

19 Tutorial on Pattern Classification in Cell Recording

Ethan Meyers and Gabriel Kreiman

Summary

In this chapter, we outline a procedure to decode information from multivariate neural data. We assume that neural recordings have been made from a number of trials in which different conditions were present, and our procedure produces an estimate of how accurately we can predict the labels of these conditions in a *new* set of data. We call this estimate of future prediction the “decoding/readout accuracy,” and based on this measure we can make inferences about what information is present in the population of neurons and on how this information is coded. The steps we cover to obtain a measure of decoding accuracy include: (1) designing an experiment, (2) formatting the neural data, (3) selecting a classifier to use, (4) applying cross-validation to random splits of the data, (5) evaluating decoding performance through different measures, and (6) testing the integrity of the decoding procedure and significance of the results. We also discuss additional topics, including how to examine questions about neural coding and how to evaluate whether the population is representing stimuli in an invariant/abstract way. Chapter 18 (Singer and Kreiman) discusses statistical classifiers in further detail. The ideas discussed here are applied in several chapters in this book including chapter 2 (Nirenberg), chapter 3 (Poort, Pooresmaeili, and Roelfsema), chapter 7 (Pasupathy and Brincat), chapter 10 (Hung and DiCarlo), chapter 21 (Panzeri and Ince), and chapter 22 (Berens, Logothetis, and Tolias). Chapter 20 discusses related concepts within the domain of functional imaging.

Introduction

In this chapter, we describe a procedure to decode information from multivariate neural data. The procedure is derived from cross-validation methods that are commonly used by researchers in machine learning (ML) to compare the performance of different classification algorithms (see chapter 18 for more information on

machine-learning classification algorithms). However, instead of comparing different ML algorithms, here we assess how accurately a particular algorithm can extract information about different experimental conditions in order to better understand how the brain processes information. These procedures and algorithms are extensively used to quantitatively examine the responses of populations of neurons at the neurophysiological level (e.g., chapters 2, 3, 7, 10, 21, and 22).

Our motivation for using the procedure described in this chapter is based on a simple intuition for what we believe is an important computational function that the brain must perform—namely, to *reliably* distinguish in real time between different behaviorally relevant conditions that are present in the world. Cross-validation is an excellent measure for assessing such reliability. If we can build a model (classifier) for how neurons can distinguish between different conditions using only part of the data and show that the same model works for distinguishing between these same conditions in a new set of data, then this gives us a significant degree of confidence that the current neural activity can reliably distinguish between these conditions, and that our model is capturing the reliability in the data. Additionally, we can compare different models, and if one model is able to extract from the neural data a more reliable signal than another model, this can give us insight into how information is coded in the data. Finally, by building a model to distinguish between one set of conditions and then finding that the same model can *generalize* to a different but related set of conditions, we can infer that the brain contains information in a way that is *invariant* to the exact conditions that were used to build the model. Since all information entering the brain is already present in the sensory nerves and early processing areas, assessing how the brain selectively loses information in order to create behaviorally relevant invariant representations is important for understanding the functional role of higher-level brain regions.

To put things in the terminology used by the machine learning and computational neuroscience communities, we call the processes of building a model on a subset of data “training the classifier” or “learning,” and we call the process of assessing if the model (classifier) still works on a new set of data “testing the classifier.” The “decoding accuracy” (also referred to as “classification accuracy” or “readout accuracy”) is a measure of how well the classifier performs on the new “test set” of data used to test the classifier’s performance. As we mentioned, a high degree of decoding accuracy indicates that the model is capturing reliable differences between different conditions.

The following chapter is a nuts-and-bolts description of how to implement a cross-validation classification scheme that we have found works well for the analysis of multivariate neural data. The methods have been developed by analyzing neural data and assessing what empirically works the best. While we have had experience analyzing several different datasets, there is still much more work to be done to

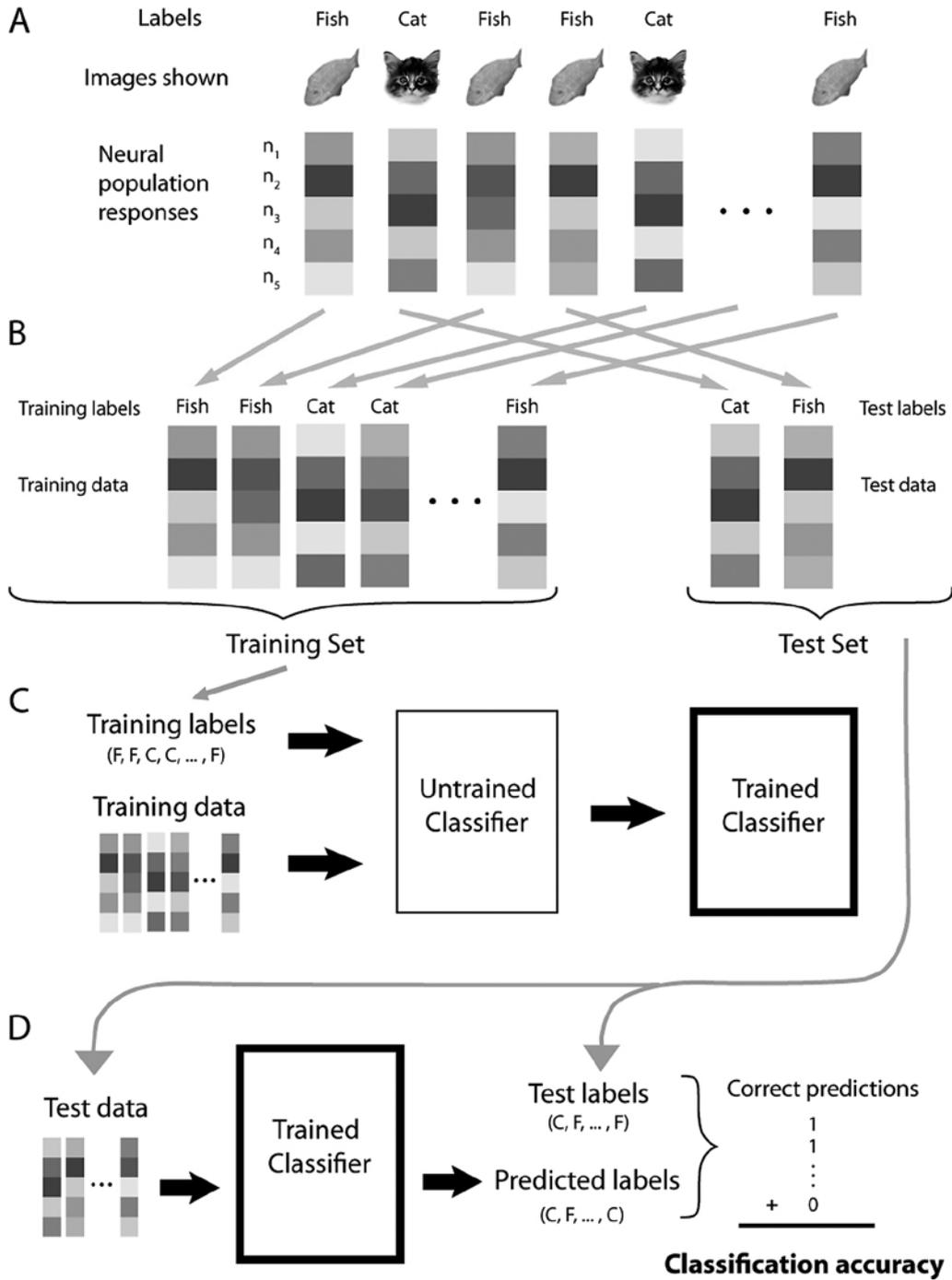
fully characterize the best methods to use. Thus the chapter constitutes work in progress explaining the best methods we have found so far.

Experimental Design

Our discussion centers on a hypothetical experiment where a subject (human or animal) is presented with different images while the investigators record the activity of multiple neurons from implanted electrodes (see, e.g., figure 19.1a). The images belong to different “conditions.” These conditions could refer to different object identities (see, for example, chapters 7 and 10), different object categories, different object positions or viewpoints, the same objects under different experimental manipulations (e.g., attention/no attention) (see, for example, chapter 3), and so on. In order for population decoding methods to work properly, it is important that the experimental design follows a few basic guidelines. First, multiple trials of each condition type must be presented to the subject. For example, if the investigator is interested in decoding which particular stimuli were shown to the subject, then each stimulus must be presented multiple times. While in general more data is always better, there are often experimental restrictions (e.g., it may be difficult to hold a stable recording for prolonged periods of time). We have found that in certain cases as few as five repetitions of each experimental condition are enough to give interpretable results (Meyers et al., 2008), although higher decoding accuracies are usually obtained with more repetitions.

Second, it is important that the stimuli are presented in random order. If the stimuli are not presented in random order (e.g., if all trials of condition 1 are presented before all trials of condition 2, etc.), then even if there is no reliable information about the stimuli in the data, above-chance decoding accuracies could still be spuriously obtained due to nonstationarities in the recording or experimental procedure (e.g., due to electrode drift, varying attentional engagement in the task, adaptation, etc.; see “Testing the Integrity of Decoding” for more details).

Finally, we note that it is not strictly necessary that the recordings from the population be made simultaneously (compare chapter 3 to chapters 7 and 10; see also chapters 2 and 21). If the same experiment is repeated multiple times with single neurons being recorded each time, a “pseudo-population” of responses can be constructed from piecing together the same trial type from multiple sessions (see the section “Formatting Neural Data” for information on how to create pseudo-lations). Due to the experimental challenges in simultaneously recording from multiple electrodes, this approach is common in the neurophysiology community (e.g., see Hung et al., 2005; chapter 10; and chapter 7). The pseudo-population approach, by construction, assumes that the activity of the different neurons is independent given the stimulus, that is, time-varying correlations among neurons are ignored. While



results from such pseudo-populations could potentially distort the estimate of amount of information decoding from the population (Averbeck, Latham, and Pouget, 2006; Averbeck and Lee, 2006), we have seen that much insight can still be gained from this type of analysis (for example, see Hung et al., 2005; Meyers et al., 2008). This question is further discussed in chapters 2, 3, and 21. Additionally, using pseudo-populations allows for population decoding to be applied to many experiments where it is currently not easy to record from populations of neurons (such as from deep brain structures like ventral macaque IT), and it allows for a population decoding reanalysis of older experiments in which simultaneous recordings were not made but for which the same experiment was run for each neuron that was recorded (e.g., see Meyers et al., 2008).

Formatting Neural Data

Analyzing Neural Spiking Data

The first step in applying population decoding to neural-spiking data is to make single unit (SU) or multi-unit (MU) extracellular recordings. In some cases, investigators record multi-unit activity and they are interested in considering the single units that constitute those MUs. There are several spike-sorting algorithms for this purpose (e.g., (Fee, Mitra, and Kleinfeld, 1996; Lewicki, 1998; Wehr, Pezaris, and Sahani, 1999; Harris et al., 2000; Quiroga, Nadasdy, and Ben-Shaul, 2004). Here we assume that spike extraction (and spike sorting) have already been performed in the data and we consider a binary sequence of 0s and 1s, with the 1s indicating the occurrence of spikes. The algorithms apply equally to a series of spikes from SU or MU.

It is useful to look at the average firing rate on each trial as a function of trial number for each neuron/site separately. If different conditions have been presented randomly to the animal, then there should not be any obvious temporal trend in firing rate as a function of trial number. However, there are many types of non-stationarities that could lead to trends over time (including changes in the quality

Figure 19.1

Basic steps involved in training and testing a classifier. (A) An illustration of an experiment in which an image of a cat and an image of a fish were shown in random order to a subject while simultaneous recordings were made from five neurons/channels. The grayscale level denotes the activity of each neuron/channel. (B) Data points and the corresponding labels are randomly selected to be in either the training set or in the test set. (C) The training data points and the training labels are passed to an untrained classifier that “learns” which neural activity is useful at predicting which image was shown—thus becoming a “trained” classifier. (D) The test data are passed to the trained classifier, which produces predictions of which labels correspond to each unlabeled test data point. These predicted labels are then compared to the real test labels (i.e., the real labels that were presented when the test data were recorded) and the percent of correct predictions is calculated to give the total classification accuracy.

of the recordings, subject fatigue, attentional changes over time, neuronal adaptation or plasticity over the course of the recordings, and so on). These time-dependent trends could subsequently be confounded with the questions of interest in the absence of good trial randomization. Eliminating neurons that appear to have non-stationary responses can lead to improvements in decoding accuracy (although in practice so far we have found the improvements due to eliminating neurons with trends to be small). An automatic method that we have used to eliminate neurons that have temporal trends is to compute the average variance found in sliding blocks of twenty trials and compare it to the overall variance among all trials. We typically eliminate all neurons for which the variance over all trials is twice as large as the average variance in twenty-trial blocks.

Once neurons with artefactual trends have been removed, the next step we usually take is to bin the data. While decoding algorithms exist that use exact spike times without binning the data (Truccolo et al., 2005), most of the common machine-learning algorithms we use achieve a higher decoding accuracy when using firing rates computed over time intervals of tens to hundreds of milliseconds. The best bin size to use depends on several factors related to the types of questions of interest. For example, the degree of temporal precision in the neural code can be quantitatively evaluated by using small bins that obviously give more precise temporal accuracy, at the potential cost of having more noisy results. Conversely, if the condition that one is trying to decode seems weak, then we have found binning over larger intervals often reduces noise and leads to more robust results (see Meyers et al. 2008; Hung et al. 2005).

Apart from bin size, it is also of interest to consider the type of filter used to bin the data. In our work we typically have used square (boxcar) filters. The advantage of using these filters is that they provide exact boundaries in terms of the latencies of spikes that contribute to the results, and thus which time bin results are independent from other time bins.¹ Other researchers (Nikolic, Haeusler, and Singer, 2007) have used exponential filters with short (20ms) time constants, in order to mimic what is believed to be the synaptic integration time of neurons, thus creating a potentially more biologically realistic model of the information available to downstream neurons.

Pseudo-populations

In many situations it is not currently practical or possible to record simultaneously from many neurons (for example, it is currently difficult to implant multi-electrode arrays in deep brain structures such as macaque inferior temporal cortex). Additionally, one might want to reanalyze older data that were not recorded simultaneously using population decoding, without having to redo the entire experiment using simultaneous recordings. In such cases, applying population

decoding to pseudo-populations of neurons can give some insight into population coding questions.

We define a pseudo-population of neurons as a population of neurons that was not recorded simultaneously but is treated as if it were.² To create pseudo-populations, one concatenates together responses from different neurons that were recorded when the same condition (stimulus) was presented into a population response—although in fact these neurons were recorded from different experimental sessions (see figure 19.2). We usually create these pseudo-population response vectors inside of a cross-validation procedure, and we recalculate them each time we divide the cross-validation data into blocks (see the section on cross-validation for more details). It should be noted that when creating pseudo-populations, all “noise correlations” within the data are destroyed, and the overall estimate of the amount of information in a population could be over or under estimated (Averbeck, Latham, and Pouget, 2006; Averbeck and Lee, 2006). However, at the moment it remains unclear whether such noise correlations are important for information transmission³ (and there is evidence that in many cases they do not matter, e.g., Panzeri, Pola, and Petersen, 2003; Averbeck and Lee, 2004; Aggelopoulos, Franco, and Rolls, 2005; Anderson, Sanderson, and Sheinberg, 2007). Additionally, at least in principle, we might expect that in many circumstances this bias should affect all conditions equally, which would leave most conclusions drawn from experiments on pseudo-populations unchanged. Still, until more evidence is accumulated about the influence of noise-correlations, it is important to keep in mind that it is possible that population decoding results based on pseudo-populations could differ from results obtained using simultaneously recorded neurons.

Selecting a Classifier

A classifier is an algorithm that “learns” a model from a training set of data (one that consists of a list of population neural responses and the experimental conditions that elicited these responses) and then makes predictions as to what experimental conditions a new “test” dataset was recorded under based on the model that was learned (see chapter 18 for more formal definitions).⁴ For example, the training data could represent neural population responses to different images and a list of the image names (labels) that elicited these population responses, and the test data could consist of population responses to a different set of trials in which the same images were shown. The classifier would then have to predict the names (labels) of the images for each of the neural population responses in the test set. The model that was learned in this example could be a list of which neurons had high firing rates to particular images, and the classifier would make its predictions by combining

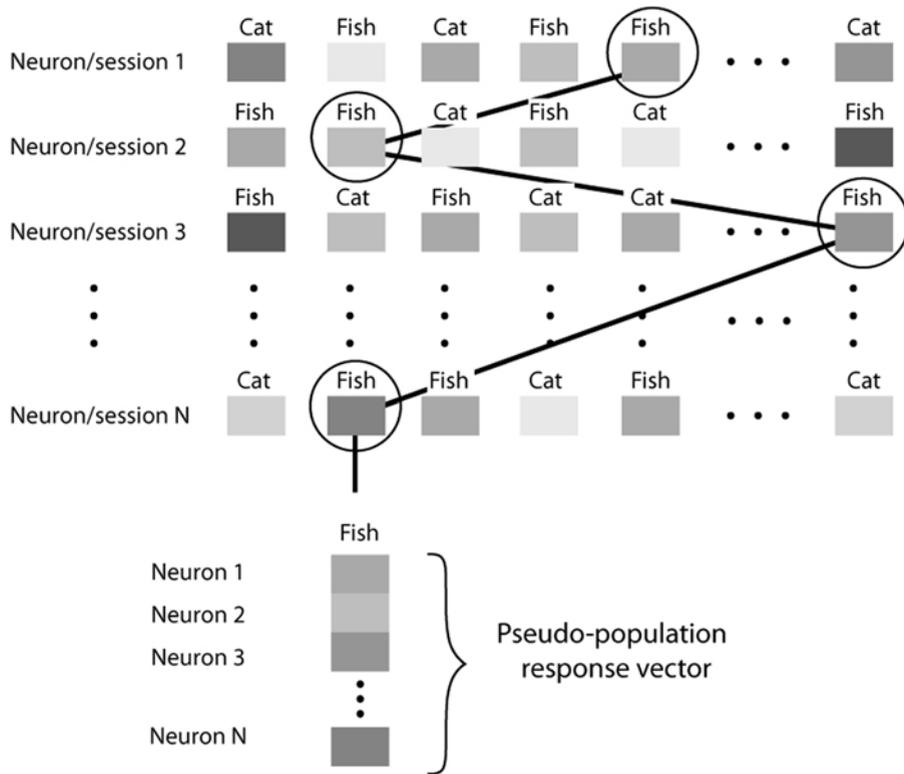


Figure 19.2

Creating pseudo-populations from data that were not recorded simultaneously. Figure 19.1 illustrates a set of experiments in which in each session the responses from a single neuron were recorded (the responses of each neuron is in each row with the darkness indicating the firing rate on particular trials). After recordings have been made from N neurons, a pseudo-population vector can be created by randomly choosing neuron responses from trials in which the same image was shown (circled neuron responses) and concatenating them together to create vector. This pseudo-population vector will be treated as if all the responses had been recorded simultaneously in subsequent population decoding analyses.

the information in this list with the actual firing rates observed in the test set (see figure 19.1).

Many different classifiers exist (see chapter 18 for more details on different classifiers), although we have found that unlike in the analysis of fMRI data, where using regularized classifiers greatly improves decoding performance, decoding results based on neural spiking data seem to be less sensitive to which exact classifier one uses (see figure 19.1). Empirically, we have found that we almost always achieve approximately the same level of performance using linear and nonlinear support vector machines (SVMs), linear and nonlinear regularized least squares

(RLS), Poisson Naïve Bayes classifiers (PNB), Gaussian Naïve Bayes classifiers (GNB), and a simple classifier based on taking the maximum correlation between the mean of training points for each class (MCC) (see figure 19.3, plate 13). The only classifier that consistently yielded worse results was the Nearest Neighbor classifier (NN). Since the MCC classifier has the fastest run time and is the simplest to implement and understand, we recommend using this classifier when initially running experiments. However, since we do not have a deep theoretical reason why all these classifiers seem to be working equally well,⁵ we also recommend testing a few different classifiers, since it is possible that better performance could be achieved on certain datasets, particularly if there are many training examples available.

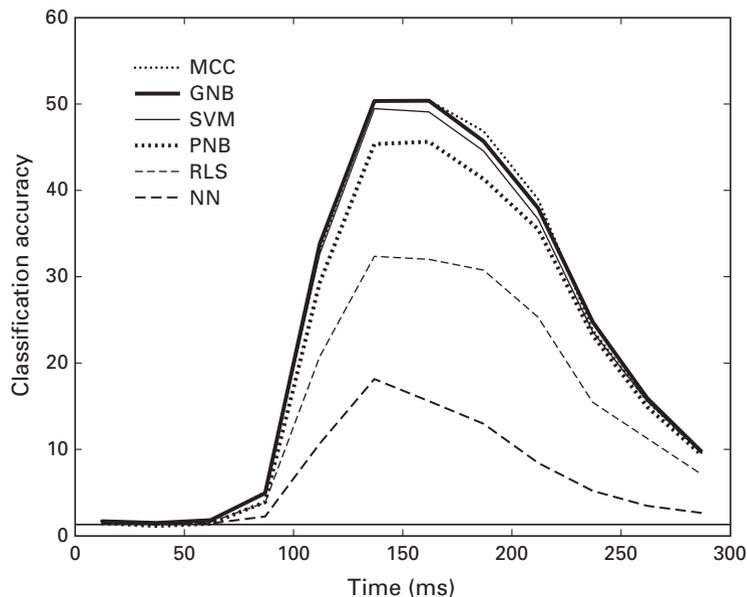


Figure 19.3 (plate 13)

A comparison of different classifiers. The classifiers used are: a maximum correlation coefficient classifier (MCC), a Gaussian Naïve Bayes classifier (GNB), a linear support vector machines (SVM), a Poisson Naïve Bayes classifiers (PNB), a linear regularized least squares (RLS), and a Nearest Neighbor classifier (NN). While the best results here were achieved with the MCC, GNB, and SVM, the overall ordinal increases and decreases in decoding accuracy is similar across classifiers—thus similar conclusions would be drawn regardless of which classifier was used (although the power to distinguish between subtle differences in conditions is enhanced when better classifiers are used). The results in this figure are based on decoding which of seventy-seven objects was shown to a macaque monkey using mean firing rates in 25 ms successive bins (see Hung et al., 2005, and chapter 10 for more details on the experiment).

Cross-validation

Cross-validation is the process of selecting a subset of data to train a classifier on and then using a different subset of data to test the classifier, and it forms one of the most significant components of offline neural population decoding schemes. Typically, cross-validation involves splitting the data into k parts, with each part consisting of j data points. Training of the classifier is done on $k - 1$ sections of the data, and testing is done on the remaining section. The process is usually repeated k times, each time leaving out a different subset of data and testing on the remaining pieces. Classification accuracy is typically reported as the average percent correct over all k splits of the data.

When implementing a cross-validation scheme, it is critically important that *there are no overlapping data between the training set and the test set*, and that *the condition labels that belong to the test set are only used to verify the decoding performance*, and that *they are not used at any other point in the data processing stream*. Any violation of these conditions can lead to spurious results. Thus we recommend doing several sanity checks to insure that the cross-validation scheme has been implemented correctly (see “Testing the Integrity of the Decoding Procedure and Significance of the Results” for more details).

When applying a cross-validation scheme to neural data, we typically use the following procedure. First, if the experimental data from different (stimulus) conditions have been repeated different numbers of times, we first calculate the number of repetitions present for the condition that has the fewest number of repeated trials.⁶ For the purpose of this discussion, let q be a number that is equal to or less than the number of trial repetitions for the condition that has minimum number of repetitions, and let k be a number that divides q evenly (i.e., $q = k * j$, where q , k , and j are all integers). We then randomly select (pseudo-) population responses for q trials for each condition, and put these q repetitions into k nonoverlapping groups, with each group having j population responses to each of the conditions (if pseudo-populations are being used, then it is at this step that these pseudo-populations are created; see figure 19.4a). Next we do cross-validation using a “leave-one-group-out” paradigm, which involves training on $k - 1$ groups and testing on the last group (see figure 19.4B). We then repeat this procedure k times leaving out a different group each time. Finally, we repeat the whole procedure (usually around 50 times) each time selecting a different random set of q trials for each condition, and putting these conditions together in a different random set of k groups. This final step of repeating the whole procedure multiple times and averaging the results gives a smoother estimate of the classification accuracy and is similar to bootstrap smoothing described by Efron and Tibshirani (1997). See algorithm 19.1 for an outline of the complete decoding procedure.

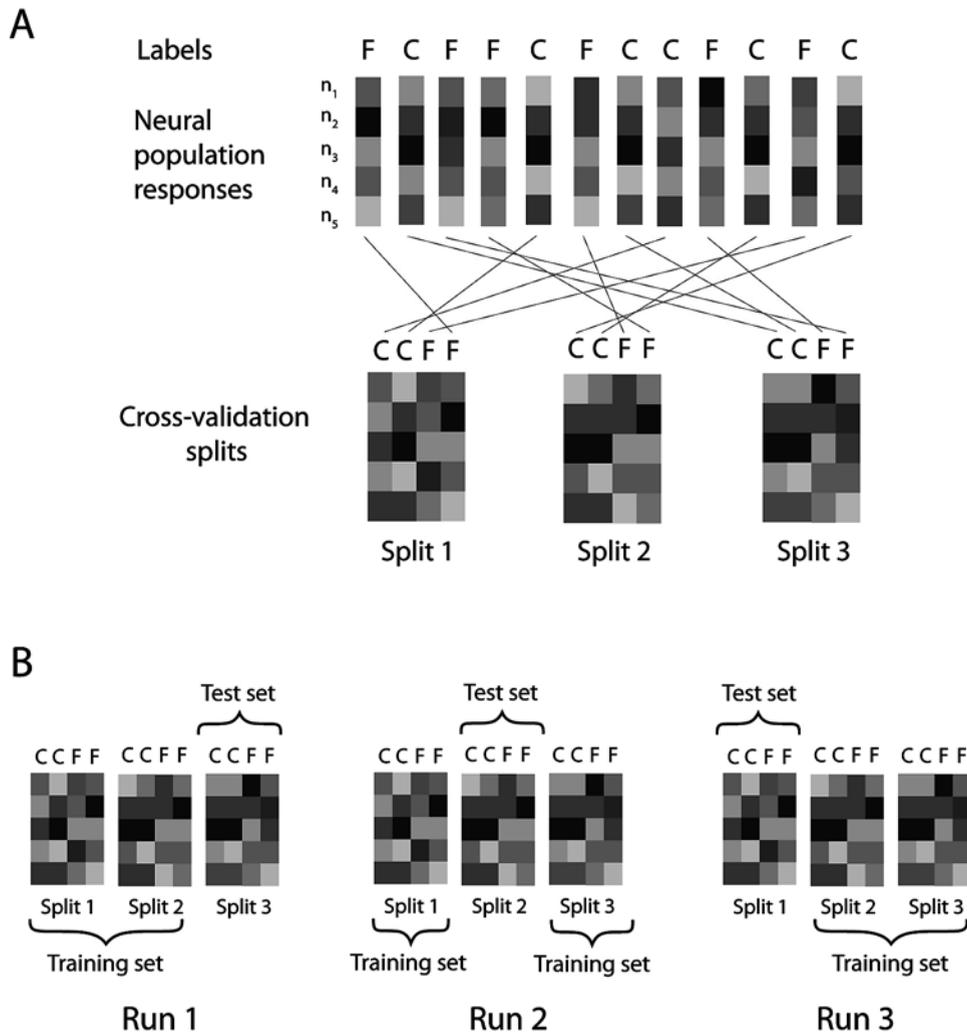


Figure 19.4
 An example of cross-validation. (A) An experiment in which images of a fish or a cat are each shown six times in random order ($q = 6$). Three cross-validation splits of the data ($k = 3$) are created by randomly choosing data (without replacement) from two cat trials and two fish trials ($h = 2$) for each cross-validation split. (B) A classifier is then trained on data from two of the splits and then testing on data from the third remaining split. This procedure is repeated three times ($k = 3$), leaving out a different test split each time.

Algorithm 19.1

The bootstrap cross-validation decoding procedure.

For 50 to 100 “bootstrap-like” trials

Create a new set of cross-validation splits (if the data were not recorded simultaneously, pseudo-population responses are created here).

For each cross-validation split i

1. (Optional) Estimate feature normalization and/or feature selection parameters using only the training data. Apply these normalization and selection parameters to both the training and test data.

2. Train the classifier using all data that is not in split i . Test the classifier using the data on split i . Record classification accuracy.

end
end

Final decoding accuracy is the decoding accuracy averaged over all bootstrap and cross-validation runs.

Feature Selection and Data Normalization

Because different neurons often have very different ranges of firing rates, normalizing the data so that each neuron has a similar range of firing rates is often beneficial in order to ensure that all neurons are contributing to the decoding (and the performance is not dominated by those neurons with the highest overall firing rates). Also, for some decoding analyses examining questions related to neural coding, it is useful to apply feature selection methods in which only a subset of the neurons are used for training and testing the classifier (e.g., see Meyers et al., 2008). When applying either data normalization or feature selection, it is critically important to apply these methods separately to the training and test data, since applying any information from test set to the training set can create spurious results (see “Testing the Integrity of the Decoding Procedure and Significance of the Results” for more information). Thus, since splitting the training and test data occur within the cross-validation procedure, the normalization and feature-selection process must occur within the cross-validation procedure as well.

An example of how data normalization can be applied using a z-score normalization procedure (in which each neuron’s mean firing rate is set to 0, and the standard deviation is set to one) is as follows. Within each cross-validation repetition, take the $k - 1$ groups used for training and calculate each neuron’s mean firing rate and standard deviation across all the training trials, regardless of which conditions were shown. Then normalize the training data by subtracting these *training set* means from each neuron, and dividing by these *training set* standard deviations. Finally,

normalize the test set data by subtracting the *training set* mean and dividing by the *training set* standard deviations for each neuron. In practice we have found that applying z-score normalization to each neuron usually marginally improves decoding accuracies, although overall we have found the results with and without such normalization to be qualitatively very similar.

A similar method can be applied when performing feature selection. In feature selection, a smaller number of neurons/features (s) that are highly selective are chosen from the larger population of all neurons. These s neurons are found *using only data from the training set*. Once a smaller subset of neurons/features has been selected, a classifier is trained and tested using only data from these neurons/features.⁷ For both the data normalization and for the feature selection (and for all data preprocessing in general), the key notion is that the preprocessing is applied separately first to the training set without using the test set, and then it is applied to the test set separately. This insures that the test set is treated like a random sample that was selected after all parameters from training set have been fixed, and thus insures that one is rigorously testing the reliability in the data.

Evaluating Decoding Performance

As mentioned, the output of a classifier is usually a list of predictions for the conditions under which each test data point was recorded. The simplest way to evaluate the classification performance is to compare the predictions that the classifier has made to the actual conditions that the test data were really recorded under, and report the percent of times the classifier's predictions are correct. This method of classification evaluation is usually referred to as a 0–1 loss function, and gives reasonably interpretable results, particularly for easy classification tasks. Another method that exists for evaluating classifier performance is to use a “rank” measure of performance (Mitchell et al., 2004). When using a rank measure of performance, the classifier must return an ordinal list that ranks how likely each test data point is to have come from each of the conditions. The rank measure then assesses how far from the bottom of the list the actual correct condition label is. The rank measure can also be normalized by the number of classes to give a “normalized rank” measure in which a value of 1 corresponds to perfect classification and a value of 0.5 corresponds to chance, which makes the results easy to interpret. This measure also has the advantage of being more sensitive because there is no hard requirement placed on getting the actually condition exactly correct, and thus we find that this method generally works better on more difficult classification tasks.

It is also instructive to create a confusion matrix out of the classification results. If there are c conditions being decoded, a confusion matrix is a $c \times c$ sized matrix in which the columns correspond to the real condition labels of the test set, and the

rows correspond to the number of times a condition labels was predicted by the classifier. The advantage of the confusion matrix is that it allows one to easily evaluate what conditions the classifier is making mistakes on, and thus what conditions elicit neural population responses that are similar. Additionally, one can convert a confusion matrix into a lower bound on the amount of mutual information (MI) between the neural population response and the conditional labels, which gives a way to compare decoding results to information theoretic measures of neural data (Samengo, 2002). Mutual information calculated from the confusion matrix can potentially be more informative than just looking at 0–1 loss results since MI takes into account the pattern of classification errors that was made (Quiñ Quiroga and Panzeri, 2009). Converting a confusion matrix into a mutual information measure can be done by normalizing the confusion matrix to sum to one, and then treating the normalized matrix as a joint probability distribution between actual and predicted conditions. Applying the standard formula for mutual information⁸ to this probability distribution gives a lower bound estimate of mutual information.

Testing the Integrity of the Decoding Procedure and Significance of the Results

Once the decoding procedure has been run, it is useful to do a few tests to ensure that any decoding accuracies that are above the expected chance level of performance are not due to artifacts in the decoding procedure. One simple test is to apply the decoding procedure to data recorded in a baseline period that occurred prior to the presentation of the condition/stimulus that has been decoded. If the decoding results are above the expected chance level during this baseline period then there is a confounding factor in the decoding procedure or in the experimental design. From our past data analyses, we have found that above-chance decoding results during baseline period are often due to changes in the average firing rate of neurons over the course of a trial combined with an experimental design or decoding procedure that is not fully randomized.

Apart from examining baseline periods, there are a few other tests that can easily be applied to check the integrity of the decoding procedure. Randomly permuting the condition labels (or randomly shuffling the data itself) are other simple tests that should result in chance levels of accuracies at all time points since the relationship between the data and the condition labels is destroyed.

Randomly permuting which labels correspond to which data points also gives a way to assess when decoding accuracies are above chance. To perform this test, a null distribution is defined by the expected readout decoding accuracies if there was no relationship between the neural data and the condition labels. This null distribution can be created by permuting the relationship between the condition labels and the data, running the full decoding procedure on this label permuted data to obtain

decoding results, and then repeating this permuting and decoding process multiple times. P -values can then be estimated from this null distribution by assessing how many of the values in the null distribution are less than the value obtained from decoding based on using the real labels. For example, upon performing 1,000 permutations, it is possible to test if decoding accuracy with the real labels is above chance by comparing against the actual decoding accuracy with the distribution in the 1,000 permutations. For an probability level of 0.01, less than 10 of the 1,000 decoding accuracies in the null distribution should be greater than the decoding accuracy found using the real condition label-data correspondence.

It has also been suggested that the significance of decoding results can be obtained by comparing the number of correct responses produced by a classifier to the number of correct responses one would expect by chance using a binomial distribution (Quiñ Quiroga and Panzeri, 2009). The method works by creating the binomial distribution

$$P(k) = \binom{n}{k} p^k (1-p)^{n-k},$$

with n being the number of test points, and p being the proportion correct one would expect by chance (e.g., $1/(\text{number of classes})$). A p -value can then be estimated as

$$p\text{-value} = \sum_{k=j} P(k),$$

where j is the number of correct predictions produced by the classifier. This procedure has the advantage of being much more computationally efficient than the permutation method described before. However, there are several pitfalls of using this method that one must be aware of. In particular, for this method to be used correctly, one should estimate the p -value for each cross-validation split separately, since using the total number of correct responses over all cross-validation splits (and/or over all “bootstrap” repetitions) violates the assumption of data point independence that this test relies on (and hence can lead to spuriously low p -values and type 1 errors). However, estimating this p -value separately for each cross-validation split greatly reduces the sensitivity of the test leading to spuriously high p -values and potential type 2 errors. Thus unless one has a large amount of test data, it is tough to get insightful results using this method.

More Advanced Topics

The preceding sections have focused on how to run simple decoding experiments in which we are primarily interested in decoding the exact conditions/stimuli that were present when an experiment was run. However, perhaps the greatest

advantage of using population decoding is that it can give insight into more complex questions about how information is coded in the brain. In the last two sections we discuss how to use neural decoding to assess how information is coded in the activity of neurons and how to assess if the information is represented in an abstract/invariant way (which is a particularly meaningful question when decoding data recorded from the highest levels of visual cortex and prefrontal cortex).

Examining Neural Coding

Despite a significant amount of research, many questions about how information is coded in the activity of neurons still have not been answered in an unambiguous way. These questions include (among others): (1) are precise spike times important for neural coding or are firing rates computed over longer time intervals all that matters to represent information, (2) is more information present in the synchronous activity of neurons, and (3) is information at any point in time widely distributed across most of the population of neurons, or is there a compact subset of neurons that contains all or most of the information.⁹ While population decoding can not completely resolve the debate surrounding these issues, it can give some important insights into these questions. We will describe how one can use population coding to address these issues, as well as some caveats one must keep in mind when interpreting the results from such analyses.

In order to address the question of how temporally precise the neural code is, it is of interest to perform population decoding using different binning schemes and to quantify how much information is lost for different representations. This can be done simply by using different bin sizes for decoding and describing which bin size gives rise to highest decoding accuracies (Meyers et al., 2009). More complex schemes can be used in which an instantaneous rate function is estimated using precise spike timing (Truccolo et al., 2005), and then this representation is used for decoding (see also the discussion in chapter 2). When doing such analyses, a few important caveats should be kept in mind, such as the fact that the temporal precision of the recordings and a limited sampling of data could potentially influence the results.

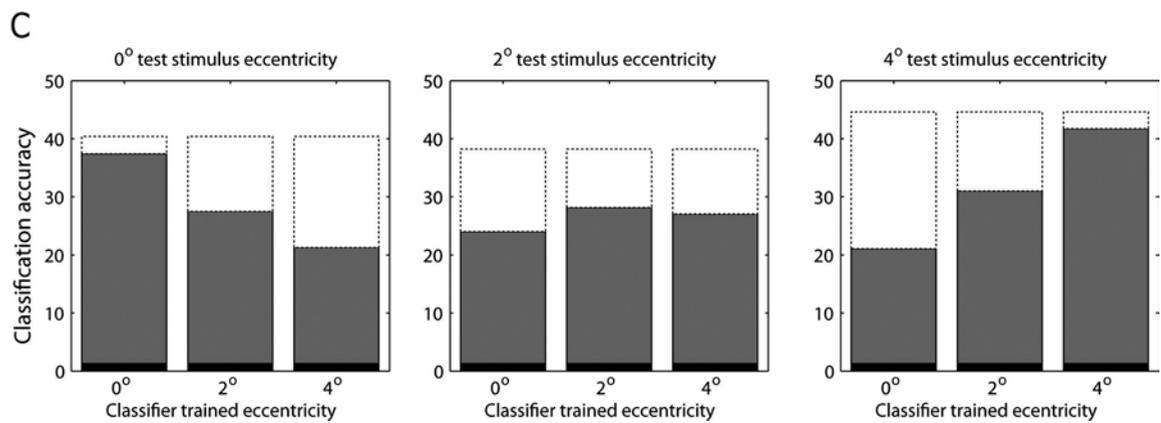
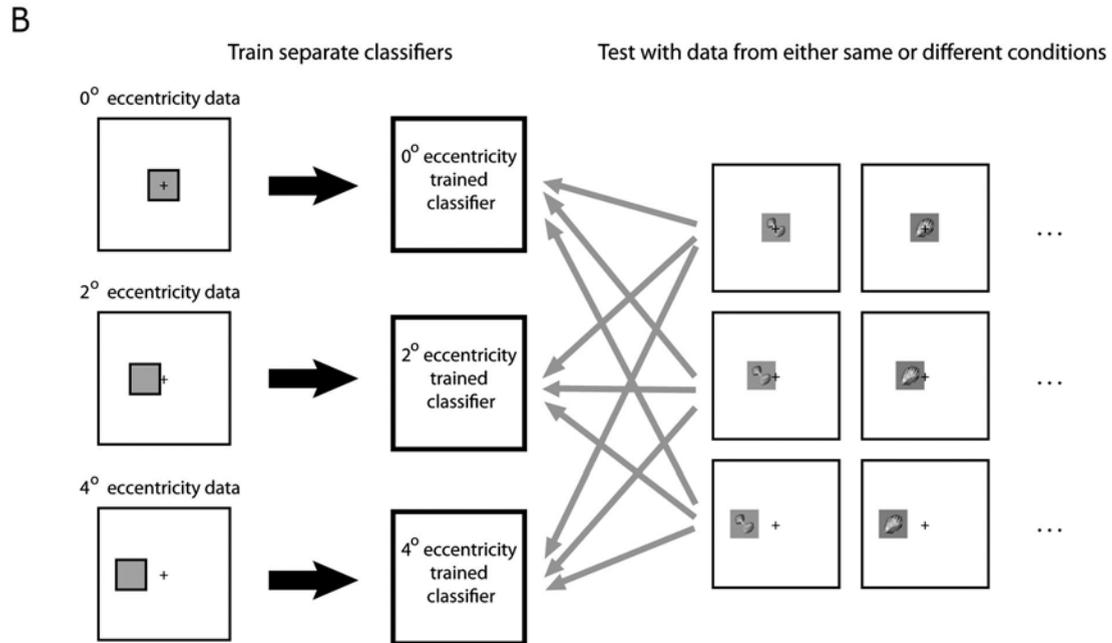
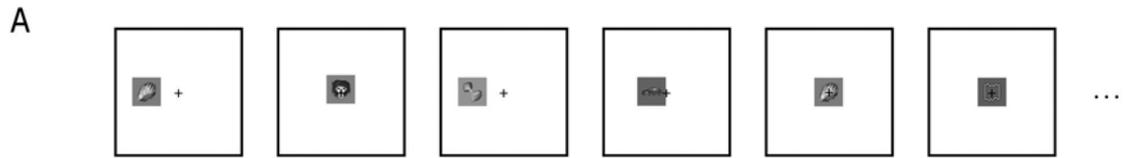
To examine whether synchronous activity is important, or alternatively, if neurons act independently given the particular trial conditions, one can decode the activity of a population of neurons that was recorded simultaneously and compare the results to training a classifier using pseudo-populations created from the same dataset (Latham and Nirenberg, 2005; see also chapters 2 and 21). Since pseudo-populations keep the stimulus-induced aspect of the neural population code intact but destroy the correlations between neurons that occurred on any given trial (noise correlations), this gives a measure of how much extra information is present when

the exact synchronous pattern of activity on a single trial basis is preserved (see also chapters 2 and 21). Of course one must use a sufficiently powerful classifier that can exploit correlations in the data. Also, one must be careful when interpreting the results since increases or decreases in the firing rates of all neurons could potentially occur due to artifacts in the recording procedure. Still, population decoding can begin to give an idea of how much potential additional information could be contained in synchronous activity. Additional methodological challenges when addressing these questions include the difficulties in finding neuronal combinations that could be synchronized given the large number of neurons in cortex, the potential dependencies of synchrony with distance, the potential dependencies on neuronal subtypes, and others. These issues are not specific to the population decoding approach described here, but they also affect other methods used to examine correlations between neurons.

Finally, feature selection can be used to examine whether information is widely distributed across most neurons or whether, at any point in time, there is a compact subset of neurons that contains all or most of the information that the larger population has. As described in the section on cross-validation, feature selection can be used to find the most selective neurons on the training set and then use only these neurons when both training and testing the classifier. If using a reduced subset of neurons leads to decoding accuracy that is just as good as that seen in the larger population, then this indicates that indeed most of the information is contained in a small compact subset of neurons. One important caveat in this analysis is that if only one time period is examined, it is possible that some of the neurons might be nonselective due to problems with the recordings. However, if one can show that different small compact set of neurons contain the information at different points in time in the experiment (as shown in Meyers et al., 2008), this rules out problems with the recording electrode as an explanation.

Evaluating Invariant/Abstract Representations

Since all information that is available to an organism about the world is present in early sensory organs (such as the retina for vision and the cochlea for audition), one of the more important questions about studying the cascade of processes involved in recognition is how information is lost in an intelligent way along the processing steps in cortex in order to create more useful invariant/abstract representations. For example, many neurons in IT are highly invariant to the position of visual objects, i.e., they respond similarly to particular objects regardless of the exact retinal position that an image is shown (within certain limits; see Li et al., 2009; Logothetis and Sheinberg 1996). Such an invariant representation is obviously not present in lower level areas that are retinotopic, and having this type



of intelligent information loss could be behaviorally useful when an animal needs to detect the presence of an object regardless of the object's exact location on the retina.

Testing whether information is contained in an invariant/abstract way can readily be approached using neural decoding. To do such a test, one can simply train a classifier on data that were recorded in one condition and then test the classifier on a different related condition. If the classifier can still perform well on the related condition, then this indicates that the information is represented in an invariant or abstract way. Taking the example of position invariance again, one can train a classifier with data recorded at one retinal location and then test with data recorded at a different location, as is done in figure 19.2 (see also Hung et al., 2005). As can be seen in figure 19.5, the representation in IT is highly invariant/tolerant to changes in the exact retinal position. A similar type of analyses can also be done to test if different brain regions contain information in an “abstract” format. For example, Meyers et al. (2008) used data from a task in which monkeys needed to indicate whether an image of a cat or dog was shown regardless of which exact image of a dog or cat was shown. By training a classifier with data that was collected from a subset of images of dogs and cats and then testing the classifier when a different set of images of dogs and cats were shown, Meyers and colleagues (2008) could see that indeed there seemed to be information about the more abstract behaviorally relevant categories apart from the information that was due to the exact visual images of particular dogs and cats. An analogous method of training a classifier with data from one time period and testing with data from a different time period was also used to show that the neural code of an image does not appear to be stationary, but instead seems to change systematically over the course of a trial—which illustrates again how training and testing with different but related data are effective ways to answer a range of different questions.

Figure 19.5

Assessing position invariance in anterior inferior temporal cortex. (A) An illustration of an experiment conducted by Hung et al. (2005) in which images of 77 objects were displayed at three different eccentricities. (B) An illustration of a classifier being trained on data from different eccentricities, yielding three different models. These three models were then tested with data from either the same eccentricity that the classifier was trained on (using data from different trials), or with data at a different eccentricity. (C) Results from this procedure show that the best performance was always achieved when the training and testing was done at the same eccentricity (gray bars), however performance is well above chance (black bars) at all eccentricities, indicating the population of IT shows a high degree of position tolerance. Also, when the classifier is trained using data from all eccentricities (dotted bars), the results are even better than when training and testing is done at the same eccentricity, indicating that the best performance can be achieved when the classifier learns to rely mostly heavily on the neurons that have the most position invariance. Decoding results are based on multi-unit recordings from 70 neurons made by Hung et al. (2005), using the mean firing rate in a 200ms bin that started 100ms after stimulus onset and a MCC classifier.

Conclusions

In this chapter we described how to implement a population decoding procedure, highlighted the analysis methods that we have found work best, and pointed out caveats to be aware of when interpreting results. Neural population decoding holds a great amount of potential as a method to gain deeper insight into how the brain functions, particularly with regard to answering questions related to neural coding and to how invariant and abstract representations are created in different brain regions.

Acknowledgments

We would like to thank Jim DiCarlo and Chou Hung for supplying the data that were used in this chapter. We would also like to thank Tomaso Poggio for his continual guidance. This work was supported by the American Society for Engineering Education's National Science Graduate Research Fellowship (EM) and by NSF (GK) and NIH (GK).

Notes

1. We found this to be particularly useful when exploring how the neural code changes with time (i.e., Meyers et al., 2008), since it was important to know which time periods were independent from each other (see Meyers et al., 2008 for more details).
2. Different researchers have used pseudo-populations to analyze their data, including Georgopoulos et al., 1983; Gochin et al., 1994; Rolls, Treves, and Tovee, 1997; Hung et al., 2005; and Meyers et al., 2008. These populations are often referred to by different names, including “pseudoresponse vectors” (Gochin et al., 1994) and “pseudosimultaneous population response vectors” (Rolls, Treves, and Tovee, 1997). Additionally, the process of recording over separate sessions to create pseudo-populations has been referred to as the “sequential method,” and the process of recording many neurons at once for the purposes of population decoding has been called the “simultaneous method” (Tanila and Shaprio, 1998).
3. If noise correlations do not matter (i.e., if the activity of each neuron is statistically independent of the activity of other neurons given the current stimulus or behavioral event being represented), then a brain region is said to use a “population code” (see also chapters 2 and 21). If interactions between neurons do code additional information, then a brain region is said to use an “ensemble code” (Hatsopoulos et al., 1998). Whether population codes or ensemble codes are used by the brain still remains an open question in neuroscience.
4. To be slightly more formal, a training set consists of a pair of values (X, y) , where X is an ordered set of neural population response vectors and y is an ordered set of labels indicating the conditions that the neural responses were recorded under (with X_i being the neural population response to the i th training trial, and y_i indicating which conditions/stimulus was shown on that trial). “Learning” consists of applying a function $f(X, y) \rightarrow M$ that takes the training neural data and the training labels and returns a set of model parameters M . This model can then be used by another “inference” function, $g(\hat{X}, M) \rightarrow \hat{y}$, that takes a new set of test data \hat{X} and produces a prediction \hat{y} indicating which labels/conditions correspond to each test point \hat{X}_i . The predicted \hat{y} can be compared to the real test labels \hat{y} to evaluate decoding accuracy. Typically, the function g is called the “classifier,” although the learning algorithm f could also be considered part of the classifier as well. Also, it is common to write the learning function f as returning the inference algorithm g (that is, $f(X, y) \rightarrow g$). Chapter 18 discusses classifiers in further detail.

5. It should be noted that in general on most classification tasks (such as on fMRI data, and on computer vision features), more complex classifiers such as SVMs and RLS tend to work better than simple ones such as the MCC. The fact that such simple classifiers work well on neurophysiological recordings suggests that there is something particular about neural spiking data that is well fit by these simple models.
6. In most properly designed decoding experiments, different conditions are presented in a random order, and since the ability to record from a neuron often ends at a random point in time within an experimental session, it is fairly common to have a different number of stimulus presentations for different conditions (particularly when doing decoding on pseudo-populations). Since having different numbers of training examples for different conditions can bias certain types of classifiers into choosing the condition with the most training examples, we make sure that there are an equal number of training examples in each condition. Of course if there is reason to believe that there would be more examples of one condition in the world than another condition, then it could be reasonable to have this bias in the classifier (i.e., this bias could be a reasonable approximation for the prior distribution of the conditions/stimuli in the world). Chance performance in this unbalanced training case then becomes the proportion of training points in the class with the most training points (i.e., the chance level is the expected proportion of correct responses if the classifier always selected the class with the highest prior probability).
7. For example, if one is trying to decode what exact images were shown to a monkey based on firing rates of individual neurons, one could use a simple feature selective method by applying a one-way ANOVA to firing rates *in the training set* (with the ANOVA groups consisting of the firing rates to particular images), and then training and testing the classifier using only neurons that had highly selective p -values.
8. The standard formulation is $I = \sum_s \sum_{s'} P(s', s) \log_2 [P(s', s) / (P(s')P(s))]$, where s' are the predicted labels on the test set, s are the real labels on the test set, and $P(s', s)$ is the joint probability distribution obtained from normalizing the confusion matrix, and the marginal distributions $P(s)$ and $P(s')$ can be derived from the joint distribution with the formulas $P(s') = \sum_s P(s', s)$ and $P(s) = \sum_{s'} P(s', s)$.
9. We use “compact” subset here rather than “sparse” subset, since sparse activity usually refers to a situation when only a few neurons are *active* at the same time, while here many neurons could be active, but only a small subset of them might *contain information* about the condition that is being decoded.

References

- Aggelopoulos N, Franco L, Rolls E. 2005. Object perception in natural scenes: Encoding by inferior temporal cortex simultaneously recorded neurons. *J Neurophysiol* 93: 1342–1357.
- Anderson B, Sanderson M, Sheinberg D. 2007. Joint decoding of visual stimuli by IT neurons' spike counts is not improved by simultaneous recording. *Exp Brain Res* 176: 1–11.
- Averbeck B, Latham P, Pouget A. 2006. Neural correlations, population coding and computation. *Nat Rev Neurosci* 7: 358–366.
- Averbeck B, Lee D. 2004. Coding and transmission of information by neural ensembles. *Trends Neurosci* 27: 225–230.
- Averbeck B, Lee D. 2006. Effects of noise correlations on information encoding and decoding. *J Neurophysiol* 95: 3633–3644.
- Efron B, Tibshirani R. 1997. Improvements on cross-validation: the .632+ bootstrap method. *J Am Stat Assoc* 92: 560, 548.
- Fee MS, Mitra PP, Kleinfeld D. 1996. Automatic sorting of multiple unit neuronal signals in the presence of anisotropic and non-Gaussian variability. *J Neurosci Methods* 69: 175–188.
- Georgopoulos AP, Caminiti R, Kalaska JF, Massey JT. 1983. Spatial coding of movement: a hypothesis concerning coding of movement direction by motor cortical populations. *Exp Brain Res Suppl* 7: 327–336.

- Gochin P, Colombo M, Dorfman G, Gerstein G, Gross C. 1994. Neural ensemble coding in inferior temporal cortex. *J Neurophysiol* 71: 2325–2337.
- Harris KD, Henze DA, Csicsvari J, Hirase H, Buzsáki G. 2000. Accuracy of tetrode spike separation as determined by simultaneous intracellular and extracellular measurements. *J Neurophysiol* 84: 401–414.
- Hatsopoulos NG, Ojakangas CL, Maynard EM, Donoghue JP. 1998. Detection and identification of ensemble codes in motor cortex. In *Neuronal ensembles: Strategies for recording and decoding*, ed. H. Eichenbaum and J. Davis, pp. 161–175. New York: Wiley.
- Hung C, Kreiman G, Poggio T, DiCarlo J. 2005. Fast readout of object identity from macaque inferior temporal cortex. *Science* 310: 863–866.
- Latham PE, Nirenberg S. 2005. Synergy, redundancy, and independence in population codes, revisited. *J Neurosci* 25: 5195–5206.
- Lewicki MS. 1998. A review of methods for spike sorting: the detection and classification of neural action potentials. *Network* 9: R53–R78.
- Li N, Cox DD, Zoccolan D, DiCarlo JJ. 2009. What response properties do individual neurons need to underlie position and clutter “invariant” object recognition? *J Neurophysiol* 102: 360–376.
- Meyers E, Freedman D, Kreiman G, Miller M, Poggio T. 2009. Decoding dynamic patterns of neural activity using a “biologically plausible” fixed set of weights. In *Frontiers in systems neuroscience*. Available at: http://frontiersin.org/conferences/individual_abstract_listing.php?conferid=39&pap=1437&ind_abs=1.
- Meyers EM, Freedman DJ, Kreiman G, Miller EK, Poggio T. 2008. Dynamic population coding of category information in inferior temporal and prefrontal cortex. *J Neurophysiol* 100: 1407–1419.
- Mitchell TM, Hutchinson R, Niculescu RS, Pereira F, Wang X, Just M, Newman S. 2004. Learning to decode cognitive states from brain images. *Mach Learn* 57: 145–175.
- Nikolic D, Haeusler S, Singer WW. 2007. Temporal dynamics of information content carried by neurons in the primary visual cortex. In *Advances in neural information processing systems*, ed. B Schölkopf, J Platt, and T Hofmann, pp. 1041–1048. Cambridge, MA: MIT Press.
- Panzeri S, Pola G, Petersen R. 2003. Coding of sensory signals by neuronal populations: The role of correlated activity. *Neuroscientist* 9: 175–180.
- Quian Quiroga R, Panzeri S. 2009. Extracting information from neuronal populations: information theory and decoding approaches. *Nat Rev Neurosci* 10: 173–185.
- Quiroga RQ, Nadasdy Z, Ben-Shaul Y. 2004. Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. *Neural Comput* 16: 1661–1687.
- Rolls ET, Treves A, Tovee MJ. 1997. The representational capacity of the distributed encoding of information provided by populations of neurons in primate temporal visual cortex. *Exp Brain Res* 114: 149–162.
- Samengo I. 2002. Information loss in an optimal maximum likelihood decoding. *Neural Comput* 14: 771–779.
- Logothetis NK, Sheinberg DL. 1996. Visual object recognition. *Annual Review of Neuroscience* 19: 577–621.
- Tanila H, Shaprio M. 1998. Ensemble recordings and the nature of stimulus representation in hippocampal cognitive maps. In *Neuronal ensembles: Strategies for recording and decoding*, ed. H. Eichenbaum and J. Davis, pp. 177–206. New York: Wiley.
- Truccolo W, Eden UT, Fellows MR, Donoghue JP, Brown EN. 2005. A point process framework for relating neural spiking activity to spiking history, neural ensemble, and extrinsic covariate effects. *J Neurophysiol* 93: 1074–1089.
- Wehr M, Pezaris J, Sahani M. 1999. Simultaneous paired intracellular and tetrode recordings for evaluating the performance of spike sorting algorithms. *Neurocomputing* 26–27: 1061–1068.