Machine learning classifiers and fMRI: a tutorial overview

Francisco Pereira [1] fpereira@princeton.edu

Tom Mitchell [2] tom.mitchell@cs.cmu.edu

Matthew Botvinick [1] matthewb@princeton.edu

1) Princeton Neuroscience Institute and Psychology Department, Princeton University, Princeton NJ 08540

2) Machine Learning Department, Carnegie Mellon University, Pittsburgh PA 15213

July 15, 2008

Abstract

Interpreting brain image experiments requires analysis of complex, multivariate data. In recent years, one analysis approach that has grown in popularity is the use of machine learning algorithms to train classifiers to decode stimuli, mental states, behaviors and other variables of interest from fMRI data and thereby show the data contain enough information about them. In this tutorial overview we review some of the key choices faced in using this approach as well as how to derive statistically significant results, illustrating each point from a case study. Furthermore, we show how, in addition to answering the question of 'is there information about a variable of interest' (pattern discrimination), classifiers can be used to tackle other classes of question, namely 'where is the information' (pattern localization) and 'how is that information encoded' (pattern characterization).

1 Introduction

In the last few years there has been growing interest in the use of machine learning classifiers for analyzing fMRI data. A growing number of studies has shown that machine learning classifiers can be used to extract exciting new information from neuroimaging data (see [32] and [17] for selective reviews). Along with the growth in interest and breadth of application, the methods underlying the use of classifiers with fMRI have continuously evolved and ramified (see [34] for a historical overview). Given the novelty of the approach, there have been few attempts to organize and interrelate available methods in a single place. The present article strives to rectify that situation by providing a tutorial introduction to classier methods in fMRI.

Our presentation will be organized around the idea that classifier- based analyses, like traditional fMRI analyses, can be characterized in terms of a series of specific choices over a series of decision points in the analysis process, starting from selection of the scientific question to be asked and ending with choices among tests for hypotheses. We begin by laying out an illustrative example of a classifier-based analysis, and then dissect the analysis process, examining the set of choices it implicitly involves, and the alternatives available at each stage. There have been other proposals for a staged procedure for using classifiers to analyse fMRI data (e.g. [40]), though with an emphasis on a single type of classifier and requiring the use of dimensionality reduction. We broaden this idea to include several kinds of classifier and also the use of feature selection. We conclude with a discussion of the sorts of scientific questions that may be fruitfully addressed using classifiers, and the ways in which the choice of question impacts subsequent analysis decisions.

Where possible, we discuss strengths and weaknesses of competing options. However, it is important to acknowledge from the outset that firm grounds for such evaluations are in many cases not yet available. Where possible, we offer recommendations based on the results of formal principles or controlled empirical tests (from [35]). Where these are lacking, we shall sometimes inject impressions drawn from our own personal experience with classifier-based analysis of fMRI data. Before entering into the discussion of the analysis proper, we begin in the next section by briefly introducing machine learning classifiers in their own right.

1.1 What is a classifier?

Classification is the analogue of regression when the variable being predicted is discrete, rather than continuous. A classifier is a function that takes the values of various *features* (independent variables or predictors, in regression) in an *example* (the set of independent variable values) and predicts the *class* that that example belongs to (the dependent variable). In a neuroimaging setting, the features could be voxels and the class could be the type of stimulus the subject was looking at when the voxel values were recorded (see Figure 1). We will denote an example by the row vector $\mathbf{x} = [x_1 \dots x_v]$ and its class label as y. A classifier has a number of parameters that have to be

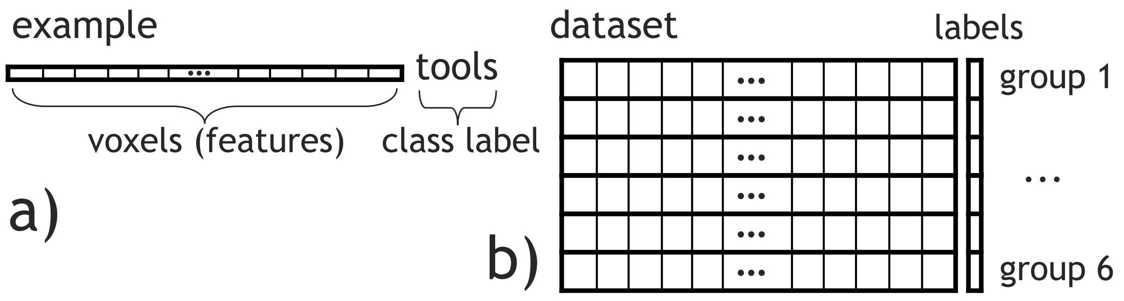


Figure 1: An example where voxels are features as a row vector (left) and a dataset as matrix of such vectors (right).

learned from *training data* – a set of examples reserved for this purpose – similarly to how regression parameters are estimated using least squares. The learned classifier is essentially a model of the relationship between the features and the class label in the training set. More formally, given an example $\mathbf{x}$, the classifier is a function f that predicts the label $\hat{y} = f(\mathbf{x})$.

Once trained the classifier can be used to determine whether the features used contain information about the class of the example. This relationship is *tested* by using the learned classifier on a *different* set of examples, the *test data*. Intuitively, the idea is that, if the classifier truly captured the relationship between features and class, it ought to be able to predict the classes of examples it hasn't seen before. The typical assumption for classifier learning algorithms is that the training (and testing) examples are independently drawn from an 'example distribution'; when judging a classifier on a test set we are obtaining an estimate of its performance on *any* test set from the same distribution. This is depicted in part c) of Figure **??**. We will denote the training and test sets by X_{train} and X_{test}, matrices with respectively n_{train} and n_{test} examples as their rows, and the example labels by the column vectors $\mathbf{y}_{train}$ and $\mathbf{y}_{test}$. The most commonly used measure of how well a classifier does on the test set is its *accuracy*. This is simply the fraction of examples in the test set for which the correct label was predicted, i.e. $\frac{\sum_{i=1}^{n_{test}} I(f(\mathbf{x}_i),y_i)}{n_{test}}$, where $I(f(\mathbf{x}_i),y) = 1$ if $f(\mathbf{x}_i) = y_i$ (the label of the i^{th} example was predicted correctly) or 0 otherwise.

As we shall discuss in Section 3.2, there are several types of classifier. However, for reasons that will become clear, our emphasis will be on *linear* classifiers; in this type the classification function relies on a linear combination of the features, i.e. $f(\mathbf{x}) = g(w_1x_1 + \ldots + w_Vx_V)$, and the weights w_i in that combination are the parameters to be learned.

2 Classifier analysis: an illustrative example

In this section we will introduce an illustrative example of a classifier-based study, which will provide a basis for subsequent discussion.

The experiment that gave rise to our dataset was designed to test the question of whether the activation as a result of seeing words that were either kinds of tool or kinds of building could be distinguished by a classifier. [1] The subject was shown one word per trial and performed the following task: the subject should think about the item and its properties while the word was displayed (3 seconds) and clear her mind afterwards (8 seconds of blank screen). There were 7 different items belonging to each of the two categories and 6 experimental epochs. In each epoch all 14 items were presented, without repetition; all 6 epochs had the same items.

Images were acquired with a TR of 1 second, with voxel dimensions $3 \times 3 \times 5mm$. The dataset underwent a typical fMRI preprocessing stream from DICOM files using SPM [38]. The steps followed were volume registration and slice acquisition timing correction, followed by a voxelwise linear detrending. A brain mask was extracted automatically and used to restrict voxels considered in the subsequent steps. Although the original dataset contained several subjects, we will focus on a single one.

Each trial was converted into an *example* by taking the average image during a 4 second span while the subject was thinking about the stimulus shown a few seconds earlier; these 4 seconds contained the expected peak of the signal during the trial. Each example was thus a vector of between 10000 and 20000 values, *feature values*, each corresponding to the 4 second average signal at a voxel. In each average image only between 10000 and 20000 of the voxels were contained in the brain mask. Using each trial to obtain an example yielded a total of 42 examples

[1] Data kindly provided by Robert Mason and Marcel Just, Center for Cognitive Brain Imaging, CMU, who ran the experiment.

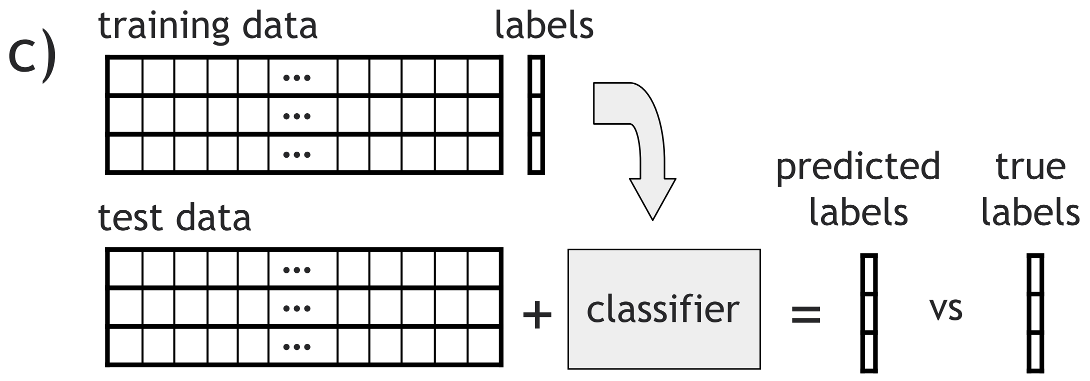


Figure 2: A classifier is learned from the training set, examples whose labels it can see, and used to predict labels for a test set, examples whose labels it cannot see. The predicted labels are then compared to the true labels and the classifier's accuracy computed.

for each of the tools and buildings categories. Each example vector was normalized to have mean 0 and standard deviation 1. Examples were further grouped by which of the six epochs they belonged, as depicted in Figure 1.

With examples in hand, the dataset was divided into a training and a test set, with a goal of learning a classifier whose *task* was to predict the category of an example. As mentioned previously, the parameters of a classifier are learned from the examples in the training set, and the classifier is then evaluated on the examples in a separate test set (see Figure 2). In the example study we divided the dataset for one subject into examples coming from either the even or the odd epochs, assigning those in epochs 1,3 and 5 to be the training set (42 examples) and those in epochs 2,4 and 6 to be the test set (42 examples). Given this, we trained a Gaussian Naive Bayes classifier (see Section 3.2) and applied it to the test set. For 65% of the examples the label was predicted correctly. Given this result, could we conclude that the classifier learnt from the training set was capable of predicting the labels of the test set better than chance and, therefore, that the fMRI features contained relevant information about the stimulus category? Or could it just have done this well on the test set by chance? In order to test the statistical significance of this result, we considered the null hypothesis that the classifier was performing at chance level (e.g. the accuracy of prediction on average would be 50%) and tested it using the procedure we describe in Section 3.4. For an accuracy of 65% with 42 test examples the test yielded a p-value of 0.0218 and the null hypothesis was rejected.

3 Classifier analysis: stages and options

The foregoing example, though presented as an unsegmented narrative, can be broken down into a set of *stages*, beginning with the conversion of raw data into a set of examples and proceeding though choice of classifier, training and test sets and interpretation of results. At each stage, the researcher faces a choice among multiple alternatives. In what follows we examine each stage and the choices it presents, as well as how those choices interact and are conditioned by practical factors.

3.1 Creating examples

Creating examples in general requires deciding what to use as features, how to extract their values and what we would like to predict. In our illustrative example we chose to average several TRs worth of images in a single trial to create an example; as a single word stimulus maps to a single trial this is a natural choice. Many other options are available, however. We could instead have used each individual TR within a trial as an example by itself, or averaged many trials with words belonging to the same category into a single example. One is not limited to using voxels as features. We could use the average of several voxels in one ROI as a single feature (essentially, the ROI becomes the feature) or use 'voxels at a particular time point in a trial'. The latter is used in [29] in a situation where what distinguishes the activation during processing of ambiguous and unambiguous sentences is the time course of activation in various voxels active in both conditions. Another possibility is to forsake creating examples directly from the activation signal and instead do it from deconvolution coefficient images, if each class can be identified with a regressor.

The question of what to use as class labels depends on the purposes of the researcher, but we note that one is not limited to using stimuli classes. One could also use subject responses or decisions in face of a stimulus (e.g. 'did the subject get this right or wrong?'). A particularly creative example of this is the study [16], where the prediction is of which of two possible images a subject reports perceiving in a binocular rivalry situation.

In creating examples, there is an important tradeoff associated with the number of examples produced. This is the tradeoff between having many noisy examples (e.g. one per trial) or fewer, cleaner ones (e.g. one of each class per run), as a result of averaging images in the same class. Although there is no hard and fast number of examples necessary to train a classifier, having more is generally better, to the degree that the classifier can see through the noise present (whereas averaging eliminates both noise and natural variability of informative activation in the examples belonging to a class). Having more examples helps in the training set side in that some of the classifiers we consider require a certain number of examples to obtain good estimates of their parameters. Conversely, there are some classifiers that are particularly suitable for a situation with very few examples, and we discuss this at greater length in Section 3.2. From the test set standpoint, having more test examples increases the power of the test for significance of the accuracy obtained. In Section 3.3 we will introduce cross-validation, a procedure that makes very efficient use of examples for both training and testing.

One issue to be cognizant of is the desirability of having the same number of examples in each class. WHen this is not the case a classifier learning algorithm may tend to focus on the most numerous class, to the detriment of the others; this will also affect the interpretation of an accuracy result (e.g. 80% accuracy may not be very good in a situation where 9 of 10 examples belong to one class and 1 of 10 to the other, if it means the classifier simply predicts the most numerous class by default). This balance between classes can be achieved by using only a subset of the examples in the most numerous class, repeating the procedure for various subsets and averaging the results.

Note that using consecutive images as individual training examples should be avoided, as the strong temporal autocorrelation in the signal will virtually ensure those examples are not independent. Intuitively, it also means that having several very similar examples will not bring in much new information. This does not bar us from using consecutive images in the test set, depending on the purpose (see Section 4 for an example where training examples come from block data and test examples from event-related data). Non-independent test examples cannot, however, be used for the significance test described earlier (the binomial distribution outcome is the sum of the results of *independent* Bernoulli trials).

Given that there are generally many more features than examples, it is often advantageous to reduce the number of features considered to focus on a subset of particular interest; this is called *feature selection*. For instance, if using voxels as features, we may just want the voxels in a particular region of interest (ROI) or their values at a particular point in a trial. The crucial issue to keep in mind is that the choice of features at this stage must not depend on the labels that will be used to train a classifier. To see why, imagine a situation where we are interested in predicting the orientation of a stimulus grating. It is acceptable to restrict the voxels considered to those in visual cortex, demarcated anatomically. It is also acceptable to restrict them further to those showing some activity during task trials relative to a baseline. What is not acceptable to select voxels that appear to distinguish one orientation from another in the *entire dataset*. The reason for this is that it is a subtle form of letting the test set information affect the learning of the classifier in the training set, and it leads to optimistic accuracy estimates. Looking at the labels for the entire dataset is sometimes called 'peeking', Note that this does not mean that the class labels cannot be used at all in feature selection. They can be used once the data have been divided into training and test sets, considering solely the training set, and we discuss all practical aspects of doing it in Section 3.3.3.

An alternative path to reducing the number of features a classifier has to consider is *dimensionality reduction* using a method such as Singular Value Decomposition/Principal Component Analysis [12] or Independent Components Analysis [3] on the entire dataset matrix (rows are examples, columns are features). The commonality between these methods is that they transform the original feature space (e.g. voxels) into a new, low-dimensional feature space, yielding a new dataset matrix with the same number of rows and a small number of columns. This is always worth trying and a necessary step for particular classifiers, as we will see in Section 3.2, but not at all guaranteed to improve results, partially because most dimensionality reduction techniques ignore class labels in their criteria (though see SVDM [36] and PLS [27]).

A final issue to consider in the construction of examples is that of preprocessing. By this we do not mean the usual preprocessing of data, e.g. motion correction or detrending, a topic covered in great depth in [39]. We mean, rather, that done on the examples, considered as a matrix with N rows (where each row is an example) and V columns (features). In the example study, we normalized each example (row) to have mean 0 and standard deviation 1. The idea in this case is to reduce the effect of large, image-wide signal changes. Another possibility would be to normalized each feature (column) to have mean 0 and standard deviation 1, either across the entire experiment or within examples coming from the same run. This is worth considering if there is a chance that some voxels will have much wider variation in signal amplitude than others. Although a linear classifier can in principle compensate for this to some degree by scaling the coefficient for each voxel, there are situations where it will not and thus this normalization will help. Our own experience suggests either row or column normalization is generally beneficial and should be tried in turn, whereas combining them may sometimes have adverse effects.

3.2 Choosing a classifier

Earlier we introduced the idea of a classifier as a function f that takes an example $\mathbf{x}$ and generates a class label prediction $\hat{y} = f(\mathbf{x})$. The specific kind of function being learnt – and the assumptions built into it – is what distinguishes the various types of classifier. In this section we review and contrast the classifiers most commonly used with fMRI data, as well as the factors at stake in choosing which kind to use. We supply much additional detail in Appendix A.

The simplest classification procedure is called 'nearest-neighbour' and it doesn't even involve explicitly learning a classification function. Classification of a test example is done by finding the training set example that is most similar to it by some measure (e.g. lowest euclidean distance, considering the entire feature vector) and assigning the label of this nearest neighbour to the test example. Variants of this idea include averaging all the training set examples in each class into a single class 'prototype' example or assigning the majority label in the k-nearest neighbours (k odd). Both variants minimize the effect of noise (if the examples of different classes are all relatively close to each other) but can be susceptible to destroying some information if there is actual non-noise variability in the examples in each class (e.g. examples in one class come in two clusters that are relatively dissimilar to each other). Whereas nearest-neighbour classification can work very well if there is a small number of features, it most likely will not in a situation where there is a large number and only a few are informative. Hence it is generally used in conjunction with some form of feature selection, as in [15] or [29].

Classifiers that do learn an actual function divide into what are called *discriminative* and *generative* models. In the former, a prediction function with a given parametric form is learnt directly from the training data by setting its parameters. In the latter, what is learned is essentially a statistical model that could *generate* an example belonging to a particular class (i.e. it models the distributions of feature values conditional on example class, $P(\mathbf{x}|y = A)$ and $P(\mathbf{X}|y = B)$), which is then inverted via Bayes Rule to classify (i.e. yields $P(y = A|\mathbf{x})$ and $P(y = B|\mathbf{x})$, and the prediction is the class for which that probability is largest).

In the typical fMRI study, we generally have many more features than examples; if using voxels in the whole brain as features, it's not unusual to have a few tens of examples and tens of thousands of features (or at least a few hundreds, if using feature selection as described in Section 3.3.2). The effect of this is that it will generally be possible to find a function that can classify the examples in the training set well, without this necessarily meaning that it will do well in the test set (for a more detailed explanation of the reason why see [28]): this phenomenon is called *overfitting* [2]. One way of trying to avoid overfitting is to use as simple a function as possible that does well in the training set.

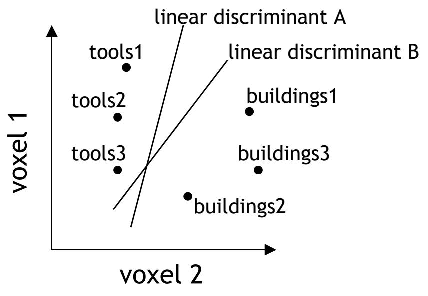
The most natural choice for a simple function is to have the prediction depend on a linear combination of the features that can dampen or increase the influence of each one of them, just as linear regression. To make the idea of a linear classifier more concrete, the classifier is parameterized by a set of weights $\mathbf{w}$ and, for an example $\mathbf{x}$ with V features

$$\mathbf{xw} = x_1 w_1 + \ldots + x_V w_V$$

the classifier predicts class A if $\mathbf{xw} > 0$ or class B if $\mathbf{xw} < 0$ (with ties broken randomly). One nice property of linear classifiers, beyond their simplicity, is the fact that each feature affects the prediction solely via its weight and without interaction with other features, giving us a measure of its influence on that prediction. We discuss ways to use this, as well as some caveats, in Section 4.1.

A more geometric intuition for this is given in Figure 3, in a rather simplified two voxel brain, with three examples of each class (each example is characterized by the value of the two voxels). Learning a linear classifier is equivalent to learning a line that separates points in the two classes as well as possible, the *decision boundary.* Examples of classifiers that learn this are Logistic Regression (LR) and linear Support Vector Machines (SVM). Whereas this might seem remote from the generative model idea introduced earlier, the classification decisions of Gaussian Naive Bayes (GNB) and Fisher's Linear Discriminant Analysis (LDA) can be expressed in this manner as well, and thus properly included in the linear classifier category. Note that there are many possible linear discriminants that perfectly classify the six training examples shown in Figure 3. The various classifiers have different rationales for choosing among these, which correspond to different definitions of 'separating points as well as possible'. Consider also that, even for a specific type of classifier, there may still be many possible settings of its parameters that lead to equally good predictions in the training set. The process of guiding the procedure that sets the parameters to a solution with desirable properties – small weights overall, for instance – is called

[2] An example of this would be to use a very high-degree polynomial to model points in a time series coming from a low-degree polynomial with added noise. The very high-degree polynomial would likely be worse at approximating new points than a low-degree polynomial would.



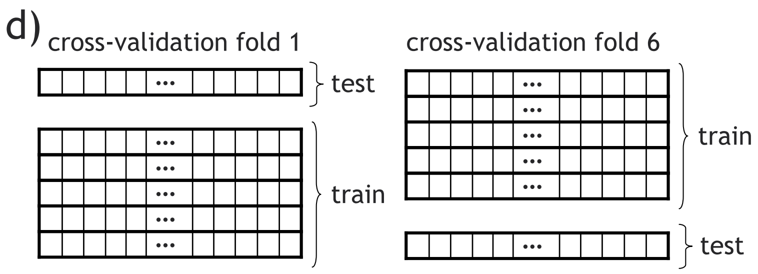


Figure 3: **left:** Learning a linear classifier is equivalent to learning a line that separates examples in the two classes (vectors in a 2-voxel brain) as well as possible. **right:** During cross-validation each of 6 groups of examples takes a turn as the test set while the rest serve as the training set.

regularization. Both the rationales and the way regularization is done for the various kinds of classifiers are described in Appendix A.

All this brings us to the question of which classifier to use, which we will attempt to answer based on our experience and a multi-study comparison of classifiers and feature selection methods [35]. In cases where there are a large number of features (e.g. all the voxels in the cortex), GNB is generally inferior to LR and linear SVM, as the regularization in the latter clearly helps weigh down the effect of noisy features that are highly correlated with each other. The disadvantage mostly disappears if using feature selection, hence GNB is a useful classifier for procedures that need to be repeated many times, such as permutation tests, due to being much faster to train than the others. LDA needs to be used either in conjunction with extreme feature selection or with dimensionality reduction ([40]), as otherwise there are typically not enough examples to estimate its covariance matrix reliably.

If choosing between LR and linear SVM, they are roughly equivalent other than for the fact that the first lends itself more naturally to cases where there are more than two classes. So far, and for the sake of simplicity, we have discussed classifiers in a situation where examples belong to one of two classes but, in general, we might have more (several different stimulus classes, subject responses, etc). Most of the classifiers described above work without modification in a multiple class scenario. The one exception are the SVM classifiers; although multiclass versions do exist, most packages in current use require you to create several two class problems (e.g. each class versus all the others, or all pairs of classes) and combine them to produce a multiway classification decision. We describe some approaches to doing this in Appendix A. That said, often the object of interest in a multiple class situation is not so much the raw accuracy (do we believe that hundreds of types of object stimuli would be distinguishable based on fMRI activation, for instance?) as which classes can be distinguished from which others, and how. Hence, it might make more sense to consider all pairs of classes, train and test a classifier for each pairwise distinction and produce what is called a *confusion matrix*, a `#classes` $\times$ `#classes` matrix where entry (i, j) contains the accuracy of the distinction between classes i and j. Often this reveals groups of classes that are hard to distinguish from each other but that are distinguishable from another group (e.g. several small object stimuli versus animal and human faces).

There are more complex classifiers, such as nonlinear SVMs or artificial neural networks, which can let interactions between features and nonlinear functions thereof drive the prediction; we will not discuss them at length for two main reasons. The first is that it is unclear that they provide a significant advantage in practical performance relative to linear classifiers (in extant studies or the multi-study comparison in [35]). In our opinion this is more a reflection of the fact that the number of examples available is so small than of the inexistence of complicated relationships between features. The second reason is that the relationship between features and the prediction becomes harder to interpret (though see [13] which derives a class taxonomy from the hidden layer node activity in a neural network, for instance). This doesn't mean that these classifiers cannot be used fruitfully when the number of features is smaller, in particular when doing information mapping as discussed in Section 4.2. We describe these classifiers and their relationship to linear classifiers in more detail Appendix A.

3.3 Training and testing in cross-validation

3.3.1 Cross-validation

In our illustrative example we divided the dataset in two halves, one used for training and the other for testing. As we later discussed, we would like to train a classifier with as much data as possible; sadly, we cannot train and test on the same data if the goal is obtain a useful estimate of the classifier's true accuracy, i.e. the probability that it would label any future test example correctly. Furthermore, using just a few examples for testing will not lead to a good estimate: it will be inordinately variable, as well as constrained to a few possible values. Splitting the dataset in half to have more test examples, as we did in the example study, means we might be learning a much worse classifier than we potentially could with more training examples. Fortunately, there is a procedure that allows us to have our cake and eat (almost) all of it too... The procedure is called *cross-validation*. In its most extreme variant, cross-validation follows these steps:

- leave one example out, train on the remaining ones, make a prediction for this example
- repeat for each example in turn
- compute the accuracy of the predictions made for all the examples

This variant is called 'leave-one-out' cross-validation. Although each classifier trained is technically different, one can expect them to be similar and predict similarly, since they share so much training data. Hence we treat the accuracy estimate obtained in this manner as the expected accuracy of a classifier trained on a dataset with all but one example, following [23]. Note this also provides a slightly conservative estimate of the accuracy we expect if we were to train a classifier using all the available examples.

In practice, leaving each example out can be computationally expensive because instead of a single classifier we are training as many classifiers as there are examples. One compromise is a procedure called a k-fold cross-validation, where k is the number of parts into which the dataset is divided; common choices are $k = 10$ or $k = 5$, corresponding to leaving out 10% or 20% of the examples on each fold. Fortunately, fMRI datasets are often naturally divided in a way that addresses all these issues, be it in runs or epochs within a run where all the possible classes or stimuli appear. Our illustrative dataset divides naturally into six groups, corresponding to the six epochs of stimuli in the experiment; each group is used in turn as the test set in cross-validation, with the remaining groups used as the training set (on the right of Figure 3). Hence we advise the reader to take advantage of those natural divisions at least on a first attempt, possibly moving to a finer grain division that still has the desired properties if more training examples are needed.

Aside from the number of examples left out there are other important considerations. The first is that the training data in each fold must contain examples of all classes, as otherwise the classifier for that fold will not be able to predict the absent ones. As mentioned earlier, classes should be balanced, i.e. have roughly the same number of examples. Furthermore, examples that are correlated - by being close in time, for instance - should end up in the same fold. Otherwise, the classifier may be able to predict accurately for test examples with a correlated counterpart in the training set.

3.3.2 Feature selection

In Section 3.1 we introduced the idea of feature selection, in situations where we have some *a priori* reason for picking them, e.g. wishing to test whether voxels in a particular ROI have information. Feature selection is often deemed necessary in order to train classifiers in domains where datasets have very large numbers of features. Why might it help? Intuitively, the goal is to reduce the ratio of features to examples – decreasing the chance of overfitting – as well as to getting rid of uninformative features to let the classifier focus on informative ones. Almost all the fMRI decoding results reported in the literature used either feature selection or some other form of dimensionality reduction ([32],[17],[4],[40]). Feature selection in general is too vast a topic to broach in great detail here, but a useful review of both terminology and the thrust of various approaches is given in ([11]).

One distinction often made is between *scoring/filtering* and *wrapper* methods. The former involves ranking the features by a given criterion - each feature is scored by itself, a bit like a univariate test of a voxel - and selecting the best in the ranking. The latter consists broadly of picking new features by how much impact they have on the classifier *given the features already selected*. This can be done by repeatedly training and applying the classifier in cross-validation within the training set. Given the often prohibitive computational expense of wrapper methods – due to the large number of features – we will be concerned solely with feature scoring methods.

There are several generic methods for selecting informative features. In parallel, features in the fMRI domain are often voxels and there is a desire to be able to look for certain types of voxel behaviour, e.g. is a voxel very selective for one particular condition or discriminating between two groups of conditions. Hence most extant papers resort to methods that draw from both these sources, which we summarize below. Note that method

method	number of voxels						method	NCV
	100	200	400	800	1000	all		
accuracy	0.81	0.81	0.75	0.73	0.74	0.65	accuracy	0.81
searchlight	0.81	0.82	0.82	0.77	0.79	0.65	searchlight	0.82
activity	0.79	0.80	0.77	0.73	0.74	0.65	activity	0.77
ANOVA	0.77	0.75	0.75	0.73	0.71	0.65	ANOVA	0.77

Table 1: Classification results using each of several selection methods with either various fixed numbers of voxels (left) or with the number determined using nested cross-validation (NCV) inside the training set of each fold (right).

rationales are described in terms of voxels and conditions (classes), for clarity, but could be adapted for other kinds of features:

- Activity - This method selects voxels that are active in at least one condition relative to a control-task baseline, scoring a voxel as measured by a t-test on the difference in mean activity level between condition and baseline.
- Accuracy - This method scores a voxel by how accurately a Gaussian Bayesian classifier can predict the condition of each example in the training set, based only on the voxel. It does a cross-validation within the training set for the voxel and the accuracy is the voxel's score.
- Searchlight Accuracy - The same as Accuracy, but instead of using the data from a single voxel (GNB classifier with one feature) we use the data from the voxel and its immediately adjacent neighbours in three dimensions (a classifier with up to 27 features, classifier can be GNB, LDA or SVM).
- ANOVA - This method looks for voxels where there are reliable differences in mean value across conditions (e.g. A is different from B C D, or AB from CD, etc), as measured by an ANOVA.
- Stability - This method picks voxels that react to the various conditions *consistently* across cross-validation groups in the training set (e.g. if the mean value in A is less than B and greater than C, is this repeated in all groups).

Precise details of how these methods can be implemented and other alternatives are provided in Appendix B. Note that the methods above are, with exception of searchlight accuracy, univariate. There are promising results that show that, in some circumstances, multivariate feature selection can work better, which are discussed in Section 4. Finally, the methods listed above provide rankings of voxels; one might still have to decide on how many of the top-ranked voxels to use from the ranking for each method. For methods that are based on a statistical criterion (e.g. t-tests of activation), it is common to use a multiple comparison criterion such as Bonferroni or False Discovery Rate [8] and keep just the features deemed significant at a given level. This is reasonable but problematic if few or no voxels are significant (possible given the amount of noise and very small sample sizes) or the score produced by the method does not map to a p-value (though nonparametric methods can still be used to obtain one, see Appendix C).

3.3.3 Feature selection and cross-validation

Up till now we have discussed feature selection as a topic by itself. Often, however, feature selection is used in conjunction with cross-validation and this entails particular precautions. The first is that selection must happen *on the training set*. As mentioned earlier, for every fold in cross-validation we have a different training set and thus features have to be selected *separately* for each fold. A second reason is that, as mentioned earlier, the methods used in practice *rank* features and we still have to decide how many to use, unless the method provides a rationale (e.g. the number of voxels where a t-test is significant at a certain level). A typical approach is to not commit to a specific number, but to instead produce a feature ranking by each method for each fold. This can then be used to obtain, for each method, an accuracy score using each of the numbers of voxels desired (e.g. top 20, top 50, top 100, etc). The end product is a #`methods` $\times$ #`voxels` table of accuracy results, such as Table 1 (left) for our sample dataset.

If feature selection could be done prior to cross-validation (e.g. by selecting voxels from one ROI, or by some other method that avoids examimining the labels of the test examples), the whole procedure can be run as if the dataset contained only the selected features. In this and the case where the method determines the number of features, the end product is a #`methods` $\times$ 1 vector of accuracy results.

An alternative to considering all numbers of voxels is to let the training set in each cross-validation suggest which number to pick. To do this, we can run a *nested cross-validation* (NCV) inside the training set. Consider our illustrative example, which had 6 groups of examples based on epochs. During the first cross-validation fold, we tested on group 1 anded train on groups 2-5. Using a nested cross-validation means performing the cross-validation procedure described above inside groups 2-5 by testing on 2 and training on 3-5, then testing on 3 while training on 2,4,5,6, etc. The result for this training set is, again, #`methods` $\times$ #`voxels` table of results. For each method, this table will yield the number of voxels at which the accuracy is highest; this is the number that will be select from the ranking produced from groups 2-5. The end product is a #`methods` $\times$ 1 vector of accuracy results, as shown in Table 1 (right) for our sample dataset.

3.4 Evaluating results

The person training a classifier on fMRI data is concerned with establishing that a variable of interest can be decoded from it, i.e. a classifier can be trained whose true accuracy is better than that of a classifier deciding at random. A *significant* result is thus one where one can reject the null hypothesis that there is no information about the variable of interest. Establishing significance is generally done by determining how improbable a classification results would be were the null hypothesis to be true (this probability is called a p-value).

The accuracy on the test set is an estimate of the *true accuracy* of this classifier. Formally, this is the probability that it will correctly label a new example drawn at random from the same distribution that the other examples came from; it can also be viewed as the accuracy one would get if one had an infinite number of examples in the test set. Hence, how good an estimate it is depends on the size of the test set. It is an unbiased estimate because the test examples have not been seen by the classifier during training.

In this situation the probability of obtaining the result under this null hypothesis is easy to calculate. It can be viewed as the outcome of tossing a coin for each of the examples to be classified, with the coin's bias reflecting the probability of labelling an example correctly by chance (50% in a two class problem, 25% in a four class problem, etc). Thus, the p-value under this null hypothesis of k correctly labelled test examples is, if we define X to be the number of correctly labeled examples, simply $P(X \geq k)$ under a binomial distribution with n trials and probability of success 0.5 (two class), 0.25 (four class), etc; if the p-value is below a certain threshold (typically 0.05 or 0.01) the result will be declared significant.

The accuracy estimate obtained through cross-validation is a combination of results produced by classifiers trained on mostly overlapping training sets, hence we will treat that estimate as if it came from a single classifier, following common practice [23]. This allows us to test it in exactly the same way.

Extending the significance test to multiple, independent results is straightforward, in that one can simply resort to a multiple comparison correction criterion and adjust the significance threshold accordingly. This is the situation we have when we consider the nested cross-validation (NCV) results, as each one is the product of training a classifier using voxels selected with a different method. If, instead, we are just producing a #`methods` $\times$ #`voxels` table of results, we could use the same approach to simultaneously test all the results in the table. This would, however, be conservative, in that the results for a given method are correlated to some extent; this follows from the fact that the half the voxels used by the classifier based on the top 100 voxels are also used by the one based on the top 50, say. A common idea to deal with this issue is to select the maximum result among all those obtained using a particular selection method (a table row); it is often accompanied by the mistake of testing that maximum result for significance in the way described. To see why this can be a problem, consider a situation where the classifiers are using different numbers of features and, in each case, have a true accuracy (over the underlying example distribution) of 0.5. It is possible even in this situation that, at least for one number of features, the experimentally observed accuracy will be fairly high; this becomes more likely the smaller the dataset is. If we are just deeming the 'best' number of features to be the one at which the maximum accuracy is attained, we would pick this result and possibly deem it significant. Hence one possibility is to report the entire table and the number of examples used, and correct for all of them, at the risk of being conservative. The other possibility is to report the maximum accuracy result and test it differently using a permutation test, as follows.

Assuming there is no class information in the data, the labels can be permuted without altering the expected accuracy using a given classifier and number of features (this would be chance level). This is exactly the kind of situation where one can use a permutation test, by repeating the entire process of producing a result table from a dataset with permuted labels (see [9], which reviews the use of permutation tests in classification with one functional neuroimaging application, and also Appendix C for more details). Over many permutations this yields a sample of accuracy results for each cell in the table under the null hypothesis but also, more importantly, for any derived measures such as the maximum of each table row. The p-value under the null hypothesis for each entry of the original result table is the fraction of the corresponding sample that is greater than or equal to it, and this applies to the maximum as well. The main problem of resorting to this approach is its computational cost.

4 Applications of pattern classification

We have been treating a classifier-based analysis as a sequence of stages with choices at each stage. One choice that we have not yet directly addressed is perhaps the most important: the choice of the initial scientific question to be answered. Our focus so far has been on the use of classifiers for *pattern discrimination*. Here, the basic question being addressed is simply whether fMRI data carry information about a given variable of interest. Determining the answer to this question is of interest, for example, in clinical diagnosis or lie detection [5]. We can view traditional univariate methods as answering this question on a voxel by voxel or ROI by ROI basis. Classifiers bring to the question an increase in sensitivity of detection, both by pooling information across many voxels but also by relaxing constraints on spatial contiguity or the need for voxel responses to be similar. In addition to the question of whether fMRI data carry class information, there are at least two other basic questions to which classifiers can be applied, namely 'where or when is class information represented' (pattern localization), 'how is class information encoded' and 'how does its encoding relate to known relationships between stimuli' (pattern characterization). This section examines the extent to which questions these questions can be tackled using the machinery of linear classifiers and feature selection, as well as the limitations and caveats. For the rest of the section we will assume features are voxels, as it will simplify discussion.

4.1 Pattern Localization

Once it has been established that class information is present in a dataset, one may go on to ask *where* in the brain this information resides. A natural approach for addressing this is to ask which voxels contribute most strongly and reliably to the classifier's success. There are two aspects to this: determining which voxels are being selected and also how their weight affects the classifier prediction.

When cross-validation is being used, a particular voxel selection method can and very likely will rank different voxels at the top from fold to fold. It may come as a surprise that, for many datasets, the overlap between the set of voxels selected in all folds versus the set selected in an individual fold is fairly small (between $\frac{1}{10}$ and $\frac{1}{3}$ is typical) even if selecting a relatively large fraction of the total number of voxels [35]. This is not a problem, intrinsically, it just reflects the fact that whatever voxels are selected contain *sufficient* information. There might be redundancy, in that many voxels encode the same things and a few get picked in each fold, and there might be a small *necessary* subset, present in all folds. One possible strategy for finding the maximal subset of voxels containing information would be to start from the ones present in all or almost all folds and use them as seeds in a clustering algorithm that would find voxels with similar response profiles across examples.

A related issue is whether the information carried by voxels is the same in the sets identified by various selection methods. There are two ways of considering this issue: one is to look at overlap in selected sets, now between methods, and the other to consider whether a given classifier using different voxel sets performs differently (in particular, whether it makes different prediction errors). For problems with two classes, most selection criteria will find similar voxels, as there aren't many ways that a voxel could respond differently to the two classes. In multiclass problems results suggest there are different types of voxel behaviour and the various voxel selection methods described earlier can pick on different types [35]. Hence, if we wish to use classifier prediction success as an indication that the voxel set it uses contains informative voxels, it is wise to consider different selection methods; it is also a good idea to determine whether there is any overlap between the voxel sets selected in all folds by each method.

Given a set of voxels, a different question is which of those voxels most strongly affect classifier predictions. The use of linear classifiers makes it relatively simple to answer this question, as each voxel affects the decision only through the magnitude of its weight, and not though the interaction with other voxels. Appendix A explains how to get these weights for the various linear classifiers, as well as for nearest neighbour classifiers with particular distance measures (although in this case the weights are test example specific, rather than the same for any example), and also shows what a typical image of voxel weights looks like. Note that you can take this further by multiplying the weights by the average example of each class, giving you class-specific images.

A practical issue if using cross-validation is that, again, we have a different set of voxels and thus a different classifier in each fold. A heuristic approach is to group all the voxels that were selected in at least one of the folds and run the cross-validation again on that set; if the accuracy is comparable to what was obtained with the original cross-validation, the classifier weights can be averaged across folds and considered. Alternatively, if the accuracy using all the voxels available is high – more common in problems with few classes - we can just consider the weights over all those voxels.

One point to be aware concerning linear classifiers is that the regularization for LR or linear SVMs will cause the weights on correlated voxels to go down proportionally to the number of correlated voxels. This is beneficial when the correlation is mostly driven by a noise source and is the reason why, in general, both types of classifier outperform GNB if all voxels are used [35] (though not if coupled with voxel selection). However, it is quite

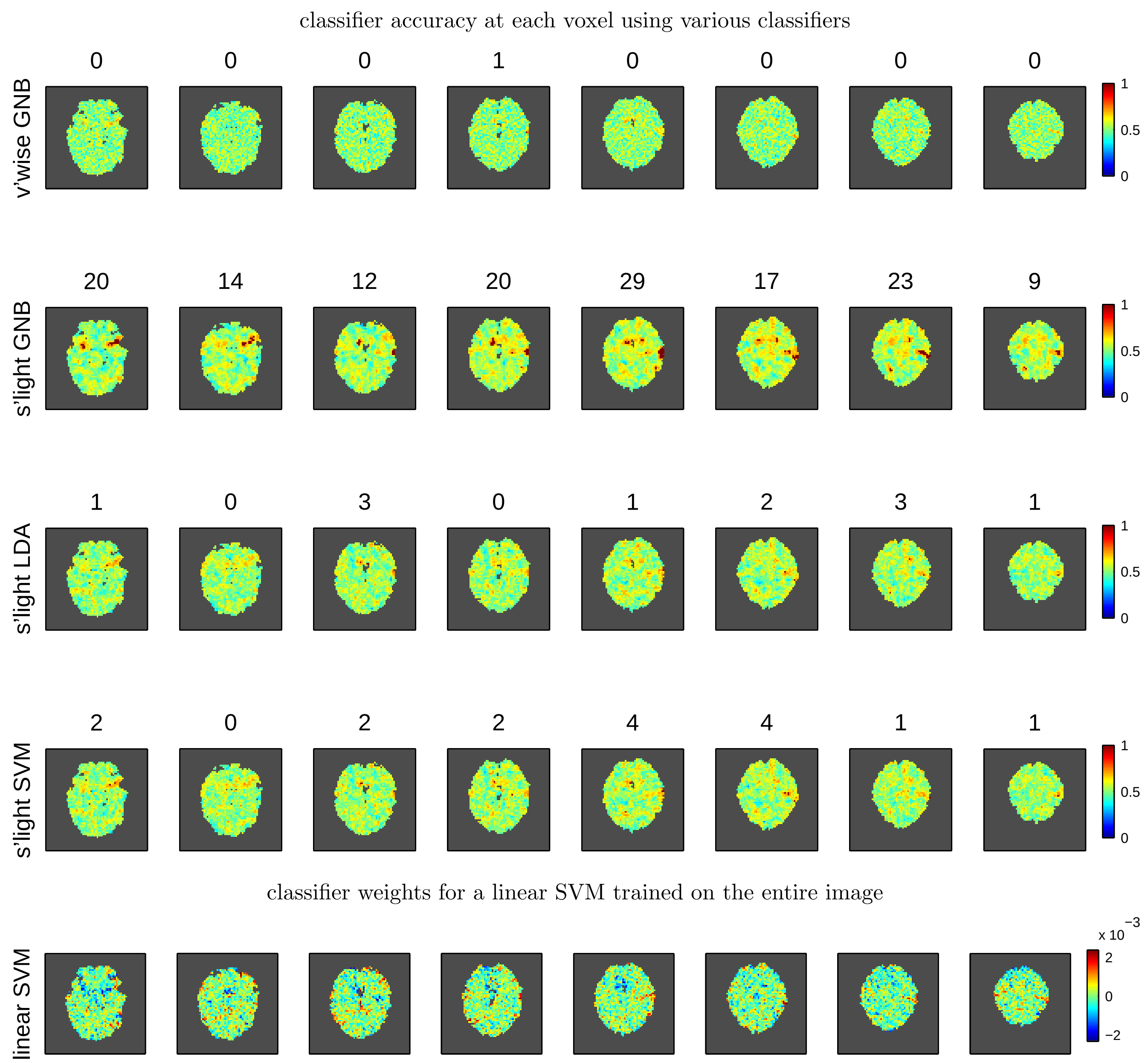


Figure 4: **top:** Comparison of accuracy maps obtained with 6-fold cross-validation. Each row contains eight slices from an accuracy map, inferior to superior, top of each slice is posterior. The classifiers used in each row are single voxel GNB and radius 1 neighbourhood searchlight GNB, LDA and linear SVM. On top of each slice is the number of voxels for which accuracy was deemed significant using FDR $q = 0.01$, which are highlighted in dark red. **bottom:** Classifier weights for a linear SVM classifier trained on the entire image.

possible that there will be voxels that contribute to the decision but whose weights are smaller than those of other voxels contributing voxels.

Note that, in addition to asking *where* class information is present, we can also ask *when* classes are represented – temporal as opposed to spatial localization. This discussed less commonly in the literature and appears in two different guises. The first is to either select a feature space corresponding to voxels at all time points in a trial or simply restrict the time points data is extracted from, as in [29] or [31]. A more sophisticated approach is used in [37], where classifiers are trained on data where classes are cleanly separated between blocks and then deployed as 'sensors' of emergence of class-specific patterns in event-related test data.

The issues outlined make localization of information to specific voxels less straightforward than one might wish and, furthermore, the use of linear classifiers means that no nonlinear relationships between voxels can be learned. There is, however, an alternative approach to answering the localization question, 'information-based functional brain mapping' [24], which can ameliorate these problems and also address characterization questions to some extent.

A general version of this idea can be summarized as training classifiers on many small voxel sets which, put together, cover the whole brain. One natural candidate for this is to train a distinct classfier for each voxel, using only the voxels spatially adjacent to it. This is akin to shining a 'searchlight' on every voxel neighbourhood in succession, and is often referred to as training 'searchlight classifiers.' Common variants for the classifier are versions of gaussian classifiers (GNB, LDA) or linear SVM (the original paper used the Mahalanobis distance, which is part of the multivariate gaussian distribution in LDA). The result is a brain map where, for every voxel, we have the cross-validation accuracy of a classifier trained on the neighbourhood surrounding it (typically, the 26 neighbours in 3D, although a wider radius could be considered). This accuracy can be converted into a p-value using the analytical method in Section **??** or by using a permutation test as described in Section 3.4. In either case, we will obtain a p-value map that can be thresholded using a multiple comparison correction criterion, to identify voxel neighbourhoods of significant accuracy.

We again have a choice of which classifier to use, although in this case the tradeoffs are different. Given the relatively small number of voxels in a neighbourhood, it is now feasible to train more complex classifiers. LDA is the first obvious choice, in that a covariance matrix of voxels captures relationships between them, but one could also attempt a nonlinear classifier if more complex relationships are hypothesized. This will possibly be more advantageous if there are more than two classes, although with the caveat that it is unlikely that activation distinguishing *all* the classes from each other can be found in a small region.

We provide a comparison of single voxel GNB and searchlight GNB, LDA and linear SVM in Figure 4 for our illustrative study. Whereas there are reassuring similarities in the location of voxels deemed significantly informative, there are also interesting differences between the maps which we discuss in Appendix B. For reference, the figure also provides the weights of a linear SVM classifier on each voxel.

4.2 Pattern Characterization

Beyond asking whether and where class information is represented, classifiers can also be used to answer the question of *how* classes are represented in the brain. It is perhaps here that classifiers and other machine learning methods stand to be of greater value.

The work on information mapping in the previous section can also be considered from the point of view of pattern characterization. Each neighbourhood classifier attempts to find relationships between the voxels in a small set that contain information. As said earlier, given the small number of voxels, we have the possibility of training nonlinear classifiers to learn fairly complex relationships. But just how complex should one expect these relationships to be? One extreme in the range of answers is the situation in [20], where voxels are barely more sensitive to one class of stimulus than another, and thus do not reflect an encoding of those stimuli. There is no advantage in training a classifier that does more than weigh a 'vote' of each voxel to pool information. The other extreme is what [13] calls a combinatorial code: the information is in the complex combination of voxel activities in different classes. In between lie situations where voxels have very selective responses (e.g. FFA or PPA in [21]) or voxels are selective for groups of classes (e.g. semantic category classes that correspond to animate things versus those that do not).

Thus far, we chose to describe the work done resorting to linear classifiers or information mapping, both because it is simple and a good starting point to address characterization questions. There are, however, several recent papers that tackle this kind of question in more sophisticated ways that elude a unified view, and hence we will cover them only briefly. The common points between them are the wish to correspond relationships between stimulus with those in fMRI data, either descriptively and actually changing classifiers to reflect domain information, such as such as models of neuron populations behaviour or psychophysics division of the stimulus space. On the first camp, [13] shows that a neural network trained to distinguish examples of eight semantic categories learns a representation on its hidden layer that captures a plausible taxonomy of the classes and [33], on the same dataset, that the ability to distinguish the category stimuli correlates with the ability of the classifier to distinguish the corresponding classes. [2] shows that that the similarity matrix of phonetic stimuli corresponds to the similarity matrix of patterns of activation while processing them and [1] relates perceptual similarity of stimuli to the structure of their neural representation. [18] introduces hidden process models, a technique to identify multiple cognitive processes present simultaneously in the brain with overlapping spatial activation, taking advantage of domain information and class labels to constrain their causal ordering.

Finally, two recent studies leverage domain knowledge to predict fMRI activation for new stimuli, something which can be used to classify by comparing the predicted activation to the activation of stimuli in the training set. [22] does this for visual stimuli, using a model of the responses of neuron populations in the visual cortex and predicting the fMRI signal from the model populations for each voxel. [30] learns a model that predicts fMRI activation for new words, based on their similarity in a space of semantic features to words for which fMRI activation is known.

5 Conclusions

In this paper we have described the various stages in a machine learning classifier analysis of fMRI data. Aside from discussing the choices available at each analysis stage, their interactions and the practical factors conditioning them, we explored the use of this kind of analysis to answer three types of scientific question. These are 'is there information about a variable of interest' (pattern discrimination), 'where is the information' (pattern localization) and 'how is that information encoded' (pattern characterization). We purposefully focused on linear classifiers and the discrimination and localization questions, as these are both better understood and the ones most readers are likely to want to answer first. Space constraints meant that some important practical considerations had to be left out of the main text. Various appendices are provided in a longer version of this paper available at `http://www.cs.cmu.edu/~fpereira/research.html`. Software for performing a classifier analysis (the Princeton Multi-Voxel Pattern Analysis toolbox) can be found at `http://www.csbmb.princeton.edu/mvpa`, together with tutorials and a very responsive online community.

We are well aware that the current frontier of research is in pattern characterization, and provided pointers to what we think some of the most thought-provoking work in this area. We are confident that, regardless of what progress may come, it will have machine learning methods at its root, classifier-based or otherwise.

References

[1] G. K. Aguirre. Continuous carry-over designs for fmri. *NeuroImage*, 35:1480–1494, 2007.

[2] M. M. Botvinick and L. M. Bylsma. Imaging phonological representations and their similarity structure with pattern-analytic fmri. *NeuroImage*, 26, suppl. 1, 2005.

[3] V. Calhoun, T. Adali, L. Hansen, J. Larsen, and J. Pekar. Ica of functional mri data: an overview. In *4th international symposium on Independent Component Analysis and Blind Signal Separation (ICA 2003)*, 2003.

[4] T. A. Carlson, P. Schrater, and S. He. Patterns of activity in the categorical representations of objects. *Journal of Cognitive Neuroscience*, 5(15):704–717, 2003.

[5] C. Davatzikos, K. Ruparel, Y. Fan, D. G. Shen, M. Acharyya, J. W. Loughead, R. C. Gur, and D. D. Langleben. Classifying spatial patterns of brain activity with machine learning methods: application to lie detection. *Neuroimage*, 28(3):663–8, 2005.

[6] G. J. Detre. Defining voxelsets using rich psychological models. Master's thesis, Psychology Department, Princeton University, Princeton NJ, 2006.

[7] T.G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[8] C. R. Genovese, N. A. Lazar, and T. E. Nichols. Thresholding of statistical maps in functional neuroimaging using the false discovery rate. *NeuroImage*, 15:870–878, 2002.

[9] P. Golland and B. Fischl. Permutation tests for classification: Towards statistical significance in image-based studies. In *Proceedings of IPMI: International Conference on Information Processing and Medical Imaging*, pages LNCS 2732:330–341. Springer, 2003.

[10] Phillip Good. *Permutation, Parametric and Bootstrap Tests of Hypotheses.* Springer, New York, 2005.

[11] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[12] LK Hansen et al. Generalizable patterns in neuroimaging: How many principal components? *Neuroimage*, ?:?, 1999.

[13] S. J. Hanson, T. Matsuka, and J. V. Haxby. Combinatorial codes in ventral temporal lobe for object recognition: Haxby(2001) revisited: is there a face area? *Neuroimage*, 23, 2004.

[14] T. Hastie, R. Tibshirani, and J. H. Friedman. *The elements of statistical learning: data mining, inference and prediction.* Springer-Verlag, 2001.

[15] 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 tempora l cortex. *Science*, 293(28 September):2425–2430, 2001.

[16] J. Haynes and G. Rees. Predicting the stream of consciousness from activity in human visual cortex. *Current Biology*, 16:1301–1307, 2005.

[17] J. Haynes and G. Rees. Decoding mental states from brain activity in humans. *Nature Reviews Neuroscience*, 7:523–34, 2006.

[18] R. Hutchinson, T. M. Mitchell, and I. Rustandi. Hidden process models. In *International Conference on Machine Learning*, 2006.

[19] T. Joachims. Making large-scale svm learning practical. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1999.

[20] Y. Kamitani and F. Tong. Decoding the visual and subjective contents of the human brain. *Nature Neuroscience*, 8:679–685, 2005.

[21] N. Kanwisher. The ventral visual object pathway in humans: Evidence from fmri. In L. Chalupa and J. Werner, editors, *The Visual Neurosciences*. MIT Press, 2003.

[22] K. N. Kay, T. Naselaris, R. J. Prenger, and J. L. Gallant. Identifying natural images from human brain activity. *Nature*, 452:352–355, 2008.

[23] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence*, 1995.

[24] N. Kriegeskorte, R. Goebel, and P. Bandettini. Information-based functional brain mapping. *Proceedings of the National Academy of Sciences of the USA*, 103:3863–3868, 2006.

[25] J. Langford. Tutorial on practical prediction theory for classification. *Journal of Machine Learning Research*, 6:273–306, 2005.

[26] O. Ledoit and M. Wolf. Improved estimation of the covariance matrix of stock returns with an application to portfolio selection. *Journal of Empirical Finance*, 10 (5):365–411, 2003.

[27] A. R. McIntosh and N. J. Lobaugh. Partial least squares analysis of neuroimaging data - applications and advances. *NeuroImage*, 23:250–263, 2004.

[28] T. M. Mitchell. *Machine Learning*. McGraw Hill, New York, 1997.

[29] T. M. Mitchell, R. Hutchinson, R. S. Niculescu, F. Pereira, X. Wang, M. Just, , and S. Newman. Learning to decode cognitive states from brain images. *Machine Learning*, 57:145–175, 2004.

[30] T. M. Mitchell, S. V. Shinkareva, A. Carlson, K. Chang, V. L. Malave, R. A. Mason, and M. A. Just. Predicting human brain activity associated with the meanings of nouns. *Science*, 320:1191–1195, 2008.

[31] J. Mourao-Miranda, K. Friston, and M. Brammer. Dynamic discrimination analysis: a spatial-temporal svm. *NeuroImage*, 36:88–99, 2007.

[32] K. Norman, S. M. Polyn, G. Detre, and J. V. Haxby. Beyond mind-reading: multi-voxel pattern analysis of fmri data. *Trends in Cognitive Sciences*, 10, 2006.

[33] A. J. OToole, F. Jiang, H. Abdi, and J. V. Haxby. Partially distributed representations of objects and faces in ventral temporal cortex. *Journal of Cognitive Neuroscience*, 17(4):580–590, 2005.

[34] A. J. OToole, F. Jiang, H. Abdi, N. Penard, J. P. Dunlop, and M. A. Parent. Theoretical, statistical, and practical perspectives on pattern-based classification approaches to functional neuroimaging analysis. *Journal of Cognitive Neuroscience*, 19:1735–1752, 2007.

[35] F. Pereira. *Beyond Brain Blobs: Machine Learning Classifiers as Instruments for Analyzing Functional Magnetic Resonance Imaging Data*. PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, 2007.

[36] F. Pereira and G. Gordon. The support vector decomposition machine. In *International Conference on Machine Learning*, 2006.

[37] S. M. Polyn, V. S. Natu, J. D. Cohen, and K. A. Norman. Category-specific cortical activity precedes recall during memory search. *Science*, 310:1963–1966, 2005.

[38] SPM. Statistical parametric mapping. Technical report, Functional Imaging Laboratory, UCL (http://www.fil.ion.ucl.ac.uk/spm).

[39] S. C. Strother. Evaluating fmri preprocessing pipelines. *IEEE Engineering in Medicine and Biology Magazine*, 25 (2):27–41, 2006.

[40] S. C. Strother, J. Anderson, L. K. Hansen, U. Kjems, R. Kustra, J. Siditis, S. Frutiger, S. Muley, S. LaConte, and D. Rottenberg. The quantitative evaluation of functional neuroimaging experiments: The npairs data analysis framework. *NeuroImage*, 15:615–624, 2002.

[41] Larry Wasserman. *All of Statistics*. Springer, New York, 2004.

Online supplementary material

A Linear classifiers

As discussed earlier, a linear classifier with a set of weights $\mathbf{w}$ applied to an example $\mathbf{x}$ with V features

$$\mathbf{xw} = x_1 w_1 + \ldots + x_V w_V$$

predicts class A if $\mathbf{xw} > 0$ or class B if $\mathbf{xw} < 0$ (ties broken randomly), in a two class situation. Technically we would also need a bias term w_0, but this can be included in the weight vector $\mathbf{w}$ if an extra 1 entry is added to the example $\mathbf{x}$ in the corresponding position. As discussed in Section 3.2 different classifiers vary in how those weights are set, and whether they are set explicitly or follow as a consequence of optimizing another criterion. The sections that follow describe both the criteria optimized and how weights may be obtained for the classifiers we covered earlier. Good references on this topic are [28] and [14].

A.1 Discriminative

A.1.1 Logistic Regression

Logistic Regression (LR,[14]) models $P(\mathtt{class_k}|\mathbf{x})$ as

$$P(\mathtt{class_k}|\mathbf{x}) = \frac{\mathtt{exp}(\mathbf{x}'\mathbf{w_k} + w_{0,k})}{1 + \sum_{j=1}^{K-1} \mathtt{exp}(\mathbf{x}'\mathbf{w_j} + w_{0,j})}$$

for classes $1, \ldots, K-1$ and

$$P(\mathtt{class_k}|\mathbf{x}) = \frac{1}{1 + \sum_{j=1}^{k-1} \mathtt{exp}(\mathbf{x}'\mathbf{w_j} + w_{0,j})}$$

for class K and fits parameters $\mathbf{w_j}$ to maximize the log conditional likelihood $\sum_{i=1}^{n} log(P(y_i|\mathbf{x_i}))$, where $\mathbf{x_i}$ is the i^{th} example and y_i is its label.

Unless the features are linearly independent - clearly not the case, since at the very least we can expect many neighbouring voxels to be correlated - it is generally necessary for stability reasons to regularize the solutions $\mathbf{w_k}$ by adding a term such as $\lambda\|\mathbf{w_k}\|_2$ (L2 regularization) or $\lambda\|\mathbf{w_k}\|_1$ (L1 regularization) to the log conditional likelihood to be maximized, for each class k; The former will lead to a vector of small weights, whereas the latter will lead to a sparse weight vector.

At classification time, the predicted class k for example $\mathbf{x}$ is $\mathtt{max_k} P(\mathtt{class_k}|\mathbf{x})$. Given that the expressions for $P(\mathtt{class_k}|\mathbf{x})$ above all have the same denominator, we can consider just the numerators for the purpose of determining the maximum. From that perspective, the problem can be viewed as the learning m-dimensional discriminants $\mathbf{w_k}$ for each class such that $\mathtt{max_k}\ (\mathbf{x}'\mathbf{w_k} + w_{0,k})$ (plus the regularization term) is the predicted class label.

A.1.2 Linear Support Vector Machine

A linear Support Vector Machine (SVM,[19]) learns a m-dimensional discriminant $\mathbf{w}$ to minimize

$$\|\mathbf{w}\|_2^2 + \lambda \sum_{i=1:n} h(y_i \mathbf{x_i}'\mathbf{w})$$

where h is the hinge loss function

$$h(\rho) = \begin{cases} 0 & \rho \geq 1 \\ 1-\rho & \text{otherwise} \end{cases}$$

and where y_i is either -1 or 1.When classifying a new example, we apply each of the binary problem discriminants $\mathbf{w_k}$ to it and predict the label k such that $\mathtt{max_k}\ \mathbf{x}'\mathbf{w_k} + w_{0,k}$. This discriminant differs from the one learnt by LR in that it maximizes the margin of the discriminating hyperplane, but is similar in that it is regularized using the L2 norm of the weight vector.

If the classification problem has more than two classes it generally has to be converted into several binary problem. One way of doing this is to use a *one versus rest encoding*, where one binary problem is created for each class. That problem is created by assigning one label to all the examples in class k and another to examples in all other classes; this is done for all K classes, yielding K binary classification problems. Another possibility is to use a *all versus all* encoding, in which one classifier is trained for each of all possible binary problems involving a pair of classes. A test example label is predicted by applying each binary classifier, recording a 'vote' for the winning class for that pair and, after doing this for all pairs, picking the class with the most votes. An alternative that seems to empirically lead to better results is to use the binary problems created via an error-correcting output code [7], although this is a more complicated than either of the two procedures above.

A.2 Generative

A.2.1 Bayes Classifiers and Gaussian Naive Bayes

Bayes Rule tells us how to predict one of k classes from an example $\mathbf{x}$

$$P(\texttt{class}_\texttt{k}|\mathbf{x}) = \frac{P(\mathbf{x}|\texttt{class}_\texttt{k})P(\texttt{class}_\texttt{k})}{P(\mathbf{x})}$$

in terms of the class conditional distribution $P(\mathbf{x}|\texttt{class}_\texttt{k})$.

The classifier obtained by assuming that that distribution is multivariate gaussian is the rule in Linear Discriminant Analysis (LDA), if we assume all classes share the same covariance matrix, or Quadratic Discriminant Analysis (QDA), if not [14].

If, furthermore, we assume that the class conditional distribution of each feature is independent of the others, i.e. $P(\mathbf{x}|\texttt{class}_\texttt{k}) = \prod_{j=1}^{m} P(x_j|\texttt{class}_\texttt{k})$, then we have a Gaussian Naive Bayes (GNB) classifier [28]. In addition, we will assume that the variance of each feature is the same for all class conditional distributions. This assumption seems reasonable in fMRI data (if the noise in each voxel is independent of its mean condition activation) and appears to lead to more robust estimates and better classifier accuracy by letting us pool data from multiple classes. Given this assumption, GNB is equivalent to LDA with a diagonal covariance matrix. The parameters to learn are the mean and variance (μ and σ^2) in the normal distribution density function for each class (e.g. class A) $f(x|\mu_A, \sigma_A^2) = \frac{1}{\sqrt{2\pi\sigma_A^2}} exp(-\frac{1}{2}(\frac{x-\mu_A}{\sigma_A})^2)$, with $\mu_A = \frac{\sum_{i=1}^{n_A} x_i}{n}$ (index ranges over x_i with class A only) and $\sigma_A^2 = \sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \mu_{x_i})^2}{n}$ (index ranges over x_i in all classes, subtracting the respective class mean).

LDA/QDA have been used in fMRI before [40],[4], though always preceded by a dimensionality reduction process (generally a singular value decomposition) in order to train the classifier having fewer dimensions than examples and be able to estimate a covariance matrix. Even in a reduced dimensionality situation it might be problematic to do this estimation; in such cases, a shrinkage estimator can be used instead of the usual maximum likelihood estimator (see [26], if shrinking towards a diagonal matrix the result will be more like GNB).

A.2.2 GNB as a linear classifiers

Both LR and SVM are generally expressed as linear discriminants with particular regularizations, but so can GNB with the class-shared variance estimates. To see how, consider a situation with two classes, A and B (this can be generalized to k classes by considering each of $k-1$ against the k^{th}).

Classification of a new example $\mathbf{x}$ is done by comparing the posterior probabilities for each class, $P(A|\mathbf{x})$ and $P(B|\mathbf{x})$, and picking the class for which that posterior probability is highest. Equivalently, one could also say we are considering the *odds ratio* $R = \frac{P(A|\mathbf{x})}{P(B|\mathbf{x})}$ and deciding A if the ratio is greater than 1 or B otherwise. By taking the logarithm of this ratio and assuming equal prior probabilities over both classes ($P(A) = P(B)$), we can rewrite it as follows:

$$R = log\frac{P(A|\mathbf{x})}{P(B|\mathbf{x})} = log\frac{P(\mathbf{x}|A)P(A)}{P(\mathbf{x}|B)P(B)} = log\frac{P(\mathbf{x}|A)}{P(\mathbf{x}|B)}$$

and the decision rule becomes 'select class A if $R > 0$ or class B if $R < 0$'.

Applying the naive bayes assumption, R is turned into a sum of per-feature ratios

$$R = \Sigma_{j=1}^{n} log\frac{P(x_j|Y=A)}{P(x_j|Y=B)} = \Sigma_{j=1}^{n} R_j(x_j, \mu_{Aj}, \sigma_{Aj}, \mu_{Bj}, \sigma_{Bj})$$

each of which is a function of the feature value x_j and the means and standard deviations for the two class conditional feature densities. Each ratio R_j is a fraction of two normal distribution probability density functions, and so can be rewritten as:

$$R_j(x_j, \mu_{Aj}, \sigma_{Aj}, \mu_{Bj}, \sigma_{Bj}) = log\frac{\frac{1}{\sqrt{2\pi\sigma_{Aj}^2}} exp(-\frac{1}{2}(\frac{x_j-\mu_{Aj}}{\sigma_{Aj}})^2)}{\frac{1}{\sqrt{2\pi\sigma_{Bj}^2}} exp(-\frac{1}{2}(\frac{x_j-\mu_{Bj}}{\sigma_{Bj}})^2)}$$

Given that we also assume that the class conditional distributions for each feature have the same variance, the ratio R_j simplifies to the expression

$$R_j(x_j, \mu_{Aj}, \mu_{Bj}, \sigma_j) = \frac{1}{2\sigma_j^2}(2(\mu_{Aj} - \mu_{Bj})x_j + \mu_{Bj}^2 - \mu_{Aj}^2) = \frac{\mu_{Aj} - \mu_{Bj}}{\sigma_j^2}x_j + \frac{\mu_{Bj}^2 - \mu_{Aj}^2}{2\sigma_j^2}$$

where x_j stands by itself. The decision can thus be expressed in terms of a linear discriminant w such that $w_j = \frac{\mu_{Aj}-\mu_{Bj}}{\sigma_j^2}$ and $w_0 = \sum_{j=1}^{m} \frac{\mu_{Bj}^2-\mu_{Aj}^2}{2\sigma_j^2}$.

A.3 k-nearest neighbours

A.3.1 nearest neighbour classification

'Nearest neighbour' is the simplest type of classifier, in that no parameters are learnt. Classification of a test example is done by finding the training set example that is closest to it by some measure (e.g. euclidean distance considering the entire feature vector) and assigning its label to the test example.

Given row vectors $\mathbf{a}$ and $\mathbf{x}$ with m features, and their means μ_a,μ_b and standard deviations σ_a,σ_b, the distances/similarities most commonly used are

$$\begin{aligned}
\texttt{euclidean}^2(\mathbf{a},\mathbf{x}) &= (\mathbf{a}-\mathbf{x})(\mathbf{a}-\mathbf{x})' \\
\texttt{cosine}(\mathbf{a},\mathbf{x}) &= \frac{\mathbf{a}\mathbf{x}'}{\|\mathbf{a}\|\|\mathbf{x}\|} = \bar{\mathbf{a}}\bar{\mathbf{x}}' \\
\texttt{correlation}(\mathbf{a},\mathbf{x}) &= \frac{(\mathbf{a}-\mu_a)(\mathbf{x}-\mu_x)'}{(m-1)\sigma_{\mathbf{a}}\sigma_{\mathbf{x}}} = \frac{\bar{\mathbf{a}}\bar{\mathbf{x}}'}{m-1}
\end{aligned}$$

where $\bar{\mathbf{x}}$ is a normalized version of the data vector $\mathbf{x}$ (making it norm 1 for cosine similarity or z-scored, i.e. mean 0 and standard deviation 1, for correlation similarity).

A.3.2 obtaining feature weights from nearest neighbour classification

Given the closest training set example (or prototype) for each of two classes, $\mathbf{a}$ and $\mathbf{b}$, the label predicted for a new example $\mathbf{x}$ is a function of the sign of $\texttt{similarity}(\mathbf{a},\mathbf{x}) - \texttt{similarity}(\mathbf{b},\mathbf{x})$. This expression is a linear function of $\mathbf{x}$ (assume the vectors have been already been normalized as above, in order to make the formulas simpler):

$$\begin{aligned}
\texttt{euclidean}^2(\mathbf{a},\mathbf{x}) - \texttt{euclidean}^2(\mathbf{b},\mathbf{x}) &= 2(\mathbf{a}-\mathbf{b})\mathbf{x}' + (\mathbf{a}\mathbf{a}'-\mathbf{b}\mathbf{b}') = \mathbf{a}\mathbf{a}'-\mathbf{b}\mathbf{b}' + 2\sum_{j=1}^{m}(a_j-b_j)x_j = w_0 + \sum_{j=1}^{m} w_j x_j \\
\texttt{cosine}(\mathbf{a},\mathbf{x}) - \texttt{cosine}(\mathbf{b},\mathbf{x}) &= (\mathbf{a}-\mathbf{b})\mathbf{x}' = \sum_{j=1}^{m}(a_j-b_j)x_j = \sum_{j=1}^{m} w_j x_j \\
\texttt{correlation}(\mathbf{a},\mathbf{x}) - \texttt{correlation}(\mathbf{b},\mathbf{x}) &= \frac{1}{m-1}(\mathbf{a}-\mathbf{b})\mathbf{x}' = \sum_{j=1}^{m}\frac{a_j-b_j}{m-1}x_j = \sum_{j=1}^{m} w_j x_j
\end{aligned}$$

and, given this, the classification rule is to predict class A if $\mathbf{x}\mathbf{w} > 0$ or class B otherwise. Note that, unlike the linear classifiers described earlier, there is one weight vector for each example being tested – instead of a single weight vector for all test examples – and this is not a linear classifier. What these expressions show is that the decision for a given test example is obtained by a linear combination of all the features, and each feature affects the decision only throw its weight. The absolute values of these weights can be averaged across test examples to provide a measure of how much each feature contributed to make the classification decisions for those test examples.

A.4 Non-linear classifiers

For reasons described in the main text this paper, this paper is not concerned with the use or interpretation of nonlinear classifiers. However, there is a perspective that makes their connection with linear classifiers particularly intuitive and may thus be helpful. The main idea is that nonlinear classifiers can be viewed as linear classifiers on a *new* feature space where the features are nonlinear functions of the original ones. From this perspective

- Support Vector Machines (with a kernel, [14]) - The new feature space is determined by the particular kind of *kernel* used by the SVM, e.g. if the original image had two voxels x_1 and x_2, a quadratic SVM would have features x_1, x_2 and x_1x_2 (similar to a linear regression with interaction terms). While the new feature space is not generally as intuitive for other kernels, there are kernel types for many useful kinds of functions of features in the original space.
- Neural Networks (with a hidden layer of nonlinear units, [28]) - The new features in this case are the hidden units of the network. Instead of being a deterministic function of the original feature space, as in SVMs, the mapping of old features to new features is learnt on the training set to optimize whatever loss function is desired.

.

B Feature selection and information mapping

In this section we will assume features are voxels, for clarity. Voxel selection methods produce a score of interest for each voxel and this is often a measure of information for the voxel, or voxel plus its neighbours; this means it can be used in information mapping and this is why we discuss both these issues together. useful in mapping and the reason we discuss it here.

B.1 Voxel Selection

What follows is an expanded version of the listing of voxel ranking methods in the paper, with additional details on how they can be computed.

- Activity - This method selects voxels which are active in at least one condition. It scores a voxel by the difference in its mean activity level during each task condition versus a baseline condition, as measured by a t-test. This can be a test of whether the mean is greater than 0 or of whether it is different from 0. The former makes sense in datasets normalized so that 0 is the mean value for each voxel during the baseline condition, whereas the latter would be more appropriate if the normalization is to units of percent change from mean level across all conditions. The t-test for each condition yields one ranking of voxels, the overall score of a voxel is its best position in any ranking (ties are broken at random).
- Accuracy - This method scores a voxel by how accurately a gaussian bayesian classifier can predict the condition of each example in the training set, based only on the voxel. This requires performing a cross-validation withint the training set for each voxel. The classifier learns the parameters of a gaussian distribution for the values the voxel takes, in each class, and uses that to generate predictions. The cross-validated accuracy is the voxel's score.
- Searchlight Accuracy - The same as Accuracy, but instead of using the data from a single voxel (classifier with one feature) we use the data from the voxel and its immediately adjacent neighbours in three dimensions (classifier with up to 27 features). This is a variant of a a strategy proposed by [24], and can be done using GNB, LDA or SVM, as well as more complex classifiers if we suspect there are interesting interactions between voxels. Note that if using LDA or QDA it might be necessary to resort to a more robust estimator of the local covariance matrix, especially if there are few examples (see [26]).
- ANOVA - This method looks for voxels where there are reliable differences in mean value across conditions (e.g. A is different from B C D, or AB from CD, etc). This is done using an Analysis of Variance (ANOVA) to test the null hypothesis that the mean voxel value for the various task conditions is the same. The score is the p-value (the lower, the better).
- Stability - This method picks voxels that react to the various conditions *consistently* across cross-validation groups in the training set (e.g. if the mean value in A is less than B and greater than C, is this repeated in all groups). The method requires sorting examples by groups and, within a group, ordering them by condition. This criterion uses as a score the average correlation of this ordered set of values for a voxel across all pairs of groups.

Other possible methods are simple functions of the distribution of the mean values the voxel takes across the different conditions, such as the range of that distribution (with the intuition that a large range means there are differences in activity between conditions). [35] shows that these methods generally perform worse than the ones listed above, as they are more sensitive to noise.

Finally, the methods described above are univariate, with exception of the searchlight ones. As described in Section 3.2, it is not clear that informative nonlinear relationships across voxels can be found if those voxels can come from anywhere in the brain; the searchlight methods provide some evidence that there is *local* information and a structure such as a covariance matrix (in the LDA case) can be found. Beyond this, [6] used a known relationship (cosine similarity) between stimulus classes (saccade angles) to look for voxel neighbourhoods whose patterns of activation for those classes had a similar relationship, obtaining much better accuracy than using univariate voxel selection.

B.2 Information Mapping

We provide a comparison of single voxel GNB and searchlight GNB, LDA [3] and linear SVM in Figure 4 for our illustrative study. The main results observable are that there are differences across classifiers in both sensitivity to

[3]Note that the LDA in this case has a small value added to the covariance matrix to allow inversion; the Mahalanobis distance used in [24] uses instead an estimator that shrinks towards a diagonal covariance matrix which means that, in practice, it will learn something between the GNB and LDA classifiers.

the number of examples available (contrast full with half) and the numbers of voxels with high enough accuracy to be deemed significant. This happens for the greater part in voxels where GNB can reach close to perfect accuracy. More reassuringly, the locations of significant accuracy voxels are similar across classifiers. Finally, the accuracy of searchlight classifiers is higher than that of a classifier based on a single voxel.

We again have a choice of which classifier to use, although in this case the tradeoffs are different. Given the relatively small number of voxels in a neighbourhood, it is now feasible to train more complex classifiers. LDA is the first obvious choice, in that a covariance matrix of voxels captures relationships between them, but one could also attempt a nonlinear classifier if more complex relationships are hypothesized. This will possibly be more advantageous if there are more than two classes, although with the caveat that it is unlikely that activation distinguishing *all* the classes from each other can be found in a small region.

C Nonparametric methods and classifiers

Although we provided an analytical method for testing a single classification result and there are such results for other purposes such as confidence intervals ([25] for an excellent review), we also encountered situations where it was simpler to resort to nonparametric methods (in Section 3.4, for instance). We are often asked about these and hence thought it would be worthwhile to provide a brief overview and an example in the context of our illustrative study. There are two different techniques:

- Bootstrap [41] - The bootstrap is a method for estimating the variance and distribution of a statistic, i.e. a function of a sample such as its median. It can also be used to generate a confidence interval. The main idea is to *resample* a new sample with the same size as the original one by drawing with replacement from it and computing the statistic. After many repetitions, we obtain a distribution of values for the statistic. This technique is useful when it is hard to get the distribution or a confidence interval analytically or, even if possible, those results are only valid asymptotically and we have a finite number of examples. In the case of classifiers this technique is especially relevant for asking questions that pertain to how much something would vary if we had a different *training* set, e.g. 'if we had gotten a different training set of the same size from the same distribution, would we still get a similar classifier weight on this voxel?'. It could also be used for the distribution of accuracy on the test set, though we can get a confidence interval on the accuracy by traditional means since what we are estimating is the parameter of a binomial distribution.
- Permutation test [10] - The permutation test is a method for estimating the distribution of a statistic under a null hypothesis where we can rearrange the labels on the observations without affecting the underlying distribution of possible outcomes (this is known as *exchangeability*). The main idea is to *permute* the labels on the sample and then compute the statistic. After many repetitions, we obtain a distribution of values for the statistic and can see where on that the distribution the original value of the statistic lies. If it is an extreme value, it is unlikely that we would have obtained it under the null hypothesis and it can thus be rejected. In the classifier case, this would translate to there being no difference between the distribution of examples in each class and we would permute example labels, for instance. This is the situation in Section 3.4. The power of this technique lies in the fact that, regardless of what procedure we do on the data, we can obtain a distribution for its result under the null hypothesis.

The main caveat of both techniques for our applications is that one has to be careful to respect both class balance and the structure of example groups if doing cross-validation and/or reusing the code used to produce the original results.

To illustrate the use of the bootstrap in our example study, we will consider using the bootstrap to determine whether the weights put on each voxel by a linear SVM classifier are significantly different from 0. For every resampled dataset we train a classifier and produce a map of voxel weights. Over all resamplings this produces a distribution of weight values for each voxel; the 'significant' voxels in this case are those whose distribution does not contain 0. Figure 6 shows the voxel weights learned with a linear SVM classifier, the binary mask that indicates which voxels are significant and the same voxel weights after eliminating those that are not significant.

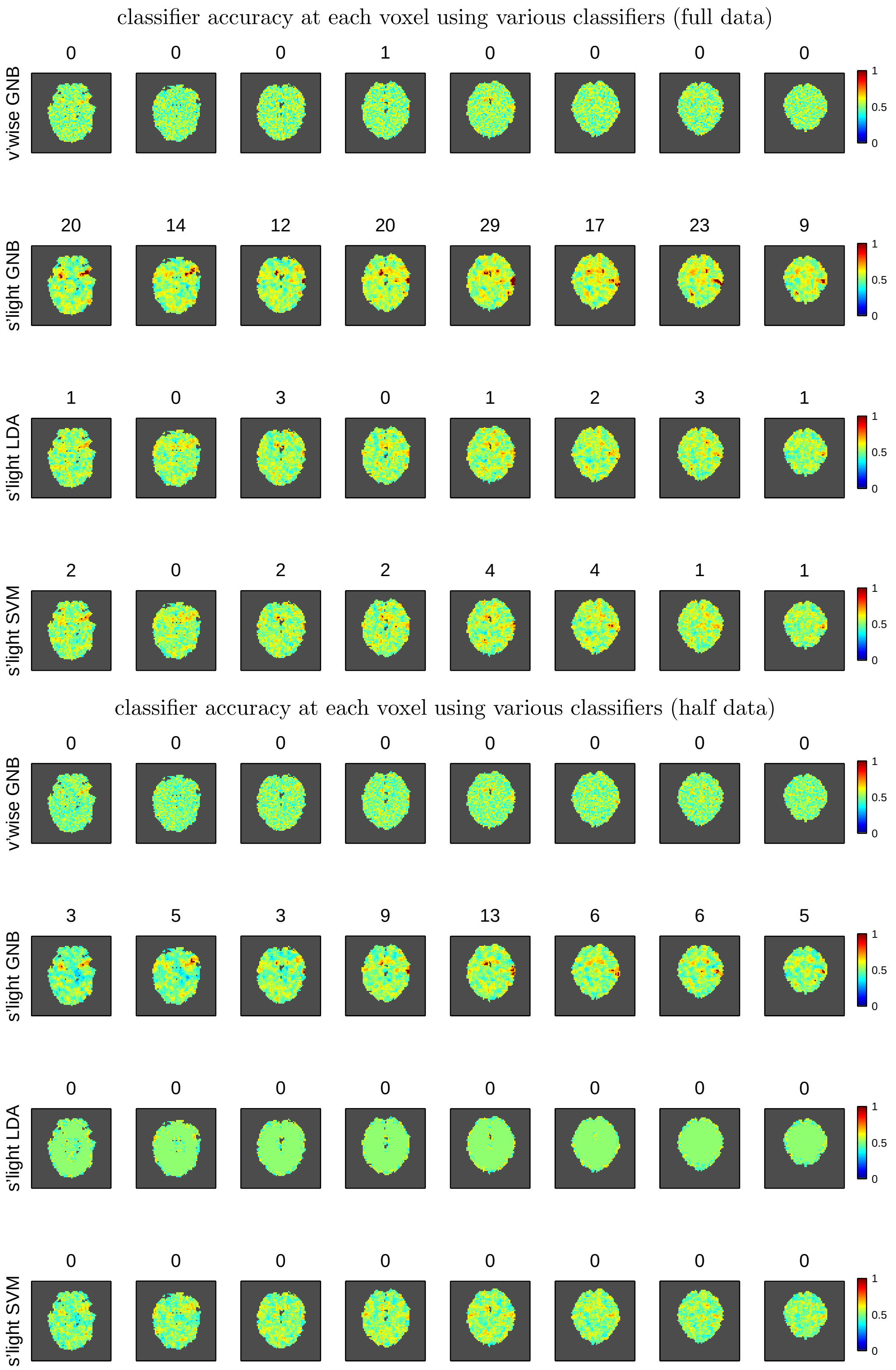


Figure 5: Comparison of accuracy maps obtained with 6-fold cross-validation (top) and 2-fold cross-validation (bottom); the former have $\frac{5}{6}$ of the data to train on, the latter only $\frac{1}{2}$. For both plots, each row contains eight slices from an accuracy map, inferior to superior, top of each slice is posterior. The classifiers used in each row are single voxel GNB and radius 1 neighbourhood GNB, LDA and linear SVM. On top of each slice is the number of voxels for which accuracy was deemed significant using FDR $q = 0.01$, which are highlighted in dark red.

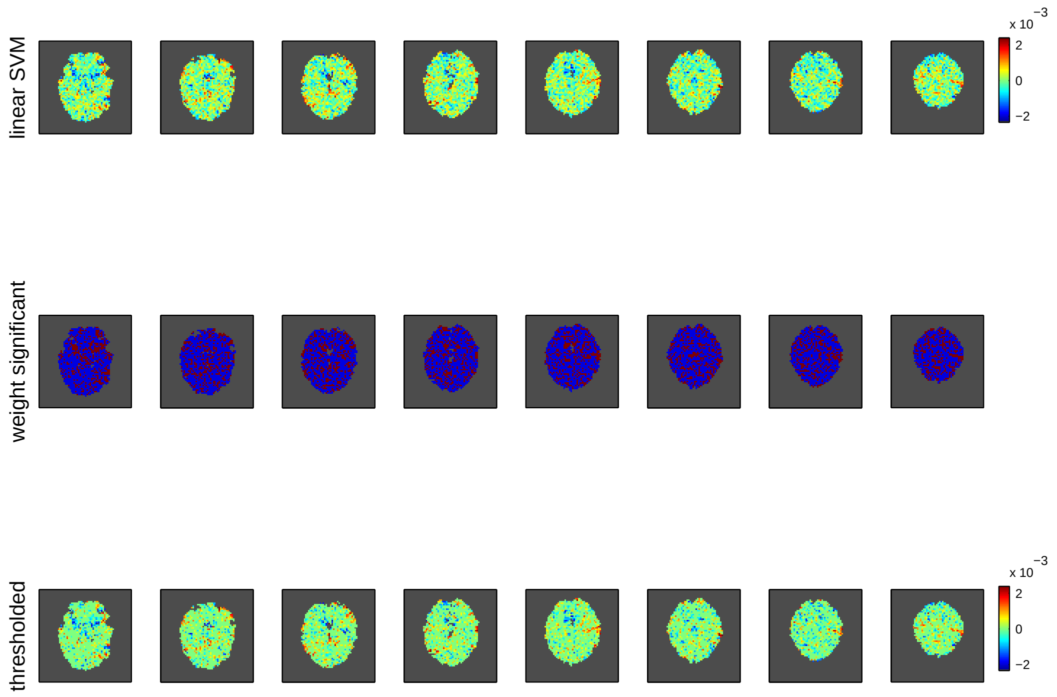


Figure 6: **top:** Voxel weights learnt by a linear SVM classifier. **middle:** Binary mask indicating which weights are significantly different from 0. **bottom:** Voxel weights after thresholding by significance.