier( LinearSVC(penalty='l2', C = 0.01 , dual=False))
#rfe_svc = OneVsRestClassifier( Pipeline([('rfe', feature_selection)
, ('svc', LinearSVC())]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat') #conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]


from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt # this brings nice loking plots to python



Confused_Mask_SVM_l2 = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    classifier.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = classifier.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused_Mask_SVM_l2 += temp_matrix



fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused_Mask_SVM_l2, cmap='YlGnBu' )

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
```

```
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for Ventral Temporal Cortex Mask \n and SVM with Penalty l2 Norm Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')


# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)

for i in range(8):
    for j in range(8):
        if Confused_Mask_SVM_l2[i, j] > 50:
            text = ax.text(j, i, Confused_Mask_SVM_l2[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused_Mask_SVM_l2[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python


"""
RFE, LDA, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir +
        i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) &
                    (timehorizon <= (15 + 36*abc + 20)))] = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
```

```
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j]) #index_img(haxbydir + i + '/'+'
                mask = masking.compute_brain_mask(target_img, threshold=0.5, connected=True, opening=2
                , verbose=0)
                masker = NiftiMasker(mask_img=mask,
                         smoothing_fwhm=4, standardize=True,
                         memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis


fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

from sklearn.feature_selection import RFE
feature_selection = RFE(LinearDiscriminantAnalysis(), 360, step=0.2)
rfe_lr = OneVsRestClassifier( Pipeline([('rfe', feature_selection),
('lda', LinearDiscriminantAnalysis())]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat') #conditions
!= 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]




from sklearn.metrics import confusion_matrix

Confused = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    rfe_lr.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = rfe_lr.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused += temp_matrix

"""
The code for drawing a nice plot.
"""

fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused, cmap='YlGnBu')
```

```
# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for RFE Preprocessor \n and LDA Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')


# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)

for i in range(8):
    for j in range(8):
        if Confused[i, j] > 50:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python


"""
RFE, Logistic Regresion l1, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <= (15 + 36*abc + 20)))]
                    = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions
```

```
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+
                haxby[i]['fun'][j]) #index_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], haxby[i]['data'][str(i
                mask = masking.compute_brain_mask(target_img,
                threshold=0.5, connected=True,
                opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                          smoothing_fwhm=4, standardize=True,
                          memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression


fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

from sklearn.feature_selection import RFE
feature_selection = RFE(LogisticRegression(penalty = 'l1', C = 1 , solver = 'liblinear'),360, step=0.2)
rfe_lr = OneVsRestClassifier( Pipeline([('rfe', feature_selection), ('lr',
LogisticRegression(penalty = 'l1', C = 1 , solver = 'liblinear'))]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat') #conditions
!= 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]




from sklearn.metrics import confusion_matrix

Confused = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    rfe_lr.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = rfe_lr.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused += temp_matrix

"""
The code for drawing a nice plot.
```

```
"""

fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused, cmap='YlGnBu')

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for RFE Preprocessor and\nLogistic Regression with
Penalty l1 Norm Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')


# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)

for i in range(8):
    for j in range(8):
        if Confused[i, j] > 50:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn.input_data import NiftiMasker



"""
MASK, Logistic Regression l2, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+
                haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
```

```
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <=
                    (15 + 36*abc + 20)))] = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j]) #index_img(haxbydir + i + '/'+'
                mask = haxby['sub-2']['mask']#masking.compute_brain_mask(target_img, threshold=0.5, connected=True,
                masker = NiftiMasker(mask_img=mask,
                         smoothing_fwhm=4, standardize=True,
                         memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression


fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

#feature_selection = RFE(LogisticRegression(penalty = 'l1', solver = 'liblinear'),360, step=0.2)
classifier =OneVsRestClassifier( LogisticRegression(penalty = 'l2', C = 0.1 , solver = 'liblinear'))
#rfe_svc = OneVsRestClassifier( Pipeline([('rfe', feature_selection), ('svc', LinearSVC())]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat') #conditions
!= 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]


from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt # this brings nice loking plots to python



Confused_Mask_LR_l2 = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    classifier.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = classifier.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused_Mask_LR_l2 += temp_matrix


fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused_Mask_LR_l2, cmap='YlGnBu')
```

```
# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for Ventral Temporal Cortex Mask \n and Logistic Regression with Penalty
l2 Norm Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')

# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)

for i in range(8):
    for j in range(8):
        if Confused_Mask_LR_l2[i, j] > 50:
            text = ax.text(j, i, Confused_Mask_LR_l2[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused_Mask_LR_l2[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn.input_data import NiftiMasker



"""
MASK, SVM l1 norm, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <= (15 + 36*abc + 20)))] = behavior ['tri
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
```

```
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j]) #index_img(haxbydir + i + '/'+'
                mask = haxby['sub-2']['mask']#masking.compute_brain_mask(target_img, threshold=0.5, connected=True,
                masker = NiftiMasker(mask_img=mask,
                         smoothing_fwhm=4, standardize=True,
                         memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()



from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier

fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

from sklearn.feature_selection import RFE
feature_selection = RFE(LinearSVC(penalty='l1', C = 0.1 , dual=False), 50, step=0.25) #LinearSVC(), 50,
step=0.25
classifier =OneVsRestClassifier( LinearSVC(penalty='l1',  C = 0.1 , dual=False))
#rfe_svc = OneVsRestClassifier( Pipeline([('rfe', feature_selection), ('svc', LinearSVC())]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat') #conditions
!= 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]


from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt # this brings nice loking plots to python



Confused_Mask_SVM_l1 = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    classifier.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = classifier.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused_Mask_SVM_l1 += temp_matrix



fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused_Mask_SVM_l1, cmap='YlGnBu' )

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
```

```
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for Ventral Temporal Cortex Mask \n and SVM with Penalty l1 Norm Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')

# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)


for i in range(8):
    for j in range(8):
        if Confused_Mask_SVM_l1[i, j] > 50:
            text = ax.text(j, i, Confused_Mask_SVM_l1[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused_Mask_SVM_l1[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn.input_data import NiftiMasker



"""
MASK, SVM l2 norm, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <= (15 + 36*abc + 20)))]
                    = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j]) #index_img(haxbydir + i + '/'+'
                mask = haxby['sub-2']['mask']#masking.compute_brain_mask(target_img, threshold=0.5, connected=True,
                masker = NiftiMasker(mask_img=mask,
```

```
                        smoothing_fwhm=4, standardize=True,
                        memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()



from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier

fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

#feature_selection = RFE(LinearSVC(penalty='l2', C = 0.0085 , dual=False), 50, step=0.25)
#LinearSVC(), 50, step=0.25
classifier =OneVsRestClassifier( LinearSVC(penalty='l2', C = 0.01 , dual=False))
#rfe_svc = OneVsRestClassifier( Pipeline([('rfe', feature_selection), ('svc', LinearSVC())]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat') #conditions
!= 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]


from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt # this brings nice loking plots to python



Confused_Mask_SVM_l2 = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    classifier.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = classifier.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused_Mask_SVM_l2 += temp_matrix



fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused_Mask_SVM_l2, cmap='YlGnBu' )

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)
```

```
# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for Ventral Temporal Cortex Mask \n and SVM with Penalty l2 Norm Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')


# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)

for i in range(8):
    for j in range(8):
        if Confused_Mask_SVM_l2[i, j] > 50:
            text = ax.text(j, i, Confused_Mask_SVM_l2[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused_Mask_SVM_l2[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python


"""
RFE, LDA, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <= (15 + 36*abc + 20)))]
                    = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j]) #index_img(haxbydir + i + '/'+'
                mask = masking.compute_brain_mask(target_img, threshold=0.5, connected=True,
                opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                         smoothing_fwhm=4, standardize=True,
```

```
                        memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis


fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

from sklearn.feature_selection import RFE
feature_selection = RFE(LinearDiscriminantAnalysis(), 360, step=0.2)
rfe_lr = OneVsRestClassifier( Pipeline([('rfe', feature_selection), ('lda', LinearDiscriminantAnalysis())]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat')
#conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]




from sklearn.metrics import confusion_matrix

Confused = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    rfe_lr.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = rfe_lr.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused += temp_matrix

"""
The code for drawing a nice plot.
"""

fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused, cmap='YlGnBu')

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)
```

```
# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for RFE Preprocessor \n and LDA Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')


# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)

for i in range(8):
    for j in range(8):
        if Confused[i, j] > 50:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python


"""
RFE, Logistic Regresion l1, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <= (15 + 36*abc + 20)))]
                    = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j]) #index_img(haxbydir + i + '/'+'
                mask = masking.compute_brain_mask(target_img, threshold=0.5,
                connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
```

```
                        smoothing_fwhm=4, standardize=True,
                        memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression


fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

from sklearn.feature_selection import RFE
feature_selection = RFE(LogisticRegression(penalty = 'l1', C = 1 , solver = 'liblinear'),360, step=0.2)
rfe_lr = OneVsRestClassifier( Pipeline([('rfe', feature_selection), ('lr', LogisticRegression(penalty = 'l1', C = 1
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat') #conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]




from sklearn.metrics import confusion_matrix

Confused = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    rfe_lr.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = rfe_lr.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused += temp_matrix

"""
The code for drawing a nice plot.
"""

fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused, cmap='YlGnBu')

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
```

```
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for RFE Preprocessor and\nLogistic Regression with Penalty l1 Norm Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')


# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)

for i in range(8):
    for j in range(8):
        if Confused[i, j] > 50:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python


"""
RFE, Logistic Regresion l2, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <= (15 + 36*abc + 20)))] = behavior ['tri
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j]) #index_img(haxbydir + i + '/'+'
                mask = masking.compute_brain_mask(target_img, threshold=0.5, connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                         smoothing_fwhm=4, standardize=True,
```

```
                        memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression


fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

from sklearn.feature_selection import RFE
feature_selection = RFE(LogisticRegression(penalty = 'l2', C = 1 , solver = 'liblinear'),360, step=0.2)
rfe_lr = OneVsRestClassifier( Pipeline([('rfe', feature_selection),
('lr', LogisticRegression(penalty = 'l2', C = 1 , solver = 'liblinear'))]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat')
#conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]




from sklearn.metrics import confusion_matrix

Confused = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    rfe_lr.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = rfe_lr.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused += temp_matrix

"""
The code for drawing a nice plot.
"""

fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused, cmap='YlGnBu')

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
# ... and label them with the respective list entries
```

```
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for RFE Preprocessor and\nLogistic Regression with Penalty l2 Norm Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')


# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)

for i in range(8):
    for j in range(8):
        if Confused[i, j] > 50:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python


"""
RFE, SVM l1 Norm, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <= (15 + 36*abc + 20)))]
                    = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j]) #index_img(haxbydir + i + '/'+'
                mask = masking.compute_brain_mask
```

```
                (target_img, threshold=0.5, connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                          smoothing_fwhm=4, standardize=True,
                          memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier

fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

from sklearn.feature_selection import RFE
feature_selection = RFE(LinearSVC(penalty='l1', C = 0.1 , dual=False), 50, step=0.25)
#LinearSVC(), 50, step=0.25
rfe_svc = OneVsRestClassifier( Pipeline([('rfe', feature_selection), ('svc', LinearSVC(penalty='l1', C = 0.1 , dual=
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat')
#conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]



from sklearn.metrics import confusion_matrix


Confused = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    rfe_svc.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = rfe_svc.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused += temp_matrix

"""
The code for drawing a nice plot.
"""

fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused, cmap='YlGnBu')

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
# ... and label them with the respective list entries
```

```
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for RFE Preprocessor \n and SVM with Penalty l1 Norm Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')


# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)


for i in range(8):
    for j in range(8):
        if Confused[i, j] > 50:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="k")
```

```
import os
import numpy as np
import pandas as pd
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python


"""
RFE, SVM l2 Norm, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/a/Documents/Brain data for EEE 482 project/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <= (15 + 36*abc
                    + 20)))] = behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j]) #index_img(haxbydir + i + '/'+'
```

```
                mask = masking.compute_brain_mask(target_img, threshold=0.5, connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                         smoothing_fwhm=4, standardize=True,
                         memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier

fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

from sklearn.feature_selection import RFE
feature_selection = RFE(LinearSVC(penalty='l2', C = 0.01 , dual=False), 50, step=0.25)
#LinearSVC(), 50, step=0.25
rfe_svc = OneVsRestClassifier( Pipeline([('rfe', feature_selection),
('svc', LinearSVC(penalty='l2', C = 0.01 , dual=False))]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat')
#conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]



from sklearn.metrics import confusion_matrix


Confused_Mask_SVM_l2 = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    rfe_svc.fit(fmri_masked2[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = rfe_svc.predict(fmri_masked2[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused_Mask_SVM_l2 += temp_matrix

"""
The code for drawing a nice plot.
"""

fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused_Mask_SVM_l2, cmap='YlGnBu')

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
```

```
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion Matrix for RFE Preprocessor \n and SVM with Penalty l2 Norm Classifier")
plt.xlabel('True Class')
plt.ylabel('Predicted Class')


# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)


for i in range(8):
    for j in range(8):
        if Confused_Mask_SVM_l2[i, j] > 50:
            text = ax.text(j, i, Confused_Mask_SVM_l2[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused_Mask_SVM_l2[i, j], ha="center", va="center", color="k")




succes_count = 0

for i in range(8):

    succes_count += Confused_Mask_SVM_l2[i, i]


succes_rate = succes_count / 7.68
```

E. MASK+SVD+LDA Confusion Matrix

Listing 1. MASK+SVD+LDA Confusion Matrix

```
import os
import numpy as np
import pandas as pd
from nilearn import plotting
from nilearn.image import index_img
from nilearn.image import load_img
from nilearn import masking
from nilearn.input_data import NiftiMasker

import matplotlib.pyplot as plt # this brings nice loking plots to python

"""
Anova, LDA, Confusion matrix
"""

'''
Write the filepaths by changing \ to /
'''
#plotting.plot_img('C:/Users/mehme/EEE482_data/sub-1/anat/sub-1_T1w.nii.gz')



def parser():
    haxbydir = "C:/Users/mehme/EEE482_data/"
    haxbysubjects = os.listdir(haxbydir)
    timehorizon = np.arange(0,302.5,2.5)

    #reference arrays for dictionaryies
    haxbysubjects = haxbysubjects[4:10]
    inputs = ['face', 'cat', 'shoe', 'chair', 'scissors', 'bottle', 'house', 'scrambled']
    timehorizon = np.arange(0,302.5,2.5)

    #Parsing directory into dictionary for easier access
    haxby = {}
    for i in haxbysubjects:
        directory = os.listdir(haxbydir+i)
```

```
        haxby[i] = {"mask": haxbydir + i + '/'+directory[2],
                    'anat': haxbydir + i + '/'+directory[0]+'/'+(os.listdir(haxbydir+i+'/'+directory[0])[0]),
                    'fun': os.listdir(haxbydir + i + '/'+directory[1])}
    for i in (haxbysubjects):
        haxby[i]['data'] = {}
        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                continue
            else:
                behavior = pd.read_csv(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j], sep='\t')
                conditions = np.array(['rest']*121)
                for abc in range(8):
                    conditions[(((15 + 36*abc)<timehorizon) & (timehorizon <= (15 + 36*abc
                    + 20)))] =
                    behavior ['trial_type'][abc*12]
                haxby[i]['data'][str(int((j+1)/2))+'_behavioral'] = conditions

        for j in range(len(haxby[i]['fun'])):
            if haxby[i]['fun'][j].find('.tsv') == -1:
                target_img = load_img(haxbydir + i + '/'+'func/'+haxby[i]['fun'][j]) #index_img(haxbydir + i + '/'+'
                mask = masking.compute_brain_mask(target_img, threshold=0.5,
                connected=True, opening=2, verbose=0)
                masker = NiftiMasker(mask_img=mask,
                         smoothing_fwhm=4, standardize=True,
                         memory="nilearn_cache", memory_level=1)
                haxby[i]['data'][str(int((j+2)/2))+'_fmrimasked'] =  masker.fit_transform(target_img)
    haxby.pop('sub-5')
    haxbysubjects.pop(4)



    return haxby, haxbysubjects, inputs, masker

haxby, haxbysubjects, inputs, masker = parser()


from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.model_selection import cross_validate
from sklearn.svm import LinearSVC, SVC
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.multiclass import OneVsRestClassifier
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis


fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
conditions = haxby['sub-2']['data']['1_behavioral']
session_label = np.zeros(121)

from sklearn.feature_selection import RFE
feature_selection = SelectPercentile(f_classif, percentile=0.15)
rfe_svc = OneVsRestClassifier( Pipeline([('Anova', feature_selection), ('svc',
LinearDiscriminantAnalysis() )]) )
fmri_masked = haxby['sub-2']['data']['1_fmrimasked']
cv = LeaveOneGroupOut()

for i in range(2,13):
    fmri2 = haxby['sub-2']['data'][str(i) + '_fmrimasked']
    fmri_masked = np.concatenate((fmri_masked, fmri2), axis=0)
    conditions = np.concatenate((conditions, haxby['sub-2']['data'][str(i) + '_behavioral']))
    group2 = np.ones(121)*(i-1)
    session_label  = np.concatenate((session_label, group2))

condition_mask = conditions != 'rest' #(conditions  == 'face') | (conditions  == 'cat')
#conditions != 'rest'

conditions2 = conditions[condition_mask]
session_label2 = session_label[condition_mask]
fmri_masked2 = fmri_masked[condition_mask]
u, s, vh = np.linalg.svd(fmri_masked2, full_matrices=True)


from sklearn.metrics import confusion_matrix
from nilearn.plotting import plot_matrix, show
```

```
Confused = np.zeros((8, 8))

for i in range(12):


    print("\n\n\n")
    print(i) # for understanding run time

    rfe_svc.fit(u[session_label2 != i] , conditions2[session_label2 != i] )

    y_pred_ovo = rfe_svc.predict(u[session_label2 == i])

    temp_matrix = confusion_matrix(y_pred_ovo, conditions2[session_label2 == i])

    Confused += temp_matrix

"""
The code for drawing a nice plot.
"""

fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(Confused, cmap='YlGnBu' )

# We want to show all ticks...
ax.set_xticks(np.arange(len(inputs)))
ax.set_yticks(np.arange(len(inputs)))
# ... and label them with the respective list entries
ax.set_xticklabels(inputs)
ax.set_yticklabels(inputs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

plt.title("Confusion␣Matrix␣for␣Anova␣Preprocessor␣and␣\n␣LDA␣Classifier")
plt.xlabel('True␣Class')
plt.ylabel('Predicted␣Class')


# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)

for i in range(8):
    for j in range(8):
        if Confused[i, j] > 50:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="w")
        else:
            text = ax.text(j, i, Confused[i, j], ha="center", va="center", color="k")
```