6
Regression

This chapter introduces multivariate approaches used in studying the electrophysiology of human memory. We begin with the regression method both because it is extremely useful and because it illustrates many of the fundamental issues in multivariate analysis.

Consider a standard problem in human memory. After studying a list of items, we ask subjects to recall as many items as they can either immediately or after a delay. The same subject will sometimes exhibit better memory for the list items than at other times. Here we might want to know what features of neural activity predict variability across lists in the number of items the subject recalled. For example, does a certain signal in a particular brain region consistently increase or decrease in a manner that predicts successful memory encoding and/or retrieval.Assuming that we have a large sample of candidate features, such as the distribution of spectral power at each of a large number of recording electrodes, we could test them one at a time, asking whether each feature reliably predicts the number of words recalled on that particular list. This has long been a standard approach in cognitive neuroscience, and so long as one appropriately adjusts for multiple-comparisons (cf. Chapter 3), one can make inferences about how reliably a given feature predicts mnemonic success. This is the classic univariate approach for linking brain activity to performance on a memory task.

Suppose, however, that you want to ask the question a different way: Given a set of neural features measured during the study phase of a memory task (or the recall phase, or both), how accurately can we predict the proportion of items subjects will recall on a given list. Here we are not asking about the importance of specific features; rather, we are asking how well the ensemble of features can predict recall performance. To answer this question we need to evaluate a single model that includes information from all of the features. One simple and powerful way of doing this is the familiar multiple (linear) regression technique. Here we would try to predict recall performance by computing a weighted sum of neural features whose weighting coefficients are set to minimize some objective function (e.g., the deviation between observed and predicted values).

We are making a conceptual leap from asking which features underlie function, to asking how the distribution of features predict function. By addressing this prediction problem, we can exploit signals that may be missed by univariate methods. Specifically, univariate methods may lead us to believe that only a subset of features (e.g., activity in one particular

brain region) provides information about behavior, whereas there may be a lot of collective information distributed across features even when they individually fail to meet an univariate significance threshold (Haxby et al., 2001).

The Statistical Learning Problem

Machine learning algorithms can be divided into two categories: *supervised learning* and *unsupervised learning*. In supervised learning, the algorithm learns a mapping between a set of inputs (predictors) and a corresponding output (label) using a set of pre-labeled observations (training set). The goal of supervised learning is to predict the value of a future output based on its features. In unsupervised learning, the algorithm is trained on unlabeled data, and it aims to describe patterns and associations among the input variables. The majority of this chapter and the next chapter is devoted to the discussion of two types of supervised learning: regression (continuous output) and classification (discrete output). The last section in the next chapter will discuss one of the most important unsupervised learning algorithms: principal component analysis.

Supervised Learning

Let us consider a learning problem in which we want to predict an output variable Y given a set of input variables $X = (X_1, \cdots, X_p)$. We let $x_i^T = (x_1, \cdots, x_p)$ denote the i^{th} training observation of the input vector X. A set of N input p-vectors x_i is represented by the $N \times p$ matrix $\mathbf{X}$ whose i^{th} row is x_i and a set of N outputs, y_i, is represented by the $N \times 1$ vector $\mathbf{y}$. The task of supervised learning can then be summarized as follows: given an input vector X, make a good prediction of y. Supervised learning is also called "learning with a teacher" since the output or response variable acts as a teacher and the learning algorithm acts as a student. The algorithm (student) presents a prediction $\hat{y}$ based on the input vector and the output variable y corrects the error of the prediction by telling the algorithm how far the prediction is from the actual output, which is also known as the loss of the prediction $L(\hat{y}, y)$. Supervised learning algorithms are trained to minimize the loss function associated with the input matrix $\mathbf{X}$. As a result, different loss functions will yield different learning algorithms.

One of the most common loss functions is the *squared error loss*: $L(Y, f(X)) = (Y - f(X))^2$. Suppose $X \in \mathbf{r}^p$ is a random vector and $Y \in \mathbf{r}$ is a random variable with a joint distribution $P(X, Y)$. The expected prediction error corresponding to the function f is given by

$$EPE(f) = \mathrm{E}_{(X,Y)}(Y - f(X))^2, \tag{6.1}$$

where the expectation is taken over the joint distribution of X and Y. In principle, the expected prediction error cannot be computed because the joint distribution $P(X, y)$ is unknown. Consequently, we typically minimize the *training loss* (also known as *training error* or *empirical risk*), which is the average loss over training observations. It can be shown (Hastie, Tibshirani, & Friedman, 2009) that under the squared error loss, the function f that

minimizes the expected prediction error is

$$f(x) = \mathrm{E}(Y \mid X = x). \tag{6.2}$$

In other words, the best prediction of Y for any value x is just the conditional mean of Y given x. There are several methods to approximate the conditional expectation. One of them is the nearest neighbors method that predicts the label of an observed input x by averaging all the labels of its neighbors. Another popular method is the least squares method, which assumes that f is a linear function of the input variables. We will discuss these two methods next.

Nearest Neighbors

The nearest neighbors method approximates the right-hand side of Equation 6.2 by taking the average of the responses of the neighbors of x. For each value x in the input space, let us define a neighborhood $N_k x$ of x that contains the k closest points to x according to some metric. The prediction of the responses of x is given by

$$\hat{y} = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i. \tag{6.3}$$

That is we find k observations with x_i closest to x and average their responses.[1] This algorithm assumes the k-nearest neighborhood of x is small enough such that instead of conditioning on x, we can just look at points close to x. However, we shall see that this assumption is easily violated with high-dimensional data. The effective number of parameters for k-nearest neighbors is approximately N/k because the algorithm divides the feature space into approximately N/k regions, each of which has one parameter (the neighborhood mean). It appears that the training error is close to 0 when $k = 1$.

[1] Euclidean distance is a standard metric used to identify the k observations nearest to x_i. For two patterns x_1 and x_2 the Euclidean distance is defined as: $\sqrt{\sum_{i=1}^{p} (x_{1i} - x_{2i})^2}$

Curse of Dimensionality

The nearest neighbors method suffers from the "curse of dimensionality", which refers to the poor estimation of y using the nearest neighbors when x is high-dimensional because the closest neighbors of x may indeed be very far away from x in the p-dimensional space as p increases. Moreover, it requires an exponential increase in the number of observations as p increases to maintain a constant sampling density. As a result, the nearest neighbors of x may no longer reflect the local behavior around x. A common technique to reduce the dimensionality of the input vector is principal component analysis (PCA) which we will discuss in the next chapter.

Regression

Consider the problem of evaluating the predictability of recall or recognition performance across lists on the basis of neural signals measured at many brain locations. Consider further the problem of removing the effect of known covariates, such as item difficulty, from this predictive model.

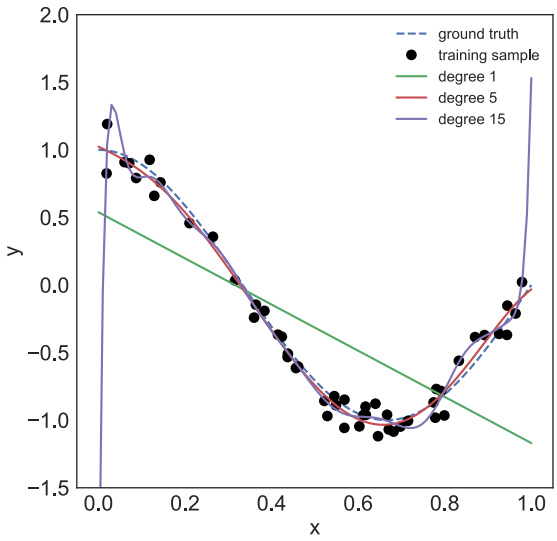

Figure 6.1: An example of polynomial regression with degree equal to $1, 5, 15$. The training data set consists of 30 observations simulated according to $y = \cos(1.5\pi x) + \epsilon$, where $\epsilon \sim \mathrm{N}(0, 0.01)$. The figures shows that a linear function is inadequate and *underfitting* because it fails to fit the training data. A polynomial of degree 5 almost fits the data perfectly. A higher degree polynomial such as one with 15 degrees *overfits* the training data because it learns the noisy component of the data.

We can formalize this problem as a regression problem in which the recall performance is the response variable and neural signals are the predictors.

In this chapter, we introduced a class of regression learning algorithms called linear models, which assume that the conditional mean of the response given the predictors is a linear function of the predictors:

$$\mathrm{E}[Y \mid X = x] = \beta_0 + \sum_{i=1}^{p} \beta_j X_j \tag{6.4}$$

The parameters $\beta = (\beta_0, \cdots, \beta_p)$ are estimated by minimizing the training error loss:

$$\hat{\beta} = \arg\min_{\beta} \frac{1}{N} \sum_{i=1}^{N} (y_i - \mathbf{x}_i' \beta)^2. \tag{6.5}$$

This optimization problem has a closed-form solution $\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$ which allows one to compute the solution easily. In practice, the relationship between the response and the predictors is often non-linear. Therefore, we need more flexible models to capture non-linear effects and this leads to a class of *generalized additive models* which assume an additive effect of transformations of predictors. Concretely,

$$f(x) = \beta_0 + \sum_{j=1}^{p} f_j(X_j), \tag{6.6}$$

where f_j is a smooth function of the predictor X_j. If f_j are polynomials of X_j, we have a class of polynomial regression algorithms. The degree of the polynomial dictates the number of parameters in the regression model. A high degree can result in overlearning (overfitting) the training data due to a high number of parameters in the model. Whereas, a low degree can result in underlearning (underfitting) the training data due to not having enough parameters. In the next section, we will describe the concepts of linear models and inferences regarding this class of model in detail.

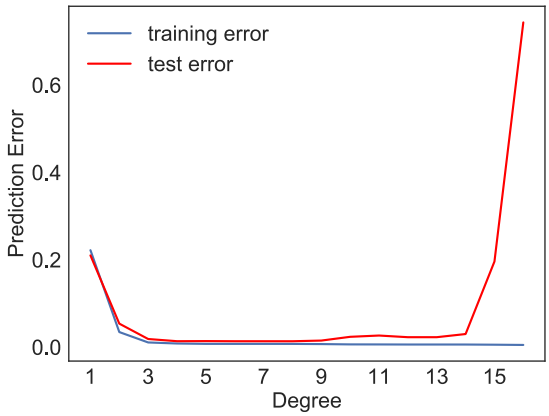

Figure 6.2: The blue line represents the training error and the red line the test error. The test error is computed using 100 simulated observations other than the training sample. The linear model is inadequate here with a high training error (underfitting) and high test error. On the other extreme, a polynomial with a high degree learns the training data too well (overfitting) and fails to generalize to independent test data. The optimal degree is around 5 in this case.

Linear Model

In this section, we introduce the *linear model* in which we postulate a linear relationship between the independent variables and the response. Mathematically, given a vector of covariates $X = (X_1, \cdots, X_p)$, we predict the response variable Y from X via a linear relationship:

$$Y = \beta_0 + \beta_1 X_1 + \cdots \beta_p X_p + \epsilon. \tag{6.7}$$

Suppose that we have a sample of n observations $(x_1, y_1), \cdots (x_n, y_n)$. Then the model says that, for each $i \in (1, \cdots, n)$,

$$y_i = \beta_0 + \beta_1 X_{1i} + \cdots \beta_p X_{pi} + \epsilon_i. \tag{6.8}$$

Using matrix notations, we can compactly write the linear system (6.8) as

$$\mathbf{y} = \mathbf{X}\beta + \epsilon, \tag{6.9}$$

where

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}_{n\times 1}, \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{21} & \cdots & x_{p1} \\ 1 & x_{12} & x_{22} & \cdots & x_{p2} \\ & \vdots & \vdots & & \vdots \\ 1 & \vdots & \vdots & \ldots & \vdots \\ 1 & x_{1n} & x_{2n} & \ldots & x_{pn} \end{bmatrix}_{n\times p}, \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix}_{p\times 1}, \epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}_{n\times 1}. \tag{6.10}$$

The X matrix is frequently referred to as the *design matrix*, β is the vector of parameters (coefficients) of the linear model.

The parameter β is often estimated to optimize an *objective function (loss function)*. The objective function is a function of the parameters and the dataset. As a result, we can obtain different estimates with different objective functions. The most popular objective function by far is based on the squared distances,

$$l(\beta) = (\mathbf{y} - \mathbf{X}\beta)'(\mathbf{y} - \mathbf{X}\beta) = \sum_{i=1}^{n} (y_i - x_i'\beta)^2, \tag{6.11}$$

where x_i is the i^{th} row of the design matrix $\mathbf{X}$. The residual corresponding to an observation i is defined to be $\epsilon_i = y_i - x_i^T\beta$. The least squares method

aims to choose the β that minimizes the residual sum of squares. The *least squares (LS) estimate* corresponding to the objective function in (6.11) can be reformulated as

$$\hat{\beta} = \arg\min_{\beta} (y - X\beta)'(y - X\beta) \tag{6.12}$$

Differentiating the objective function with respect to β, we obtain the *normal equations*

$$\mathbf{X}'(\mathbf{y} - \mathbf{X}\beta) = 0 \tag{6.13}$$

If the matrix $X'X$ is of full rank (non-singular), the unique solution is given by

$$\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}. \tag{6.14}$$

The LS estimator $\hat{\beta}$ is an unbiased estimate of β. [2] In addition, the *Gauss-Markov*'s theorem asserts that among all linear unbiased estimates, the least squares estimates of β have the smallest variance. Given a new observation x_0 of X, we predict the average value of Y to be

$$\hat{f}(x_0) = x_0'\hat{\beta}. \tag{6.18}$$

After fitting a linear model to the data, one might ask: how well does the model fit the data? A common metric that is used for assessing the performance of a linear model is called *R-squared*, which is defined to be the proportion of variance explained by the model. Mathematically,

$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS}, \tag{6.19}$$

where $TSS = \sum_{i=1}^{n}(y_i - \bar{y})^2$ is the *total sum of squares* and $RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ is the residual sum of squares. It can be shown that for a simple linear regression model with one predictor, the R-squared is the square of the correlation between Y and X. R-squared ranges from 0 to 1. If $R^2 = 1$, the model perfectly explains the variability of the response variable. If $R^2 = 0$, then the model does not explain any variability of the response variable. One potential draw back of R-squared is that it increases with the number of predictors. In other words, a model with many covariates can overfit the data and thus, reduce the residual sum of squares. The *adjusted* R-squared defined by $R^2_{adj} = 1 - \frac{RSS/(n-p-1)}{TSS/(n-1)}$ takes the number of covariates in the model into account by adjusting the residual sum of squares and the total sum of squares by their corresponding degrees of freedom.

[2] Proof that the LS estimator is unbiased.

$$E[\hat{\beta}] = E[(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}\mathbf{y}] \tag{6.15}$$
$$= E[(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}(\mathbf{X}\beta + \epsilon)] \tag{6.16}$$
$$= \beta + E[(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}\epsilon]. \tag{6.17}$$

If $\mathbf{X}$ and ϵ are uncorrelated and $E[\epsilon] = 0$, then $E[\hat{\beta}] = \beta$

Inference

The LS estimate can be obtained without having to assume the true distribution of the data. In order to perform parametric statistical inference on linear models, we need to assume that $\mathbf{X}$ is fixed (non random), the random errors ϵ_i are uncorrelated across observations and have constant variance (homoscadastic) σ^2. It is easily shown[3] that

$$\mathrm{Var}(\beta) = (\mathbf{X}'\mathbf{X})^{-1}\sigma^2. \tag{6.23}$$

In order to make an inference about $\hat{\beta}$, we need to make the additional assumption that the errors are normally distributed: $\epsilon_i \sim \mathrm{N}(0, \sigma^2)$. This allows us to show that the LS estimate follows a multivariate normal distribution,

$$\hat{\beta} \sim \mathrm{MVN}(\beta, (\mathbf{X}'\mathbf{X})^{-1}\sigma^2). \tag{6.24}$$

[3] Proof. Using the identity, $\mathrm{Var}(AX + B) = A\mathrm{Var}(X)A'$ for any random variable X and non-random matrices A and B, we have

$$\mathrm{Var}(\hat{\beta}) = \mathrm{Var}((\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}) \tag{6.20}$$
$$= (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathrm{Var}(y)\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1} \tag{6.21}$$
$$= (\mathbf{X}'\mathbf{X})^{-1}\sigma^2 \tag{6.22}$$

The last equality follows from the fact that $\mathrm{Var}(y) = \mathrm{Var}(\epsilon) = \sigma^2\mathrm{I}$.

The variance of the error term σ^2 can be estimated by

$$\hat{\sigma}^2 = \frac{1}{N-p-1}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{6.25}$$

where $\hat{y}_i = x_i'\hat{\beta}$. It can be shown that the estimate $\hat{\sigma^2}$ is an unbiased estimator of σ^2 and follows a chi-squared distribution

$$(N-p-1)\hat{\sigma}^2/\sigma^2 \sim \chi^2_{N-p-1}. \tag{6.26}$$

Power Law Revisited

In Chapter 4, we learned that the power of EEG signals follows the $1/f$ law; i.e., power $\propto 1/f^{\alpha}$. We are going to demonstrate this property by running a regression analysis of *log*-power versus *log*-frequency using a large intracranianial EEG (iEEG) data sets comprising recordings of human brain activity during the encoding periods of a free-recall task. The distribution of α across 110 FR1 subjects (Figure 6.4) demonstrates heterogeneity in iEEG signals with the exponent ranging from 0.5 to 3.5.

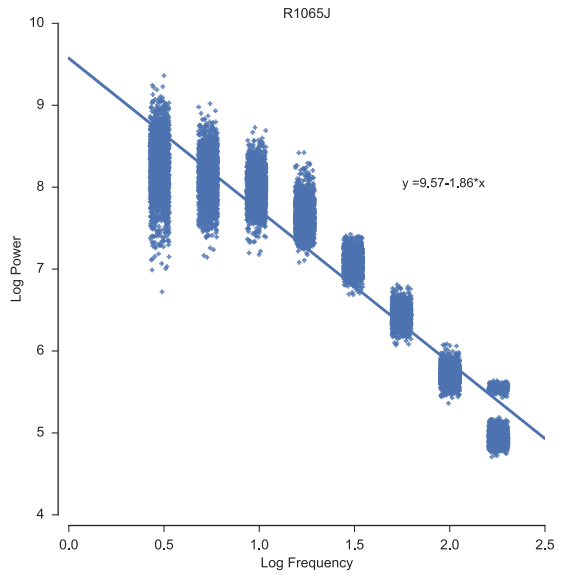

Figure 6.3: Log-power versus Log-frequency for one subject $R^2 = 0.9$, $p = 0.01$, the 95% confidence interval for the coefficient is 1.85 ± 0.01.

Hypothesis Testing

Being equipped with the distributional properties of $\hat{\beta}$, we can perform hypothesis testing on the individual coefficients β_j. Let's consider a null hypothesis $\beta_j = 0$, we can construct a test statistic

$$z_j = \frac{\hat{\beta}_j}{se(\hat{\beta}_j)}, \tag{6.27}$$

where $se(\beta_j)$ is just the j^{th} diagonal element of $(\mathbf{X}'\mathbf{X})^{-1}\hat{\sigma}^2$. z_j can be shown to follow a t distribution with $N-p-1$ degrees of freedom. The randomness of the estimate $\hat{\beta}$ comes from the randomness of the sample

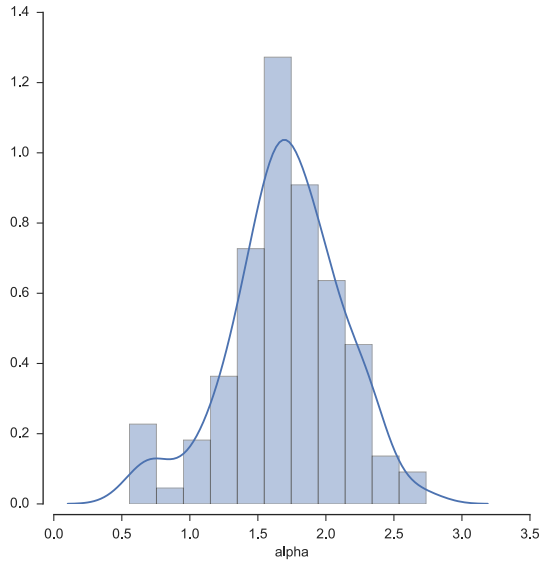

Figure 6.4: Distribution of the Exponent α. We fit regression models of log-power versus log-frequency for 8 frequencies log-spaced between 3 and 180 Hz for 110 FR1 subjects.

$\{(x_1, y_1), \cdots, (x_n, y_n)\}$. If we were to take another sample of n observations, we would expect the $\hat{\beta}$ for this sample to be different from the previous one. In other words, the sampling variability of the estimate arises from the inability to sample the whole population at a time. We can construct a 95% confidence interval for $\hat{\beta}_j$:

$$(\hat{\beta} - 1.96 \cdot \text{se}(\hat{\beta}), \hat{\beta} + 1.96 \cdot \text{se}(\hat{\beta})) \tag{6.28}$$

Sometimes, we want to test whether the set of covariates are useful at all; i.e,

$$H_0 : \beta_1 = \cdots = \beta_p = 0$$
$$H_a : \text{ at least one } \beta_j \text{ is non-zero}$$

We compute the *F-statistic*,

$$F = \frac{(TSS - RSS)/p}{RSS/(n-p-1)} \sim F_{p,n-p-1} \tag{6.29}$$

where TSS is the total sum of squares and RSS is the residual sum of squares. The F-statistic follows an F-distribution with degrees of freedom p and $N - p - 1$. One rejects the null hypothesis when the F-statistic exceeds a certain threshold given a certain significance level.

In addition, one can test a subset of $q < p$ coefficients,

$$H_0 : \beta_{i_1} = \cdots = \beta_{i_q} = 0$$

We use the partial F-statistic to test this hypothesis. First, we fit a regression model that uses all but the q covariates being tested and obtain its associated residual sum of squares $RSS_{reduced}$. Then, we compare $RSS_{reduced}$ to the

residual sum of squares of the model using all p covariates RSS_{full}. The F-statistic for this test can be constructed as follows:

$$F = \frac{(RSS_{reduced} - RSS_{full})/(q)}{RSS_{full}/(n-p-1)} \sim F_{q,n-p-1} \tag{6.30}$$

Again, we reject the null hypothesis when the F-statistic exceeds a certain threshold determined by the significance level α and the sampling distribution $F_{q,n-p-1}$. The idea of the partial F-test is that if the set of q predictors is significant, then including this set into the model will significiantly reduce the residual of sum of squares after accouting for the number of predictor in each model.

Interpretation of Regression Coefficients

Let us recall our linear regression model

$$Y = \beta_0 + \beta_1 X_1 + \cdots \beta_p X_p + \epsilon. \tag{6.31}$$

Each coefficient β_j is usually interpreted as the average effect on Y for every unit increase in X_j while holding other variables fixed. However, covariates tend to be correlated in practice. As a result, a change in one covariate is likely to induce changes in others. A more accurate interpretation of regression coefficients is the following: the coefficient β_j represents the additional contribution of x_j on y after accounting for the contributions of other predictors (see Hastie et al. (2009) for more details).

Bootstrap

In chapter 3, we introduced the bootstrap method for testing the difference between event-related potentials under different experiment conditions. One of the main advantages of the bootstrap method is that it does not require any assumption about the underlying distribution of the data. Also, the bootstrap can be applied to cases where the sampling distribution of a test statistic is too difficult to derive. The parametric framework for testing hypotheses in regression relies on several assumptions. First, this framework assumes that the linear model is the correct in order to estimate $\hat{\sigma}^2$ in Equation 6.25. Second, the errors are assumed to be uncorrelated and normally distributed. We can devise different bootstrap schemes corresponding to how much we "trust" the model. This section covers two popular bootstrap methods for assessing the variability of the estimates of the regression coefficients. The first bootstrap scheme is called *residual resampling bootstrap*. As indicated by its name, the residual resampling bootstrap method constructs bootstrap samples of the data by first calculating the residuals of the regression performed on the original sample (Y, X) and then resampling the estimated residuals with replacement and adding them back to the the fitted values. Specifically, we regress Y on X and estimate residuals $\epsilon_i = y_i - \mathbf{x}_i'\hat{\beta}$ for $i \in \{1, \cdots, n\}$. Algorithm 1 summarizes the method.

The sample distribution of $\hat{\beta}$ can be estimated using the bootstrap sampling distribution $\{\hat{\beta}^b\}_{b=1}^B$. The residual resampling bootstrap relies on the assumption that the linear model is correct and avoids the normality assumption on the error terms. The method fails when the model is not linear. Alternatively, one can resample the observations $\{(\mathbf{x}_i, y_i)\}$ without having

Algorithm 1: Residual Resampling Bootstrap

For $b = 1, \cdots, B$

1: Resample $\{\epsilon_1^b\}$ from $\{\epsilon_i\}$ with replacement.
2: Create a new sample (Y^b, X) by adding the resampled residuals back to the fitted values $y_i^b = \mathbf{x}_i'\hat{\beta} + \epsilon_i^b$.
3: Estimate $\hat{\beta}^b$ by regressing Y^b on X.

to assume a specific form of the regression function (*observation resampling*). This approach only assumes that the observations are independent. Algorithm 2 summarizes the method.

Algorithm 2: Observation Resampling Bootstrap

For $b = 1, \cdots, B$

1: Create a new sample (Y^b, X) by resampling the rows of (Y, X) with replacement
2: Estimate $\hat{\beta}^b$ by regressing Y^b on X.

Figure 6.5 demonstrates the two bootstrap methods: one resamples the residuals and the other one resamples the observations. The bootrapped samples using residual sampling yields a narrower 95% confidence interval compared to that from using the observation sampling. This is not surprising since a linear regression model is the correct model for this particular dataset. As a result, the residual sampling should produce a confidence interval that is very close to the one computed using asymptotic theory and normality assumption. However, the observation resampling also produces a very good confidence interval even though the technique does not assume any underlying distribution and any structure of the data.

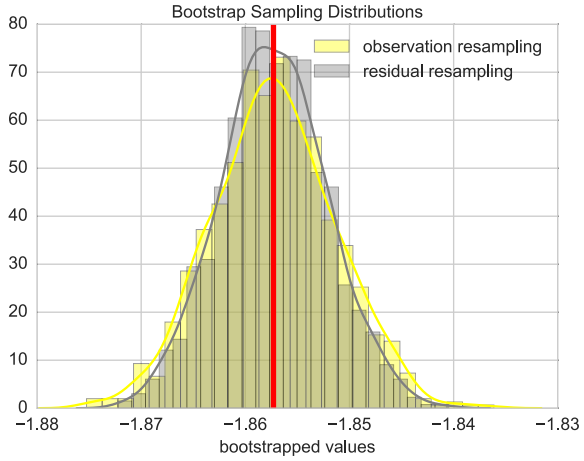

Figure 6.5: Bootstrap Confidence Intervals. The bootstrapped 95% confidence intervals (B=1000) using residual sampling and observation resampling are: -1.85 ± 0.010 and -1.85 ± 0.012.

Application: Predicting list level recall performance.

In a typical free recall experiment, participants study multiple lists of items and each study list is followed by a recall period where participants are instructed to recall as many items as they can remember from the current list (see Chapter 1). Even when an effort is made to equate the study lists with respect to variables that could affect recall performance (list length, item difficulty, etc.), the proportion of recalled items can vary dramatically from list to list (Kahana, Aggarwal, & Phan, 2018). This variability in performance can be observed for repeated tests within an individual (i.e., even without contributions from inter-individual variability) and for this example we therefore investigated the extent to which we are able to predict list-level recall performance from brain activity during the presentation of the study lists for an individual participant in the LTPFR2 study described earlier (Chapter 1). For this example, we calculated power across 15 frequencies logarithmically spaced between 2 and 200 Hz from scalp EEG recordings during the presentation of study items. We averaged the log and z-transformed power values across all study episodes in each list to use as features in a ridge regression model predicting list-level recall performance. Because the proportion of recalled items is constrained to fall between 0 and 1, we trained the ridge regression model to predict the logit-transformed proportion of recalled words ($\text{logit}\left(p(\text{rec})\right)$).[4]

The following sections will discuss the logic of regularization and procedures for determining the size of the regularization parameter. For this example we set the regularization parameter α to 250 and evaluated the performance of our ridge regression model using a leave-one-session-out cross-validation procedure (also discussed in more detail in the following sections). Figure 6.6 plots the actual list-level recall performance (ranging from no recall to perfect recall of all 24 items) against the predictions from our ridge regression model (both back-transformed from logits to probabilities for convenience). Whereas the range of predicted list-level recall performance is noticeably smaller than than that of the actual list-level recall performance and there is considerable variability in the predicted performance for each level of actual performance, it is clear that there is some relationship between actual and predicted recall performance. As explained in the main text, there are different ways to quantify this relationship. Figure 6.6 lists the correlation between actual and predicted recall performance ($r = 0.43$) and another popular metric is the coefficient of determinant R^2 which can be calculated as $1 - \frac{\text{RSS}}{\text{TSS}}$ where RSS and TSS are residual and total sum of squares respectively.[5] For our example the RSS and TSS were 691 and 848 respectively leading to $R^2 = 0.19$ which corresponds closely to the squared correlation between actual and predicted recall performance. For a comprehensive example of using brain activity to predict list-level recall performance see Weidemann and Kahana (2019b).

[4] $\text{logit}\left(p(\text{rec})\right) = \log\left(\frac{p(\text{rec})}{1-p(\text{rec})}\right)$. To avoid invalid values for lists in which all or none of the words were recalled we set $p(\text{rec})$ for those cases to 0.999 and 0.001 respectively.

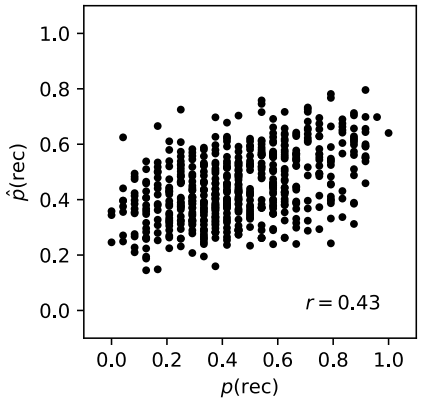

Figure 6.6: Actual and predicted list-level recall performance. The probability of list-level recall ($p(\text{rec})$) for all lists across all sessions for participant 344 in the LTPFR2 experiment, plotted against corresponding predicted list-level recall performance ($\hat{p}(\text{rec})$) from the ridge regression model described in the text. The correlation between the actual and predicted list-level performance across the 576 lists is indicated in the lower left.

[5] The RSS is the sum of the squared differences between actual and predicted values whereas the TSS is the sum of the squared deviations of the actual values from their mean. Perfect prediction corresponds to $R^2 = 1$ with a model whose predictions match the expected value regardless of input features yielding $R^2 = 0$. When calculated this way R^2 values can become negative for models producing predictions that are worse than always predicting the expected value.

Overfitting, Underfitting and The Bias-variance Decomposition

A model with many parameters, such as a high-degree polynomial regression model, can overfit the training data and fail to generalize to independent test data. Whereas, a too simple model can fail to learn the training data. The concepts of overfitting and underfitting can be described precisely by the two

theoretical properties of the estimated function $\hat{f}$: *bias* and *variance*. Consider a regression problem: $y = f(X) + \epsilon$, where $\mathrm{E}[\epsilon] = 0$ and $\mathrm{Var}[\epsilon] = \sigma^2$ and suppose that we have a prediction function $\hat{f}$ of f by applying some learning algorithm to a training data set.

The prediction error loss of a new point x_0 with $y_0 = f(x_0) + \epsilon_0$ under the squared error loss is

$$\mathrm{E}[y_0 - \hat{f}(x_0)]^2 = \mathrm{E}[f(x_0) - \mathrm{E}\hat{f}(x_0) + E\hat{f}(x_0) - \hat{f}(x_0) + \epsilon_0]^2 \tag{6.32}$$

$$= [\mathrm{E}\hat{f}(x_0) - f(x_0)]^2 + \mathrm{E}[\hat{f}(x_0) - \mathrm{E}\hat{f}(x_0)]^2 + \mathrm{Var}[\epsilon^2] \tag{6.33}$$

$$= \mathrm{Bias}^2 + \mathrm{Variance} + \mathrm{Irreducible\ Error}. \tag{6.34}$$

The term $\mathrm{E}\hat{f}(x_0) - f(x_0)$ is called the bias of the prediction function $\hat{f}$ since it measures the average deviation from the ground truth. $\mathrm{E}[\hat{f}(x_0) - \mathrm{E}\hat{f}(x_0)]^2$ is called the variance of $\hat{f}$ since it measures how much the estimate $\hat{f}$ changes with new training data. $\mathrm{Var}[\epsilon^2]$ is irreducible error due to noise and it cannot be eliminated regardless of how well we are able to estimate f. In general, we want a learning algorithm that has low bias and low variance. However, there is usually a trade-off between bias and variance. Algorithms with low bias tend to have more parameters and thus high variance. On the other hand, algorithms with low variance tend to have high bias. Overfitting occurs when an algorithm learns the training sample too well (low bias) but fails to generalize to an independent test set due to high variance. Whereas, underfitting occurs when an algorithm fails to learn the training sample (high bias) often as a result of having a very simple model (low variance). In practice, we are more concerned with overfitting since sophisticated algorithms often have many parameters and can overfit then training data set. Methods for reducing variance such as penalization are used to combat overfitting even though they might incur bias.

Cross Validation

We have seen that the training error is not a reliable estimate for prediction error since a flexible algorithm with many parameters can learn the training sample very well. The test error truly reflects how an algorithm performs on an independent data set. Therefore, it is essential that we be able to estimate the prediction error correctly. One of the challenges that we face as researchers is sparse data due to resource-constraints or unavailability of participants. Therefore, we want to use as much collected data as possible to build a machine learning model. In an ideal situation when there was enough data, we would set aside a test set for assessing the performance of an algorithm, which is not possible in most cases. One way to by-pass this sparsity problem is to *hold out* a portion of the available data for testing and to use the remaining part of the data for training. This method is called *cross-validation*. Cross-validation is one of the simplest ways to estimate prediction error and has been proven to be very effective. To facilitate the discussion of cross-validation, we split the available data into K folds of roughly equal sizes. For the k^{th} fold, we train a model on the remaining $K - 1$ folds to obtain an estimator $\hat{f}^{-k}$ and then compute the prediction error for the held-out fold

$$\mathrm{CV}_{error}(\hat{f}^{-k}) = \frac{1}{n_k} \sum_{i \in C_k} l(y_i, \hat{f}^{-k}(x_i)) \tag{6.35}$$

where C_k and n_k denote the set of indices and the number of observations in the k^{th} fold. The cross-validated estimate of the prediction error is given by,

$$\text{CV}_{error}(\hat{f}) = \sum_{i=1}^{K} \frac{n_k}{N} \text{CV}_{error}(\hat{f}^{-k}). \tag{6.36}$$

That is we take a weighted average of the fold-based errors to obtain the cross-validated error. This is called K-fold cross-validation. If $K = N$, the number training sample size, we call this procedure *leave-one-out* cross-validation since each observation is a fold itself. $K = 5, 10$ are among the most popular choices since it only requires retraining the algorithm a few times.

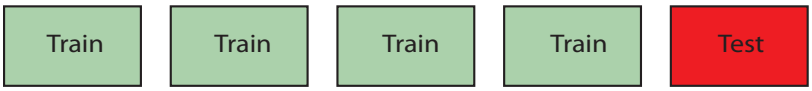

Figure 6.7: 5-fold cross-validation illustration . In this example, we train the machine learning algorithm on the first four folds (green) and then test its performance on the hold-out fold (red).

Regularization and Model Selection

A common problem in machine learning is overfitting, which occurs when a model learns not only the signal but also the noisy component of the training data set. As a result, the model fails to predict unseen future data (test set). In Figure 6.1, the polynomial regression model of degree 15 overfits the training set by learning the its idiosyncrasy and the shape of the predicted function $\hat{f}$ is heavily influenced by the noisy component in the training set. In the context of regression analysis, when there are many correlated predictors in the model, the estimates of the coefficients will be poorly determined and have high variance due to the design matrix being closer to being singular. Highly correlated coefficients with very large magnitudes can occur and they can balance each other to give predictions in a reasonable range. A common way to combat overfitting is to utilize regularization techniques which impose penalization on large coefficients. Regularization techniques are ubiquitous in machine learning and can be applied to complicated models such as logistic regression, trees, deep neural networks, etc. In this section, we will introduce regularization in the context of regression. The penalized regression model can be formulated as follows:

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{N} (y_i - \mathbf{x}_i'\beta)^2 + \lambda \text{Penalty}(\beta). \tag{6.37}$$

That is we find $\hat{\beta}$ that it minimizes the sum of squared error loss and the penalty imposed the value of β. One of the most common penalty term for β is the L_q norm: $||\beta||_q^q = \beta_1^q + \cdots + \beta_p^q$. The learning model is called *Lasso* regression when $q = 1$ and *Ridge* regression when $q = 2$. The penalty term λ controls how much we want to penalize large coefficients. Let us look at two extreme cases.When $\lambda = 0$, we do not penalize large coefficients at all and the estimated $\hat{\beta}$ is just our usual least squared coefficients. On the other hand, as $\lambda \to \infty$, the estimate regression coefficients approach o since the algorithm severely penalize non-zero coefficients.

Regularization is a technique that reduces the variances of the coefficients by constraining their sizes. However, it does not come without a price. Regularization introduces bias into the estimates. To elucidate this point, let us

consider L_2 (ridge) penalized regression. The coefficient vector $\hat{\beta}$ is estimated to minimize the following objective function:

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{N} (y_i - \mathbf{x}_i'\beta)^2 + \lambda\beta'\beta. \tag{6.38}$$

We can rewrite Equation 6.39 in a compact matrix form,

$$\hat{\beta} = \arg\min_{\beta} (\mathbf{y} - \mathbf{X}\beta)'(\mathbf{y} - \mathbf{X}\beta) + \lambda\beta'\beta \tag{6.39}$$

where $\mathbf{y}$, $\mathbf{X}$ are defined as in Chapter **??**. The ridge regression solution has a close-form solution

$$\hat{\beta}^{ridge} = (X'X + \lambda I)^{-1}X'y \tag{6.40}$$

where I is a $p \times p$ identify matrix. Again, the ridge solution is a linear function of $\mathbf{y}$. When $\lambda = 0$, we recover the usual least square solution. When $\lambda > 0$, the ridge solution is no longer an un-biased estimator of β due to the λI term in the solution. As λ gets bigger, the ridge solution becomes more biased but it variance becomes smaller. This is a archetypal example of the bias-variance trade-off.

A natural question to ask when applying penalized regression is: how do we choose the penalty parameter λ. To answer this question, we first notice that different values of λ represent different learning models. To choose to best model (as a function of lambda), we need to compare models according to some performance metric. Let us consider MSE as the metric. We can use cross-validation (say with 10 folds) to estimate the MSE for each model and then choose the λ that maximizes the MSE.

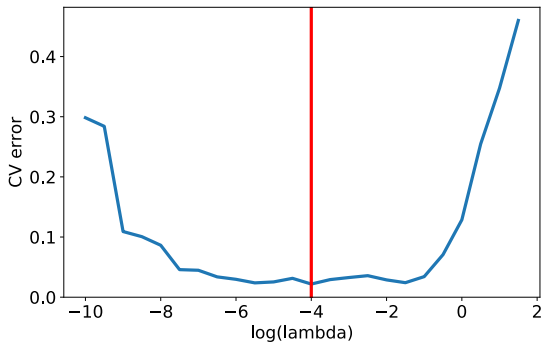

Figure 6.8: Here is an example of using cross-validation to choose the hyperparameter λ. We simulate 30 observations from the sinusoidal function as in Figure 6.1.

Example of list-level regression in intracranial data

Illustration of some of the above issues, such as overfitting, data leakage, etc.

Section to be completed!

Logistic Regression

So far, we have only considered regression models in which the response is a continuous random variable. Oftentimes, we are interested in modeling

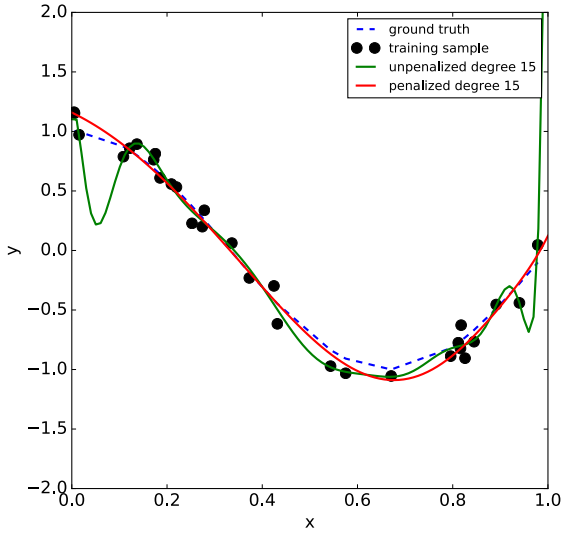

Figure 6.9: Example of penalized regression. A polynomial regression with degree 15 performs well with penalization and overfits the training data without penalization. The optimal penalty parameter λ is chosen using cross-validation.

categorical outcomes such as whether a person recalls a word presented at an earlier time, or whether the animal in a picture is a cat or not a cat, etc. Machine learning models that learn to discriminate between categories fall in the group of classification models. Classification models are still supervised learning models that have labels to guide the learning process, but the labels are discrete instead of being continuous. In this section, let us consider a task of classifying a binary variable that can be either 1 (indicating the target class) or 0 (indicating the non-target class). We note that there is nothing special about 1 and 0 or about which of the two classes is designated the target class. One can switch the labels and model the probability of the non-target class instead. Recall that the optimal function $\hat{f}$ that minimizes the squared error loss is $\mathrm{E}[y \mid X = x]$. The k-nearest neighbors algorithm approximates the conditional mean by averaging the labels around a neighborhood of x. The regression models assume that the conditional mean is a linear combination of the predictions: $\mathrm{E}[y \mid X = x] = \beta_0 + \sum_{j=1}^{p} \beta_j X_j$. One can attempt to approximate the conditional mean as a linear function of the predictors when y is categorical. However, there is a problem with this approach. It is easy to see that $E[y \mid X = x] = P(y = 1 \mid X = x)$. Therefore, the conditional mean is restricted to the range $[0, 1]$ since it is a probability now. As a result, modeling $P(y = 1 \mid X = x) = \beta_0 + \sum_{j=1}^{p} \beta_j X_j$ is problematic because the right hand side can easily be out of the range $[0, 1]$. Let $\mu(x)$ denote the conditional mean $\mathrm{E}[y \mid X = x]$. If we take the log-transformation of $\mu(x)$, the range of $\mu(x)$ is in $(-\infty, 0)$, which is a much wider range but the value of $\mu(x)$ is still bounded above. This type of transformation will work well if one believes that the probability of interest is not close to 1. In order to relax the range of $\mu(x)$ even further, one can consider the *logit* transform

$$\mathrm{logit}(\mu(x)) = \log \frac{\mu(x)}{1 - \mu(x)}, \tag{6.41}$$

which has a range from $-\infty$ to ∞. When we model the logit-transform of the conditional mean as a linear function of the predictors, we obtain a class of logistic regression models. Logistic models have been well-studied and applied to a variety of classification problems and are proven to work very well when there is not a lot of data. Different transformations of $\mu(x)$ yield different models. Here we introduce the *generalized linear models* in which we model a transformation h of $\mu(x)$ as a linear function of the predictors:

$$h(\mu(x)) = \beta_0 + \sum_{j=1}^{p} \beta_j X_j. \tag{6.42}$$

h is called the link function between $\mu(x)$ and the predictors. A variety of models fall into this class of generalized linear models. For example,

- $h(x) = x$, we have the identity link and simple linear models.
- $h(x) = \log(x)$, we have a class of log-linear models that are typically used to model count data.
- $h(x) = \text{logit}(x)$, we have a class of logistic regression models for modeling binomial probabilities.

Let us focus on logistic regression models and suppose that $p(x) = \text{E}[y \mid X = x]$ and we model $p(x)$ as a linear function of the predictors

$$\log \frac{p(x)}{1 - p(x)} = x'\beta \tag{6.43}$$

If we solve for $p(x)$ in terms of x and β, we obtain

$$p(x) = \frac{1}{1 + \exp(-x'\beta)} \tag{6.44}$$

The function $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the *sigmoid function*. In other words, the predicted probability of the positive class in a logistic regression model is a sigmoid function of the linear function of the predictors. Figure 6.19 illustrates the sigmoid function.

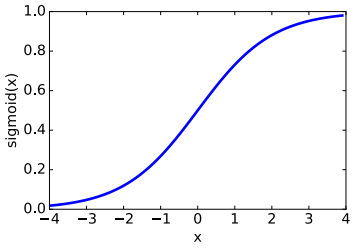

Figure 6.10: The sigmoid function takes in real inputs and returns outputs in the range [0,1] .

Loss Function

Suppose we have a training data set $(x_1, y_1), \cdots, (x_n, y_n)$. For each pair (x_i, y_i), the probability of observing y_i given x_i is

$$P(y = y_i \mid X = x_i) = p_i^{y_i}(1 - p_i)^{1-y_i}, \tag{6.45}$$

where $p_i = \frac{1}{1 + \exp(-x_i'\beta)}$. The loss function is defined to be the log-likelihood function

$$\begin{aligned} l(\beta) &= \log \prod_{i=1}^{N} P(Y = y_i \mid X = x_i) \\ &= \sum_{i=1}^{N} \{y_i \log p_i + (1 - y_i) \log(1 - p_i)\} (\text{ negative cross-entropy function }) \\ &= \sum_{i=1}^{N} \{y_i x_i'\beta - \log(1 + \exp(x_i'\beta))\} \end{aligned} \tag{6.46}$$

We want to maximize the log-likelihood function l. The optimal solution $\hat{\beta}$ for the loss function l does not have a close-formed. Therefore, we need to solve for β numerically. Hastie et al. (2009) covers the details of some the numerical methods used to estimate β.

Once we obtain our estimate $\hat{\beta}$, the prediction for a new observation x_0 is given by

$$\hat{y}_0 = \frac{1}{1 + \exp(-x_0'\beta)}. \tag{6.47}$$

That is we predict the response for x_0 has a probability $\hat{y}_0$ of observing the target class (labeled by 1).

Penalized Logistic Regression and Nested Cross Validation

Classification models also suffer from overfitting when there is a large number of predictors present. Similar to regression, one can try to prevent overfitting by penalizing large coefficients and this can be done by introducing a penalty term into the objective function in Equation 6.78:

$$l(\beta) = \sum_{i=1}^{N} \{y_i x_i'\beta - \log(1 + \exp(x_i'\beta))\} + \lambda \text{Penalty}(\beta) \ , \tag{6.48}$$

where λ is the penalty parameter that controls the degree to which the algorithm penalizes large values. Again, the penalty is typically the L_q norm of β for some $q > 0$. The most popular choices for q are 1 and 2.

The penalty term λ can be chosen by cross-validation to optimize cross-validated errors. As before we split the data set into K folds. For each λ, we repeatedly train on $K - 1$ folds and test the last fold to obtain the cross-validated errors $CV_{error}(\lambda)$ as a function of λ. We choose the λ that minimizes the CV_{error}.

One has to be very careful when comparing different models with hyper-pameters (penalty parameter, learning rate, etc.) on the same data set. The $CV_{error}(\lambda)$ corresponding to some optimal λ is an optimistic estimation of the prediction error since the algorithm with the optimal λ has "seen" the entire data set. In order to have an honest assessment of an algorithm, one either needs to chose the hyper parameter(s) on the basis of the classifier's performance in a separate validation data set that is distinct from both training and tests sets or use a nested cross-validation approach, where the effects of different hyper-parameters are evaluated only within the training set and the classifier's and the best parameter values are then used to determine the classifier's performance on the held-out test set.

Example: Predicting subsequent memory

We can use the techniques described in this chapter to investigate to what extent neural activity during the encoding of an item predicts subsequent recall. As described in Chapter 1, successful recall is sensitive to a wide range of variables during both study and retrieval. Recordings of ongoing brain activity provide unique insights into memory processes as they unfold, but, given the numerous determinants of successful recall, it is unlikely that measures of brain activity during encoding periods could predict it with very high precision, even if we were able to measure brain activity with much higher fidelity than current neuroscientific techniques allow. Nevertheless, it

is likely that brain activity during encoding contains some information about how well the item is processed and later remembered. It is not clear a priori, however, what the relevant signals would be. Furthermore, diagnostic signals might be weak in isolation and only able to predict subsequent recall reliably in concert with other such features—an ideal use-case for a classifier that learns the importance of each feature from the training data.

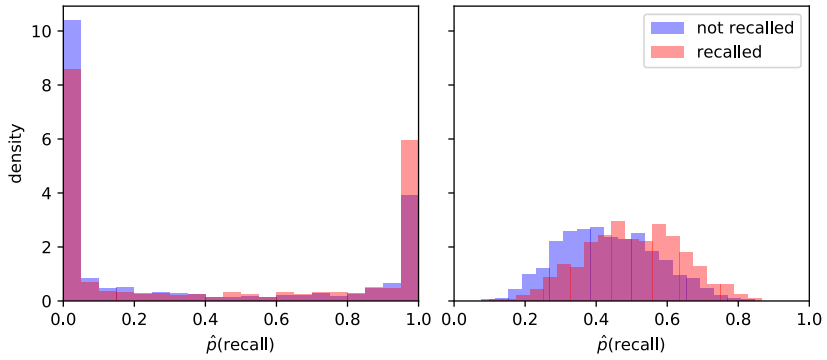

Figure 6.11: Normalized densities of predicted probabilities of subsequent recall from penalized logistic regression classifiers. The classifiers were trained on iEEG activity during encoding in a single patient for two different levels of regularization. At the lower level of regularization (left; $\lambda = 1$), predicted probabilities tend to be extreme, whereas less extreme predicted probabilities are more common with stronger regularization (right; $\lambda = 1000$). Despite considerable overlap between the distributions at both levels of regularization, it is clear that predicted probabilities of subsequent recall tended to be larger for items that were later recalled indicating that the classifiers were able to distinguish encoding events on the basis of subsequent recall.

We trained a logistic regression classifier on average power in a range of frequencies between 5 and 80 Hz during the presentation of

each word in the encoding phase of the FR1 task. Figure 6.20 illustrates predicted probabilities of subsequent recall separately for items that were subsequently recalled and those that were not for one participant and two levels of regularization. It is clear from the figure that the level of regularization has a strong effect on the distribution of predicted probabilities and the overlap between these distributions for the target and non-target class. It is also clear that the predicted probabilities tend to be larger for the target class than for the non-target class indicating that the classifier has extracted diagnostic features that distinguish between the two classes. The figure also suggests that the distributions of predicted probabilities for the two classes are better separated at the higher level of regularization—we will now turn to discussing how to quantify this intuition.

Model evaluation

When evaluating the performance of a classifier, it is tempting to simply calculate the proportion of accurate classifications. It is easy to show, however, that accuracy alone is not particularly informative. Consider the task of classifying brain activity during encoding with respect to subsequent recall. If a classifier is described as having correctly predicted subsequent recall for 85% of the considered encoding events, it might appear to have extracted meaningful encoding-related brain activity. Such a level of performance could also be achieved, however, if the classifier always predicted subsequent recall and 85% of studied words were indeed recalled. Whereas this classifier correctly predicted subsequent memory for each item that was later recalled, it got every failed recall wrong. When assessing classifier performance, it is therefore important to consider the different types of errors a classifier can make.

In Chapter 3 we introduced the concept of Type I and Type II errors in the context of statistical tests. Just as there are four different possible

outcomes in hypothesis testing (two types of correct decisions and two types of errors; see Table 3.1), there are four types of possible outcomes in any binary classification task, which can be captured by a confusion matrix (see Table 6.2). Correct predictions of targets are labeled *true positives* or *hits* whereas incorrect predictions of exemplars from the non-target category as targets are labeled *false positives* or *false alarms*. Because the total number of target (n_T) and non-target ($n_{\neg T}$) items is known, the hit and false alarm rates completely determine the confusion matrix. Thus, a comprehensive assessment of classifier performance can be achieved by taking both of these measures into account.

	Target	Non-target
"yes"	true positives (hits)	false positives (false alarms)
"no"	false negatives (misses)	true negatives (correct rejections)
Total	n_T	$n_{\neg T}$

Table 6.1: Confusion matrix for a binary classification task; "yes" and "no" labels indicate classifier predictions for and against the target class respectively with "target" and "non-target" labels reflecting the true class membership. The *hit rate* is number of true positives, divided by n_T and the *false alarm* rate is the number of false positives divided by $n_{\neg T}$. In the machine learning literature, the hit rate is sometimes referred to as *recall* and in the context of medical tests it is known as *sensitivity* (with *specificity* corresponding to the correct rejection rate).

Receiver operating characteristics

As illustrated above, all binary classification tasks share the same basic structure and thus the problem of assessing a classifier's performance is quite general with applications across a wide range of fields, including engineering, psychology, and statistics. Signal Detection Theory (Green & Swets, 1966) is a framework for analyzing performance in detection tasks that was built on work studying the characteristics of radar receiver operators during World War II. As the gain on a radar receiver is increased, it is increasingly sensitive to relevant objects (e.g., enemy aircraft), but also increasingly likely to display spurious signals (Streiner & Cairney, 2007). Receiver operating characteristic (ROC) functions describe the performance of a classifier as the threshold for target identification is varied by relating false positive rates to true positive rates.

Just as the probability of a false alarm increases for radar receiver operators as the gain on the radar receiver is increased, psychological variables are important determinants of the balance between true and false positive responses in human classifiers. In memory research, binary classification is the predominant method for assessing recognition memory (see, Chapters 1 and **??**). The probability to endorse a probe item as previously studied (and thus the hit and false alarm rates) vary with variables such as the base rates of previously studied and new probe items and the reward structure of the task (e.g., setting the reward for hits to be higher than the loss associated with making false alarms can increase the proportion of probe items endorsed as studied). The output of a logistic regression classifier is a probability which can be used to make a binary classification decision. As the threshold for endorsing the target class decreases from 1.0 to 0.0 the balance of hits and false alarms associated with these thresholds traces out the ROC function. Figure 6.21 shows the ROC functions for the classifiers whose predictions are presented in Figure 6.20.

In investigations of recognition memory, the shape of the ROC function derived from human recognition responses has been of great interest, because it can provide clues about the underlying processes (see Chapter **??**). Here we will focus on a simple and popular index of classifier performance that can be derived from it: the area under the ROC curve (AUC). Classifiers that cannot distinguish between the two classes, produce completely overlapping distributions of predictions and hence the resulting ROC function falls on the main diagonal in ROC space (because hit and false alarm rates are identical at each threshold). The resulting AUC is 0.5 whereas perfect separation of the two classes produces an AUC of 1.0.[6] Generally, the AUC reflects

[6] In case of substantial overfitting, performance on a held-out data set can sometimes fall below 0.5. For human classifiers, performance that falls considerably below 0.5 usually indicates a failure to follow instructions, e.g., by mixing up response keys.

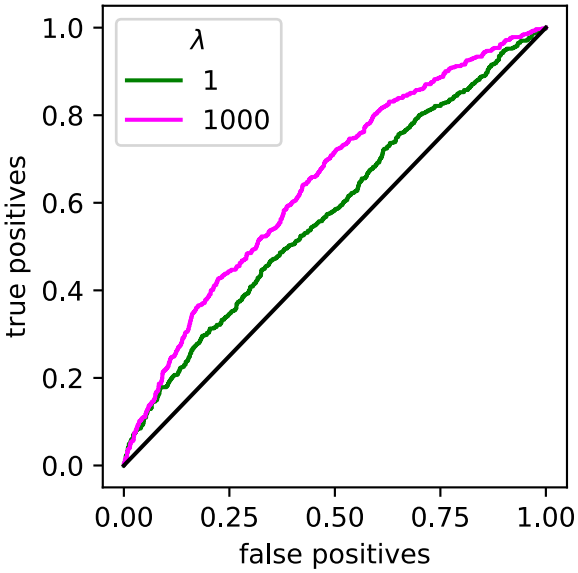

Figure 6.12: Receiver operating characteristic functions for the classifiers whose predictions are shown in Figure 6.20. This plot illustrates clearly that the classification performance is higher for the classifier with stronger regularization ($\lambda = 1000$). The corresponding areas under the ROC curves (AUC) are 0.58 and 0.65 for $\lambda = 1$ and $\lambda = 1000$ respectively. The main diagonal indicating chance performance is also displayed.

the probability that a randomly chosen target instance will be associated with a higher predicted probability than a randomly chosen non-target instance, making it a directly interpretable ("normalized") index of classifier performance. Furthermore, the AUC is the statistic computed for the Wilcoxon test of ranks and directly related to the Gini coefficient. Fawcett (2006) presents a thorough overview of ROC analyses that illustrates these properties and, among other things, addresses generalizations to multi-class classification problems, comparisons to other methods for assessing classifier performance, and statistical issues associated with ROC data.

Other Machine Learning Methods

Very brief summary of the overall enterprise of machine learning as it builds upon the basic regression models. What are the broadest categories of methods, and how can we characterize them in a relatively non-technical manner. Perhaps SVMs should be described with some formality, but the rest of the methods could be just described in narrative form. Then we can set up and describe the results of the comparison between l1, l2, svm, rf and sgboost. Finally, we can conclude this section with a brief discussion of the pros and cons of each methods and pointers to some good standard treatments.

Here we apply five commonly-used machine learning methods to the basic problem of predicting recall of items based upon neural features recorded during the study phase of an experiment. The data for this example consists of 20 subjects who performed at least three sessions of a free-recall tasks. We trained classifiers to discriminate between good versus bad memory-encoding using spectral features derived from activity recorded at many brain locations. We used a cross-validation approach in which we train the classifier on $N - 1$ sessions and then test it on the remaining session, repeating this until we obtain out-of-sample predictions for every session. We then compute an ROC curve relating true and false positives as a function of the

criterion used to assign regression output to responses labels.

All five algorithms reliably classified subsequent memory performance based on neural features recorded during item encoded. Figure **??** illustrates the area under the ROC curve for each classification algorithm (this is a standard measure of classification accuracy). Although we obtained the best performance for the L2-penalized (euclidean distance) logistic regression classifier, that does not mean that this classifier generally outperforms the others. The differences between classifier performance more likely reflects the statistical structure of the noise in the features being used. Thus, with different neural features one may well have observed a different ranking between the algorithms.

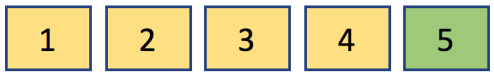

Figure 6.13: Cross-validation scheme. For each subject, we train our classifier on all (yellow) but one session and test the performance on the hold-out session (green) and repeat the procedure for each session.

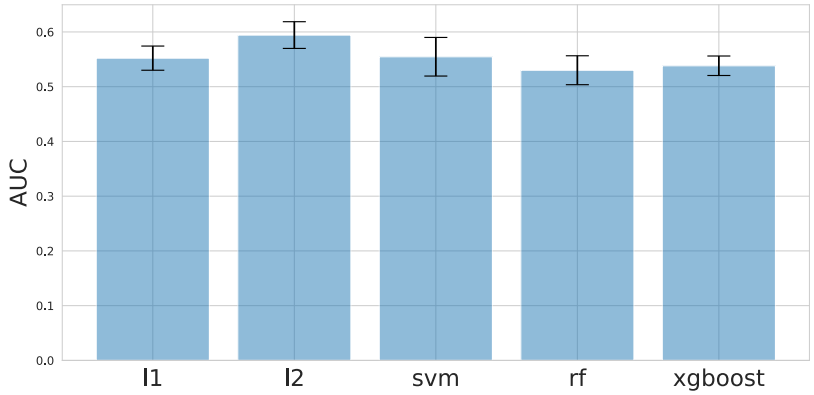

Figure 6.14: Classification Performance for Various Machine Learning Algorithms

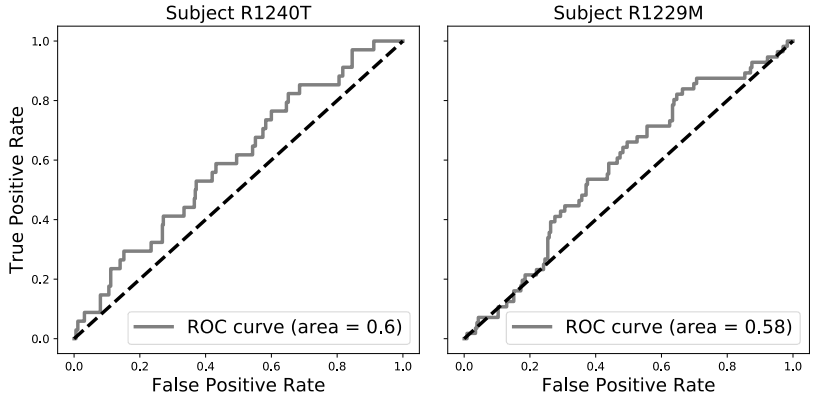

Figure 6.15: Sample ROC cruves (gray) of L2 penalized logistic classifiers for 2 intracranial subjects who performed a free-recall task. The dashed black lines indicate ROC curves of at-chance classifiers.

Feature Engineering

In all of the examples presented in this chapter, we used spectral power at several (eight) frequency bands and at each bipolar electrode pair as our "features" for decoding brain states. Prior work using univariate methods had already established spectral features as statistically reliable correlates of memory performance. But we could have made other choices and in this short section we consider the more general problem of feature selection or feature engineering.

Machine learning is model fitting and the more features one uses, the more complex the model becomes and the more likely the model will overfit the data, leading to poor generalization. Penalization and cross validation help, but sometimes you can use prior knowledge about the data, from other studies or from the statistical structure of the data itself, to improve feature selection.

If, for example, you knew that two features were extremely highly correlated, and that each was subject to independent sources of noise, then including both features would be unlikely to benefit classifier generalization enough to warrant the overfitting that would occur by fitting to each features noise during the training phase. In this case, you would prefer to throw out redundant features. One systematic technique for doing this involves transforming your original features vectors into another vector space of lower dimensionality that has less redundancy between features. You might start by examinign the correlation matrix between your features and look at the highest correlations. You could write a computer program that systematically asks how well each variable can be predicted by the other variables and then throw out those variables with the highest R^2 values. But this would also involve overfitting and would be highly senstivie to the order in which you iteratively discard variables. A much better method would be to use a dimensionality reduction technique, such as principal components analysis (PCA), as described below.

Using PCA for feature selection

Time-series analysis (OPTIONAL)

Whereas the preceding analysis of the multiple regression problem we asked how multivariate brain activity measured during a given time interval, $X(t)$ predicts a continuous behavioral measure, denoted $Y(t)$, such as the proportion of items recalled on the studied list. Although the lists here unfold over time, this regression problem assumes that each list is an independent observation, with neighboring lists and remotely studied lists being modeled as independent of one another. I think many readers will find this to be a highly unrealistic assumption. Many things that may be correlated with both brain activity and behavior change slowly over time, or at least exhibit some type of temporal organization. Here we generalize the regression approach to model the statistical relations among variables measured at different time points.

Neural signals comprise a time series of observations along each of the channels (electrodes) being recorded. Before we introduce the formal linear autoregressive model, let us consider two assumptions that we will be mak-

ing about neural data: Our first assumption is that the state of the brain, and consequently measures of its electrical potentials, change slowly over time. Thus, the difference between the voltage measured at electrode i at times t and $t + \tau$ should smoothly approach zero as the interval τ approaches zero. Our second assumption is that the time variation in brain states is not perfectly predictable based on past states. Mathematically, this lack of perfect predictability implies that we are studying a stochastic rather than a deterministic process. The models we will be working with in this chapter are thus special cases of the more general class of models known as stochastic processes. As in any situation of fitting a model to data, we recognize that our model is not perfect and in fact may be woefully incomplete. However, by capturing some essential features of the true (but unknown) data generating process of the brain, the models give us some insights into the nature of the measurements being studied.

In our previous discussion of time-frequency decomposition methods we did not take any specific stance on the nature of the data generating process that gives rise to the brain electrical signals. However, by bringing the signals into the frequency domain we made the tacit assumption that our data contained rhythmic signals that could be extracted using these decomposition techniques. However, electrical recordings of brain activity need not, and frequently do not, exhibit clear evidence of rhythmic activity. In such cases, we can still measure the power or energy within different frequency bands, but we would be wrong to interpret those band-defined signals as being oscillatory in nature.

The so-called parametric (or model-based) approach that we will introduce in the present section offers an alternative way of thinking about neural signals. By putting structure on the data-generating process, these models allow you to potentially extract more information from a noisy signal that would be possible without a model-based approach. However, if the time-series model does *not* provide a reasonable approximation of the data generating process, then the results will be misleading. However, careful study of a flawed model can also be of great value in helping to uncover features of the data that we may otherwise miss. In the end of this chapter we will discuss approaches to overcome some of the major limitations of the autoregressive approach introduced at the outset.

One of the most striking features of the spectral decomposition of the time series of neural recordings taken at any spatial scale is a power-law relation between spectral power and frequency, such that the $\text{Power}(f) \propto f^{-\alpha}$ where α is frequently found to be in the range of 0.5 to 3.5. This power law is a standard property of an AR(p) process.

Linear time-series models (AR)

Autoregressive models (abbreviated AR) assume that the EEG signals can be written as linear combinations of a certain number (the model order) of the previous signals. Suppose that we observe a time series of signals $\{X[t]\}$ at a channel. An autoregressive model of order p for this channel can be written as:

$$X[t] = \alpha + \sum_{i=1}^{p} b[i]X[t-i] + \epsilon[t], \tag{6.49}$$

where $b[i]$, $i = 1, \cdots, p$, are the coefficients of the AR model, and $\epsilon[t]$ is normally distributed with mean 0 and variance σ^2. The temporal dynamics of the EEG signal is fully characterized by the set of parameters in the AR model: $\theta = (\alpha, \beta[1], \cdots, \beta[p], \sigma^2)$. [7]

The AR framework is a model-based approach in which we assume a priori that the EEG signals are drawn from a data-generating process governed by Equation 6.49. This assumption is the main difference between the model-based approach in this present chapter and the model-free approach (time-frequency analysis) in the previous chapter.

Stationarity The analysis of time-series models usually depends on the assumption of *stationarity*. Stationarity, in layman's terms, means that if we take arbitrary "snapshots" of the time series, we would expect these snapshots to exhibit similar behavior such as having the same mean, variance, autocorrelational structure. Formally speaking, a time series X is *strict* stationarity if the joint distribution of any portion of the time-series $(X[t_1], \cdots, X[t_k])$ is the same as the distribution of any of its translation $X[t_1 + \tau], \cdots, X[t_k + \tau]$ for any integer τ. Strict stationarity is usually hard to verify empirically because one has to test the invariance of the distribution of any snapshot of the time-series over time. It usually suffices to assume *weak* stationarity which only requires constant mean and constant covariance over time. In other words, the expected value $\mathrm{E}(X[t])$ is time-invariant and the covariance $\mathrm{Cov}(X[t], X[s])$ only depends on $t - s$. We will assume weak stationarity for the analyses in the following sections.

Autocorrelation Function Recall that the correlation between two random variables X and Y, given by the formula $\rho = \mathrm{Corr}(X, Y) = \dfrac{\mathrm{Cov}(X, Y)}{\sqrt{\mathrm{Cov}(X)\mathrm{Cov}(Y)}}$, measures the linear dependence between X and Y. [8] Using the Cauchy-Schwartz's inequality, one can show that $-1 \leq \rho \leq 1$. If $|\rho| = 1$, then X can predict Y perfectly through a linear relationship and vice versa. Assuming the time series $X[t]$ is stationary and has finite variance σ^2, the *autocovariance* function (ACVF) for the time-series $\{X[t]\}$ at two time points $s \leq t$ is defined by

$$\gamma(s, t) = \mathrm{Cov}(X[s], X[t]). \tag{6.52}$$

Under weak stationarity, $\gamma(s, t)$ only depends on the temporal distance between the two time points $l = |t - s|$. As a result, one can define the autocovariance function γ_l as a function of the *lag* l:

$$\gamma_l = \mathrm{Cov}(X[t - l], X[t]). \tag{6.53}$$

The lag l autocorrelation function (ACF) can be defined by

$$\rho_l = \frac{\mathrm{Cov}(X[t - l], X[t])}{\sqrt{\mathrm{Var}(X[t - l])\mathrm{Var}(X[t])}} = \frac{\gamma_l}{\sigma^2}. \tag{6.54}$$

Suppose that the time series $X[t]$ has length T and mean $\bar{X}$. The ACVF and ACF functions can be estimated from the time series using their empirical estimates.

$$\hat{\gamma}_l = \sum_{t=l+1}^{T} (X[t] - \bar{X})(X[t - l] - \bar{X}) \tag{6.55}$$

$$\hat{\rho}_l = \frac{\sum_{t=l+1}^{T} (X[t] - \bar{X})(X[t - l] - \bar{X})}{\sum_{t=1}^{T} (X[t] - \bar{X})^2}, \quad 0 \leq l < T - 1 \tag{6.56}$$

[7] One we express the conditional distribution of $X[t]$ given its p previous values $X[t-1], \cdots, X[t-p]$ as:

$$p(X[t] \Big| X[t-1], \cdots, X[1], \theta) \tag{6.50}$$

$$= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x_t - \mu)^2 / 2\sigma^2}, \tag{6.51}$$

where $\mu = \alpha + \sum_{i=1}^{p} b[i] X[t - i]$

[8] The covariance of two random variables X and Y is defined to be $\mathrm{Cov}(X, Y) = E[(X - E[X])(Y - E[Y])]$.

9

Hypothesis Testing with ACF For any lag l, one can test $H_0 : \rho_l = 0$ versus $H_a : \rho_l \neq 0$. The test statistic is:

$$t = \frac{\hat{\rho}_l}{\sqrt{(1 + 2\sum_{i=1}^{l-1} \hat{\rho}_i^2)/T}}. \tag{6.57}$$

The t-statistic above is based on the Barlett's formula, which shows that if $\{X[t]\}$ is a stationary Gaussian time-series (i.e. the joint distribution of $\{X[t]\}$ at any arbitrary number of points is multivariate normal.), and that if $\rho_i = 0$, for all $i > l$, then the t-statistic is asymptotically normal.

Oftentimes, we are interested in jointly testing several lags of $\{X[t]\}$. Ljung-Box's test (Box, Jenkins, Reinsel, and Ljung (2015)) has been widely used in the econometrics literature for such purposes. We can apply the same principle to EEG data. Here we specify our null and alternative hypotheses,

$$H_0 : \rho_1 = \cdots = \rho_l = 0 \tag{6.58}$$

$$H_a : \rho_i \neq 0, \quad \text{for some } i \in \{1, \cdots, l\}. \tag{6.59}$$

The Q-statistic is defined to be

$$Q(l) = T(T+2) \sum_{i=1}^{l} \frac{\hat{\rho}_i^2}{T-i}. \tag{6.60}$$

It can be shown that $Q(l)$ is asymptotically a chi-squared random variable with l degrees of freedom under certain moment conditions on $\{X[t]\}$. We reject the null when the Q-statistic exceeds a certain threshold governed by a pre-set significance level α.

Properties of AR Models We discuss basic properties of AR models in this section. In particular, we will focus on understanding the AR(1) models, particularly the unconditional mean, variance, and autocorrelation function of these models and then generalize these properties to AR(p) models.

*AR(1) Model We derive the unconditional mean and variance of the model under the weak stationarity assumption. Suppose that a time series of signals $\{X[t]\}$ follows an AR(1) model

$$X[t] = \beta_0 + \beta_1 X[t-1] + \epsilon[t] \tag{6.61}$$

where $\epsilon[t] \sim \mathrm{N}(0, \sigma^2)$. Under weak stationarity, the mean, variance, and covariance do not change with time. In order words,

$$\mathrm{E}(X[t]) = \mu, \quad \mathrm{Var}[X[t]] = \gamma_0, \quad \mathrm{Cov}(X[t], X[t-l]) = \gamma_l, \quad \text{for all } t. \tag{6.62}$$

Taking expectation of both sides of Equation 6.61 and using stationarity condition of the mean,

$$\mu = \beta_0 + \beta_1 \mu \tag{6.63}$$

Solving for μ and assuming that $|\beta_1| < 1$, we get $\mu = \dfrac{\beta_0}{1-\beta_1}$. Similarly, $\gamma_0 = \dfrac{\sigma^2}{1-\beta_1^2}$, and $\rho_l = \beta_1 \rho_{l-1}$. Indeed, the necessary and sufficient condition for an AR(1) model to be stationary is $|\beta_1| < 1$. For a general AR(p) model, the unconditional mean is given by

$$\mu = \frac{1}{1 - \beta_1 - \cdots - \beta_p}. \tag{6.64}$$

and the ACF function ρ_l satisfies the following recursive relation

$$\rho_l = \beta_1 \rho_{l-1} + \cdots + \beta_p \rho_{l-p}, \quad \text{for } l > 0 \tag{6.65}$$

We can solve for ρ_l using the associated *characteristic equation* of the model

$$1 - \beta_1 x - \cdots \beta_p x^p = 0. \tag{6.66}$$

Depending on the solution of the above equation, the ACF can exhibit a mixture of damping sine and cosine patterns and exponential decays.

Example

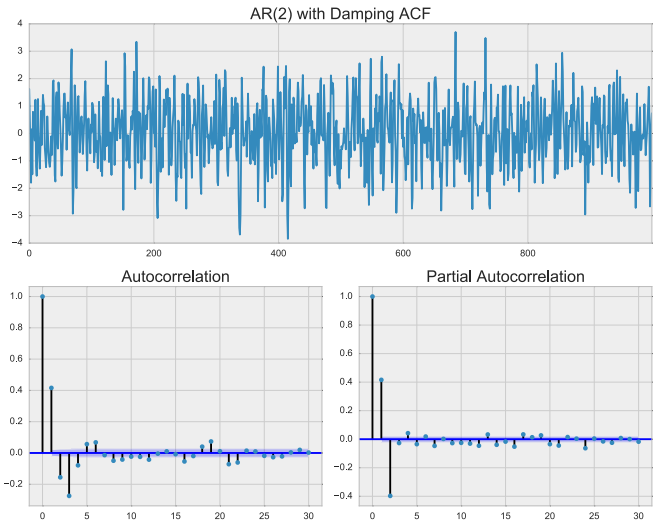

Figure 6.16: AR(2) model with complex roots. $\beta_1 = 0.6, \beta_2 = -0.4$. The characteristic equation of this model has complex roots. As a result, the ACF displays a mixture decaying and sinusoidal pattern.

Estimation of AR coefficients

The coefficents in an AR(p) model can be easily estimated using a regression framework. The AR(p) has the following specification

$$X[t] = \beta_0 + \beta_1 X[t-1] + \cdots \beta_p X[t-p] + \epsilon[t], \quad \text{for } t = p+1, \cdots, T. \tag{6.67}$$

The random observation at time t, $X[t]$, can be thought of as the reponses variable, and $X[t-1], \cdots, X[t-p]$ as the predictors. Therefore, the coefficients can be estimated by fitting a linear regression model to the time series data.

Once we obtain the regression estimates of the AR(p) models detoted by $\hat{\beta}_0, \cdots, \hat{\beta}_p$, we can define a residual series $\{r_t\}$,

$$r_t = X[t] - \hat{X}[t], \tag{6.68}$$

where $\hat{X}_t = \hat{\beta}_0 + \hat{\beta}_1 X[t-1] + \cdots \hat{\beta}_p X[t-p]$. If the AR(p) model is correctly specified, the residual series should behave like a white noise process. We can use the Ljung-Box statistics and the Q-statistic to check the closeness of the residual series to white noise.

Order Assessment and Partial Autocorrelation Function

One of the challenges of using the AR models is determining the appropriate order. The ACF function tells us how observations in the time series are correlated over time. However, a lot of the correlations (especially) for higher lags are due to propagation. One of the most common tools to determine the order is the *partial autocorrelation function* (PACF). The PACF is constructed based on a forward stepwise procedure for choosing variables in a regression. The idea is to start with an AR(1) model and successively add more lag-terms until the model's performance does not improve significantly.

$$X[t] = \beta_{00} + \beta_{01} X[t-1] + \epsilon[t] \tag{6.69}$$
$$X[t] = \beta_{10} + \beta_{11} X[t-1] + \beta_{12} X[t-2] + \epsilon[t] \tag{6.70}$$
$$\vdots = \vdots \tag{6.71}$$
$$X[t] = \beta_{p0} + \beta_{p1} X[t-1] + \cdots + \beta_{pp} X[t-p] + \epsilon[t] \tag{6.72}$$

We fit each model seperately. The estimated regression coefficient $\hat{\beta}_{jj}$ is called the lag-j sample PACF of the time series $\{X_t\}$. One can test to see whether the AR(j+1) model is better than the AR(j) model by using an F-test. For a time series generated from a stationary AR(p) model, it can be shown that $\hat{\beta}_{pp}$ converges to the true $\hat{\beta}_p$ as the sample size T goes to infinity and that $\hat{\beta}_{jj} = 0$ for $j > p$.

Example

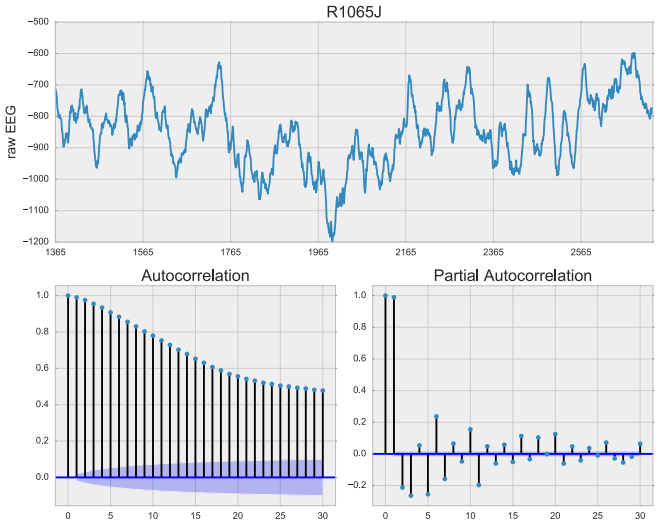

Figure 6.17: AR model fitted to real EEG signals. The top plot shows the raw time series recorded from an epileptic patient while performing the free-recall task. The ACF and PACF show that the series is non-stationary with the lag-1 correlation close to 1.

Logistic Regression

So far, we have only considered regression models in which the response is a continuous random variable. Oftentimes, we are interested in modeling categorical outcomes such as whether a person recalls a word presented at

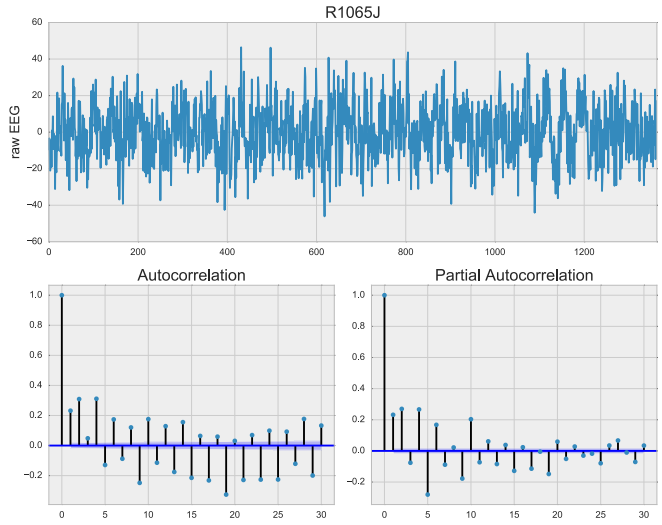

Figure 6.18: AR model fitted to the first-difference of EEG signals. The top plot shows the first-difference of raw time series recorded from an epileptic patient while performing the free-recall task. The difference series looks stationary. The ACF and PACF show that the series have several significal lags up to 30, which is expected since EEG signals are oscillatory.

an earlier time, or whether the animal in a picture is a cat or not a cat, etc. Machine learning models that learn to discriminate between categories fall in the group of classification models. Classification models are still supervised learning models that have labels to guide the learning process, but the labels are discrete instead of being continuous. In this section, let us consider a task of classifying a binary variable that can be either 1 (indicating the target class) or 0 (indicating the non-target class). We note that there is nothing special about 1 and 0 or about which of the two classes is designated the target class. One can switch the labels and model the probability of the non-target class instead. Recall that the optimal function $\hat{f}$ that minimizes the squared error loss is $\mathrm{E}[y \mid X = x]$. The k-nearest neighbors algorithm approximates the conditional mean by averaging the labels around a neighborhood of x. The regression models assume that the conditional mean is a linear combination of the predictions: $\mathrm{E}[y \mid X = x] = \beta_0 + \sum_{j=1}^{p} \beta_j X_j$. One can attempt to approximate the conditional mean as a linear function of the predictors when y is categorical. However, there is a problem with this approach. It is easy to see that $E[y \mid X = x] = P(y = 1 \mid X = x)$. Therefore, the conditional mean is restricted to the range $[0, 1]$ since it is a probability now. As a result, modeling $P(y = 1 \mid X = x) = \beta_0 + \sum_{j=1}^{p} \beta_j X_j$ is problematic because the right hand side can easily be out of the range $[0, 1]$. Let $\mu(x)$ denote the conditional mean $\mathrm{E}[y \mid X = x]$. If we take the log-transformation of $\mu(x)$, the range of $\mu(x)$ is in $(-\infty, 0)$, which is a much wider range but the value of $\mu(x)$ is still bounded above. This type of transformation will work well if one believes that the probability of interest is not close to 1. In order to relax the range of $\mu(x)$ even further, one can consider the *logit* transform

$$\mathrm{logit}(\mu(x)) = \log \frac{\mu(x)}{1 - \mu(x)}, \tag{6.73}$$

which has a range from $-\infty$ to ∞. When we model the logit-transform of the conditional mean as a linear function of the predictors, we obtain a class of logistic regression models. Logistic models have been well-studied and applied to a variety of classification problems and are proven to work very

well when there is not a lot of data. Different transformations of $\mu(x)$ yield different models. Here we introduce the *generalized linear models* in which we model a transformation h of $\mu(x)$ as a linear function of the predictors:

$$h(\mu(x)) = \beta_0 + \sum_{j=1}^{p} \beta_j X_j. \tag{6.74}$$

h is called the link function between $\mu(x)$ and the predictors. A variety of models fall into this class of generalized linear models. For example,

- $h(x) = x$, we have the identity link and simple linear models.
- $h(x) = \log(x)$, we have a class of log-linear models that are typically used to model count data.
- $h(x) = \text{logit}(x)$, we have a class of logistic regression models for modeling binomial probabilities.

Let us focus on logistic regression models and suppose that $p(x) = \text{E}[y \mid X = x]$ and we model $p(x)$ as a linear function of the predictors

$$\log \frac{p(x)}{1 - p(x)} = x'\beta \tag{6.75}$$

If we solve for $p(x)$ in terms of x and β, we obtain

$$p(x) = \frac{1}{1 + \exp(-x'\beta)} \tag{6.76}$$

The function $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the *sigmoid function*. In other words, the predicted probability of the positive class in a logistic regression model is a sigmoid function of the linear function of the predictors. Figure 6.19 illustrates the sigmoid function.

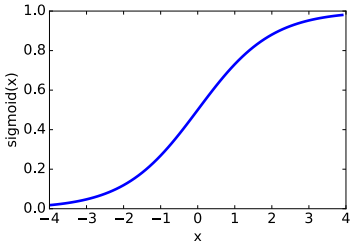

Figure 6.19: The sigmoid function takes in real inputs and returns outputs in the range [0,1] .

Loss Function

Suppose we have a training data set $(x_1, y_1), \cdots, (x_n, y_n)$. For each pair (x_i, y_i), the probability of observing y_i given x_i is

$$P(y = y_i \mid X = x_i) = p_i^{y_i}(1 - p_i)^{1-y_i}, \tag{6.77}$$

where $p_i = \frac{1}{1 + \exp(-x_i'\beta)}$. The loss function is defined to be the log-likelihood function

$$\begin{aligned} l(\beta) &= \log \prod_{i=1}^{N} P(Y = y_i \mid X = x_i) \\ &= \sum_{i=1}^{N} \{y_i \log p_i + (1 - y_i) \log(1 - p_i)\} (\text{ negative cross-entropy function }) \\ &= \sum_{i=1}^{N} \{y_i x_i'\beta - \log(1 + \exp(x_i'\beta))\} \end{aligned} \tag{6.78}$$

We want to maximize the log-likelihood function l. The optimal solution $\hat{\beta}$ for the loss function l does not have a close-formed. Therefore, we need to solve for β numerically. Hastie et al. (2009) covers the details of some the numerical methods used to estimate β.

Once we obtain our estimate $\hat{\beta}$, the prediction for a new observation x_0 is given by

$$\hat{y}_0 = \frac{1}{1 + \exp(-x_0'\beta)}. \tag{6.79}$$

That is we predict the response for x_0 has a probability $\hat{y}_0$ of observing the target class (labeled by 1).

Penalized Logistic Regression and Nested Cross Validation

Classification models also suffer from overfitting when there is a large number of predictors present. Similar to regression, one can try to prevent overfitting by penalizing large coefficients and this can be done by introducing a penalty term into the objective function in Equation 6.78:

$$l(\beta) = \sum_{i=1}^{N} \{y_i x_i'\beta - \log(1 + \exp(x_i'\beta))\} + \lambda \text{Penalty}(\beta) \ , \tag{6.80}$$

where λ is the penalty parameter that controls the degree to which the algorithm penalizes large values. Again, the penalty is typically the L_q norm of β for some $q > 0$. The most popular choices for q are 1 and 2.

The penalty term λ can be chosen by cross-validation to optimize cross-validated errors. As before we split the data set into K folds. For each λ, we repeatedly train on $K - 1$ folds and test the last fold to obtain the cross-validated errors $CV_{error}(\lambda)$ as a function of λ. We choose the λ that minimizes the CV_{error}.

One has to be very careful when comparing different models with hyper-pameters (penalty parameter, learning rate, etc.) on the same data set. The $CV_{error}(\lambda)$ corresponding to some optimal λ is an optimistic estimation of the prediction error since the algorithm with the optimal λ has "seen" the entire data set. In order to have an honest assessment of an algorithm, one either needs to chose the hyper parameter(s) on the basis of the classifier's performance in a separate validation data set that is distinct from both training and tests sets or use a nested cross-validation approach, where the effects of different hyper-parameters are evaluated only within the training set and the classifier's and the best parameter values are then used to determine the classifier's performance on the held-out test set.

Example: Predicting subsequent memory

We can use the techniques described in this chapter to investigate to what extent neural activity during the encoding of an item predicts subsequent recall. As described in Chapter 1, successful recall is sensitive to a wide range of variables during both study and retrieval. Recordings of ongoing brain activity provide unique insights into memory processes as they unfold, but, given the numerous determinants of successful recall, it is unlikely that measures of brain activity during encoding periods could predict it with very high precision, even if we were able to measure brain activity with much higher fidelity than current neuroscientific techniques allow. Nevertheless, it is likely that brain activity during encoding contains some information about how well the item is processed and later remembered. It is not clear a priori, however, what the relevant signals would be. Furthermore, diagnostic signals might be weak in isolation and only able to predict subsequent recall reliably

in concert with other such features—an ideal use-case for a classifier that learns the importance of each feature from the training data.

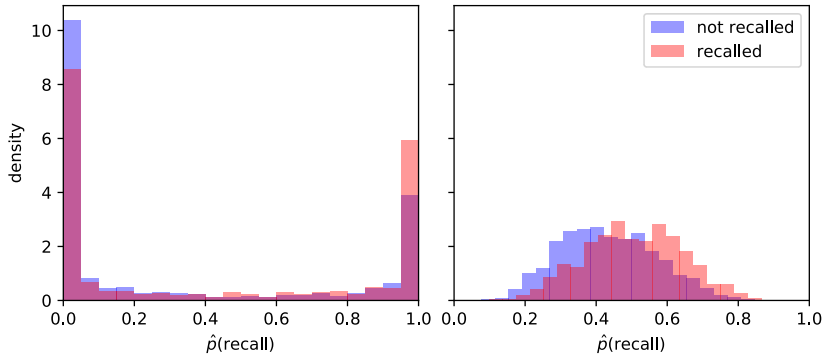

Figure 6.20: Normalized densities of predicted probabilities of subsequent recall from penalized logistic regression classifiers. The classifiers were trained on iEEG activity during encoding in a single patient for two different levels of regularization. At the lower level of regularization (left; $\lambda = 1$), predicted probabilities tend to be extreme, whereas less extreme predicted probabilities are more common with stronger regularization (right; $\lambda = 1000$). Despite considerable overlap between the distributions at both levels of regularization, it is clear that predicted probabilities of subsequent recall tended to be larger for items that were later recalled indicating that the classifiers were able to distinguish encoding events on the basis of subsequent recall.

We trained a logistic regression classifier on average power in a range of frequencies between 5 and 80 Hz during the presentation of

each word in the encoding phase of the FR1 task. Figure 6.20 illustrates predicted probabilities of subsequent recall separately for items that were subsequently recalled and those that were not for one participant and two levels of regularization. It is clear from the figure that the level of regularization has a strong effect on the distribution of predicted probabilities and the overlap between these distributions for the target and non-target class. It is also clear that the predicted probabilities tend to be larger for the target class than for the non-target class indicating that the classifier has extracted diagnostic features that distinguish between the two classes. The figure also suggests that the distributions of predicted probabilities for the two classes are better separated at the higher level of regularization—we will now turn to discussing how to quantify this intuition.

Model evaluation

When evaluating the performance of a classifier, it is tempting to simply calculate the proportion of accurate classifications. It is easy to show, however, that accuracy alone is not particularly informative. Consider the task of classifying brain activity during encoding with respect to subsequent recall. If a classifier is described as having correctly predicted subsequent recall for 85% of the considered encoding events, it might appear to have extracted meaningful encoding-related brain activity. Such a level of performance could also be achieved, however, if the classifier always predicted subsequent recall and 85% of studied words were indeed recalled. Whereas this classifier correctly predicted subsequent memory for each item that was later recalled, it got every failed recall wrong. When assessing classifier performance, it is therefore important to consider the different types of errors a classifier can make.

In Chapter 3 we introduced the concept of Type I and Type II errors in the context of statistical tests. Just as there are four different possible outcomes in hypothesis testing (two types of correct decisions and two types of errors; see Table 3.1), there are four types of possible outcomes in any binary classification task, which can be captured by a confusion matrix (see Table 6.2). Correct predictions of targets are labeled *true positives* or *hits*

whereas incorrect predictions of exemplars from the non-target category as targets are labeled *false positives* or *false alarms*. Because the total number of target (n_T) and non-target ($n_{\neg T}$) items is known, the hit and false alarm rates completely determine the confusion matrix. Thus, a comprehensive assessment of classifier performance can be achieved by taking both of these measures into account.

	Target	Non-target
"yes"	true positives (hits)	false positives (false alarms)
"no"	false negatives (misses)	true negatives (correct rejections)
Total	n_T	$n_{\neg T}$

Table 6.2: Confusion matrix for a binary classification task; "yes" and "no" labels indicate classifier predictions for and against the target class respectively with "target" and "non-target" labels reflecting the true class membership. The *hit rate* is number of true positives, divided by n_T and the *false alarm* rate is the number of false positives divided by $n_{\neg T}$. In the machine learning literature, the hit rate is sometimes referred to as *recall* and in the context of medical tests it is known as *sensitivity* (with *specificity* corresponding to the correct rejection rate).

Receiver operating characteristics

As illustrated above, all binary classification tasks share the same basic structure and thus the problem of assessing a classifier's performance is quite general with applications across a wide range of fields, including engineering, psychology, and statistics. Signal Detection Theory (Green & Swets, 1966) is a framework for analyzing performance in detection tasks that was built on work studying the characteristics of radar receiver operators during World War II. As the gain on a radar receiver is increased, it is increasingly sensitive to relevant objects (e.g., enemy aircraft), but also increasingly likely to display spurious signals (Streiner & Cairney, 2007). Receiver operating characteristic (ROC) functions describe the performance of a classifier as the threshold for target identification is varied by relating false positive rates to true positive rates.

Just as the probability of a false alarm increases for radar receiver operators as the gain on the radar receiver is increased, psychological variables are important determinants of the balance between true and false positive responses in human classifiers. In memory research, binary classification is the predominant method for assessing recognition memory (see, Chapters 1 and **??**). The probability to endorse a probe item as previously studied (and thus the hit and false alarm rates) vary with variables such as the base rates of previously studied and new probe items and the reward structure of the task (e.g., setting the reward for hits to be higher than the loss associated with making false alarms can increase the proportion of probe items endorsed as studied). The output of a logistic regression classifier is a probability which can be used to make a binary classification decision. As the threshold for endorsing the target class decreases from 1.0 to 0.0 the balance of hits and false alarms associated with these thresholds traces out the ROC function. Figure 6.21 shows the ROC functions for the classifiers whose predictions are presented in Figure 6.20.

In investigations of recognition memory, the shape of the ROC function derived from human recognition responses has been of great interest, because it can provide clues about the underlying processes (see Chapter **??**). Here we will focus on a simple and popular index of classifier performance that can be derived from it: the area under the ROC curve (AUC). Classifiers that cannot distinguish between the two classes, produce completely overlapping distributions of predictions and hence the resulting ROC function falls on the main diagonal in ROC space (because hit and false alarm rates are identical at each threshold). The resulting AUC is 0.5 whereas perfect separation of the two classes produces an AUC of 1.0.[10] Generally, the AUC reflects the probability that a randomly chosen target instance will be associated with a higher predicted probability than a randomly chosen non-target instance, making it a directly interpretable ("normalized") index of classifier performance. Furthermore, the AUC is the statistic computed for the Wilcoxon test

[10] In case of substantial overfitting, performance on a held-out data set can sometimes fall below 0.5. For human classifiers, performance that falls considerably below 0.5 usually indicates a failure to follow instructions, e.g., by mixing up response keys.

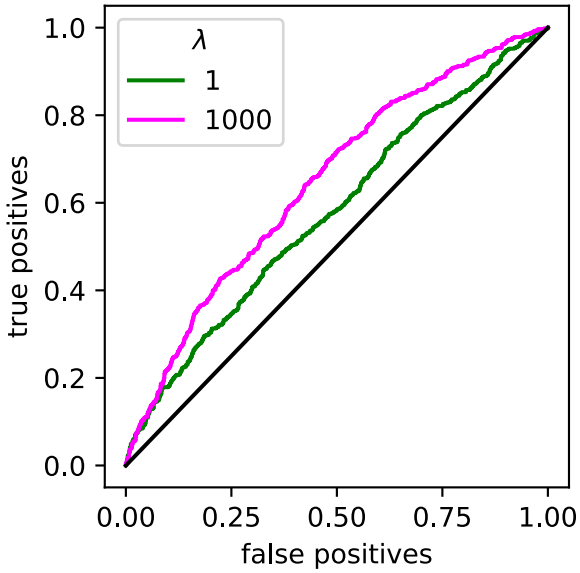

Figure 6.21: Receiver operating characteristic functions for the classifiers whose predictions are shown in Figure 6.20. This plot illustrates clearly that the classification performance is higher for the classifier with stronger regularization ($\lambda = 1000$). The corresponding areas under the ROC curves (AUC) are 0.58 and 0.65 for $\lambda = 1$ and $\lambda = 1000$ respectively. The main diagonal indicating chance performance is also displayed.

of ranks and directly related to the Gini coefficient. Fawcett (2006) presents a thorough overview of ROC analyses that illustrates these properties and, among other things, addresses generalizations to multi-class classification problems, comparisons to other methods for assessing classifier performance, and statistical issues associated with ROC data.

Other Machine Learning Methods

Very brief summary of the overall enterprise of machine learning as it builds upon the basic regression models. What are the broadest categories of methods, and how can we characterize them in a relatively non-technical manner. Perhaps SVMs should be described with some formality, but the rest of the methods could be just described in narrative form. Then we can set up and describe the results of the comparison between l1, l2, svm, rf and sgboost. Finally, we can conclude this section with a brief discussion of the pros and cons of each methods and pointers to some good standard treatments.

Here we apply five commonly-used machine learning methods to the basic problem of predicting recall of items based upon neural features recorded during the study phase of an experiment. The data for this example consists of 20 subjects who performed at least three sessions of a free-recall tasks. We trained classifiers to discriminate between good versus bad memory-encoding using spectral features derived from activity recorded at many brain locations. We used a cross-validation approach in which we train the classifier on $N - 1$ sessions and then test it on the remaining session, repeating this until we obtain out-of-sample predictions for every session. We then compute an ROC curve relating true and false positives as a function of the criterion used to assign regression output to responses labels.

All five algorithms reliably classified subsequent memory performance based on neural features recorded during item encoded. Figure **??** illustrates the area under the ROC curve for each classification algorithm (this is a

standard measure of classification accuracy). Although we obtained the best performance for the L2-penalized (euclidean distance) logistic regression classifier, that does not mean that this classifier generally outperforms the others. The differences between classifier performance more likely reflects the statistical structure of the noise in the features being used. Thus, with different neural features one may well have observed a different ranking between the algorithms.

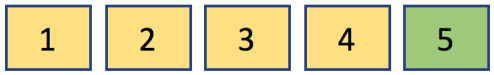

Figure 6.22: Cross-validation scheme. For each subject, we train our classifier on all (yellow) but one session and test the performance on the hold-out session (green) and repeat the procedure for each session.

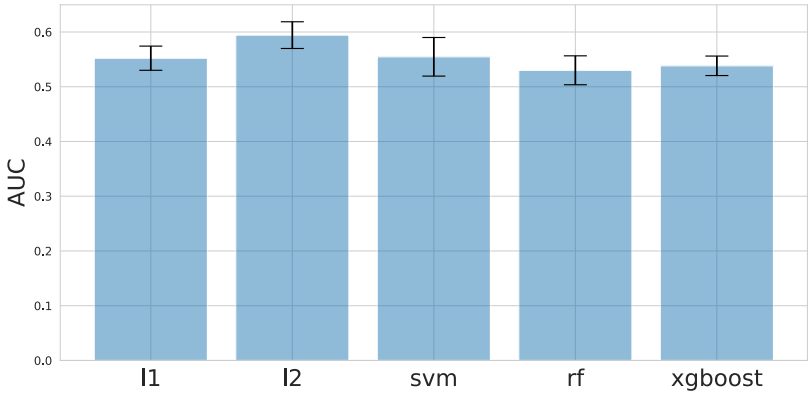

Figure 6.23: Classification Performance for Various Machine Learning Algorithms

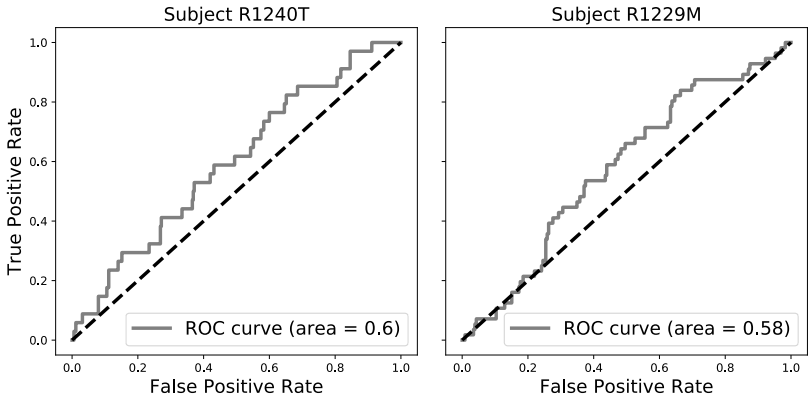

Figure 6.24: Sample ROC cruves (gray) of L2 penalized logistic classifiers for 2 intracranial subjects who performed a free-recall task. The dashed black lines indicate ROC curves of at-chance classifiers.

Feature Engineering

In all of the examples presented in this chapter, we used spectral power at several (eight) frequency bands and at each bipolar electrode pair as our

"features" for decoding brain states. Prior work using univariate methods had already established spectral features as statistically reliable correlates of memory performance. But we could have made other choices and in this short section we consider the more general problem of feature selection or feature engineering.

Machine learning is model fitting and the more features one uses, the more complex the model becomes and the more likely the model will overfit the data, leading to poor generalization. Penalization and cross validation help, but sometimes you can use prior knowledge about the data, from other studies or from the statistical structure of the data itself, to improve feature selection.

If, for example, you knew that two features were extremely highly correlated, and that each was subject to independent sources of noise, then including both features would be unlikely to benefit classifier generalization enough to warrant the overfitting that would occur by fitting to each features noise during the training phase. In this case, you would prefer to throw out redundant features. One systematic technique for doing this involves transforming your original features vectors into another vector space of lower dimensionality that has less redundancy between features. You might start by examinign the correlation matrix between your features and look at the highest correlations. You could write a computer program that systematically asks how well each variable can be predicted by the other variables and then throw out those variables with the highest R^2 values. But this would also involve overfitting and would be highly senstivie to the order in which you iteratively discard variables. A much better method would be to use a dimensionality reduction technique, such as principal components analysis (PCA), as described below.

Using PCA for feature selection