e-Chapter 9

Learning Aides

Our quest has been for a low out-of-sample error. In the context of specific learning models, we have discussed techniques to fit the data in-sample, and ensure good generalization to out of sample. There are, however, additional issues that are likely to arise in any learning scenario, and a few simple enhancements can often yield a drastic improvement. For example: Did you appropriately preprocess the data to take into account arbitrary choices that might have been made during data collection? Have you removed any irrelevant dimensions in the data that are useless for approximating the target function, but can mislead the learning by adding stochastic noise? Are there properties that the target function is known to have, and if so, can these properties help the learning? Have we chosen the best model among those that are available to us? We wrap up our discussion of learning techniques with a few general tools that can help address these questions.

9.1 Input Preprocessing

Exercise 9.1

The Bank of Learning (BoL) gave Mr. Good and Mr. Bad credit cards based on their (Age, Income) input vector.

	Mr. Good	Mr. Bad
(Age in years, Income in thousands of \$)	(47,35)	(22,40)

Mr. Good paid off his credit card bill, but Mr. Bad defaulted. Mr. Unknown who has 'coordinates' (21yrs,\$36K) applies for credit. Should the BoL give him credit, according to the nearest neighbor algorithm? If income is measured in dollars instead of in "K" (thousands of dollars), what is your answer?

One could legitimately debate whether age (maturity) or income (financial resources) should be the determining factor in credit approval. It is, however, decidedly not recommended for something like a credit approval to hinge on an apparently arbitrary choice made during data collection, such as the denomination by which income was measured. On the contrary, many standard design choices when learning from data intend each dimension to be treated equally (such as using the Euclidean distance metric in similarity methods or using the sum of squared weights as the regularization term in weight decay). Unless there is some explicit reason not to do so, the data should be presented to the learning algorithm with each dimension on an equal footing. To do so, we need to transform the data to a standardized setting so that we can be immune to arbitrary choices made during data collection.

Recall that the data matrix $\mathrm{X} \in \mathbb{R}^{n\times d}$ has, as its rows, the input data vectors $\mathbf{x}_1, \ldots, \mathbf{x}_{\text{N}}$, where $\mathbf{x}_n \in \mathbb{R}^d$ (not augmented with a 1),

$$\mathrm{X} = \begin{bmatrix} \text{---} & \mathbf{x}_1^{\text{T}} & \text{---} \\ \text{---} & \mathbf{x}_2^{\text{T}} & \text{---} \\ & \vdots & \\ \text{---} & \mathbf{x}_{\text{N}}^{\text{T}} & \text{---} \end{bmatrix}.$$

The goal of input preprocessing is to transform the data $\mathbf{x}_n \mapsto \mathbf{z}_n$ to obtain the transformed data matrix Z which is standardized in some way. Let $\mathbf{z}_n = \Phi(\mathbf{x}_n)$ be the transformation. It is the data $(\mathbf{z}_n, y_n)$ that is fed into the learning algorithm to produce a learned hypothesis $\tilde{g}(\mathbf{z})$.[1] The final hypothesis g is

$$g(\mathbf{x}) = \tilde{g}(\Phi(\mathbf{x})).$$

Input Centering. Centering is a relatively benign transformation which removes any bias in the inputs by translating the origin. Let $\bar{\mathbf{x}}$ be the in-sample mean vector of the input data, $\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n$; in matrix notation, $\bar{\mathbf{x}} = \frac{1}{N}\mathrm{X}^{\text{T}}\mathbf{1}$ ($\mathbf{1}$ is the column vector of N 1's). To obtain the transformed vector, simply subtract the mean from each data point,

$$\mathbf{z}_n = \mathbf{x}_n - \bar{\mathbf{x}}.$$

By direct calculation, one can verify that $\mathrm{Z} = \mathrm{X} - \mathbf{1}\bar{\mathbf{x}}^{\text{T}}$. Hence,

$$\bar{\mathbf{z}} = \tfrac{1}{N}\mathrm{Z}^{\text{T}}\mathbf{1} = \tfrac{1}{N}\mathrm{X}^{\text{T}}\mathbf{1} - \tfrac{1}{N}\bar{\mathbf{x}}\mathbf{1}^{\text{T}}\mathbf{1} = \bar{\mathbf{x}} - \tfrac{1}{N}\bar{\mathbf{x}}\cdot N = \mathbf{0},$$

where we used $\mathbf{1}^{\text{T}}\mathbf{1} = N$ and the definition of $\bar{\mathbf{x}}$. Thus, the transformed vectors are 'centered' in that they have zero mean, as desired. It is clear that no information is lost by centering (as long as one has retained $\bar{\mathbf{x}}$, one can always recover $\mathbf{x}$ from $\mathbf{z}$). If the data is not centered, we can always center it, so from now on, for simplicity and without loss of generality, we will assume that the input data is centered, and so $\mathrm{X}^{\text{T}}\mathbf{1} = \mathbf{0}$.

[1]Note that higher-level features or nonlinear transforms can also be used to transform the inputs before input processing.

Exercise 9.2

Define the matrix $\gamma = \mathrm{I} - \frac{1}{N}\mathbf{1}\mathbf{1}^{\mathrm{T}}$. Show that $\mathrm{Z} = \gamma\mathrm{X}$. ($\gamma$ is called the centering operator (see Appendix B.3) which projects onto the space orthogonal to $\mathbf{1}$.)

Input Normalization. Centering alone will not solve the problem that arose in Exercise 9.1. The issue there is one of scale, not bias. Inflating the scale of the income variable exaggerates differences in income when using the standard Euclidean distance, thereby affecting your decision. One solution is to ensure that all the input variables have the same scale. One measure of scale (or spread) is the standard deviation. Since the data is now centered, the in-sample standard deviation σ_i of input variable i is defined by

$$\sigma_i^2 = \frac{1}{N}\sum_{n=1}^{N} x_{ni}^2,$$

where x_{ni} is the ith component of the nth data point. Input normalization transforms the inputs so that each input variable has unit standard deviation (scale). Specifically, the transformation is

$$\mathbf{z}_n = \begin{bmatrix} z_{n1} \\ \vdots \\ z_{nd} \end{bmatrix} = \begin{bmatrix} x_{n1}/\sigma_1 \\ \vdots \\ x_{nd}/\sigma_d \end{bmatrix} = \mathrm{D}\mathbf{x}_n,$$

where D is a diagonal matrix with entries $D_{ii} = 1/\sigma_i$. The scale of all input variables is now 1, since $\tilde{\sigma}_i^2 = 1$ as the following derivation shows ($\tilde{\sigma}_i^2 = 1$ is the variance, or scale, of dimension i in the $\mathcal{Z}$ space).

$$\sigma_i^2(\mathbf{z}) = \frac{1}{N}\sum_{n=1}^{N} z_{ni}^2 = \frac{1}{N}\sum_{n=1}^{N} \frac{x_{ni}^2}{\sigma_i^2} = \frac{1}{\sigma_i^2}\cdot\underbrace{\left(\frac{1}{N}\sum_{n=1}^{N} x_{ni}^2\right)}_{\sigma_i^2} = 1.$$

Exercise 9.3

Consider the data matrix X and the transformed data matrix Z. Show that

$$\mathrm{Z} = \mathrm{XD} \qquad \text{and} \qquad \mathrm{Z}^{\mathrm{T}}\mathrm{Z} = \mathrm{D}\mathrm{X}^{\mathrm{T}}\mathrm{X}\mathrm{D}.$$

Input Whitening Centering deals with bias in the inputs. Normalization deals with scale. Our last concern is correlations. Strongly correlated input variables can have an unexpected impact on the outcome of learning. For example, with regularization, correlated input variables can render a friendly target function unlearnable. The next exercise shows that a simple function may require excessively large weights to implement if the inputs are correlated. This means it is hard to regularize the learning, which in turn means you become susceptible to noise and overfitting.

Exercise 9.4

Let $\hat{x}_1$ and $\hat{x}_2$ be independent with zero mean and unit variance. You measure inputs $x_1 = \hat{x}_1$ and $x_2 = \sqrt{1-\epsilon^2}\hat{x}_1 + \epsilon\hat{x}_2$.

(a) What are variance(x_1), variance(x_2) and covariance(x_1, x_2)?

(b) Suppose $f(\hat{\mathbf{x}}) = \hat{w}_1\hat{x}_1 + \hat{w}_2\hat{x}_2$ (linear in the independent variables). Show that f is linear in the correlated inputs, $f(\mathbf{x}) = w_1x_1 + w_2x_2$. (Obtain w_1, w_2 as functions of $\hat{w}_1, \hat{w}_2$.)

(c) Consider the 'simple' target function $f(\hat{\mathbf{x}}) = \hat{x}_1 + \hat{x}_2$. If you perform regression with the correlated inputs $\mathbf{x}$ and regularization constraint $w_1^2 + w_2^2 \le C$, what is the maximum amount of regularization you can use (minimum value of C) and still be able to implement the target?

(d) What happens to the minimum C as the correlation increases ($\epsilon \to 0$).

(e) Assuming that there is significant noise in the data, discuss your results in the context of bias and var.

The previous exercise illustrates that if the inputs are correlated, then the weights cannot be independently penalized, as they are in the standard form of weight-decay regularization. If the measured input variables are correlated, then one should transform them to a set that are uncorrelated (at least in-sample). That is the goal of input whitening.[2]

Remember that the data is centered, so the in-sample covariance matrix is

$$\Sigma = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}} = \frac{1}{N}\mathrm{X}^{\mathrm{T}}\mathrm{X}. \tag{9.1}$$

$\Sigma_{ij} = \mathsf{cov}(x_i, x_j)$ is the in-sample covariance of inputs i and j; $\Sigma_{ii} = \sigma_i^2$ is the variance of input i. Assume that Σ has full rank and let its matrix square root be $\Sigma^{\frac{1}{2}}$, which satisfies $\Sigma^{\frac{1}{2}}\Sigma^{\frac{1}{2}} = \Sigma$ (see Problem 9.3 for the computation of $\Sigma^{\frac{1}{2}}$). Consider the whitening transformation

$$\mathbf{z}_n = \Sigma^{-\frac{1}{2}}\mathbf{x}_n,$$

where $\Sigma^{-\frac{1}{2}}$ is the inverse of $\Sigma^{\frac{1}{2}}$. In matrix form, $\mathrm{Z} = \mathrm{X}\Sigma^{-\frac{1}{2}}$. Z is whitened if $\frac{1}{N}\mathrm{Z}^{\mathrm{T}}\mathrm{Z} = \mathrm{I}$. We verify that Z is whitened as follows:

$$\frac{1}{N}\mathrm{Z}^{\mathrm{T}}\mathrm{Z} = \Sigma^{-\frac{1}{2}}\left(\frac{1}{N}\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)\Sigma^{-\frac{1}{2}} = \Sigma^{-\frac{1}{2}}\Sigma\Sigma^{-\frac{1}{2}} = \left(\Sigma^{-\frac{1}{2}}\Sigma^{\frac{1}{2}}\right)\left(\Sigma^{\frac{1}{2}}\Sigma^{-\frac{1}{2}}\right) = \mathrm{I},$$

[2]The term whitening is inherited from signal processing where white noise refers to a signal whose frequency spectrum is uniform; this is indicative of a time series of independent noise realizations. The origin of the term white comes from white light which is a light signal whose amplitude distribution is uniform over all frequencies.

where we used $\Sigma = \Sigma^{\frac{1}{2}}\Sigma^{\frac{1}{2}}$ and $\Sigma^{\frac{1}{2}}\Sigma^{-\frac{1}{2}} = \Sigma^{-\frac{1}{2}}\Sigma^{\frac{1}{2}} = \mathrm{I}$. Thus, for the transformed inputs, every dimension has scale 1 and the dimensions are pairwise uncorrelated. Centering, normalizing and whitening are illustrated on a toy data set in Figure 9.1.

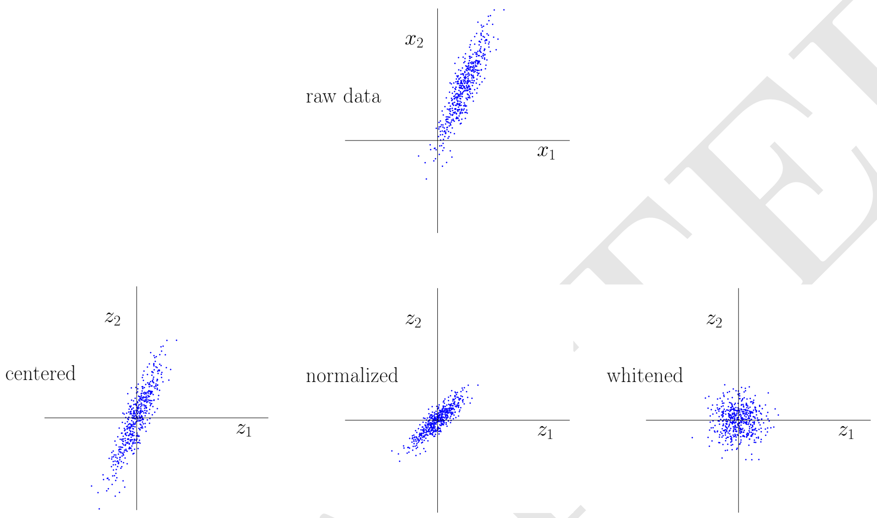


Figure 9.1: Illustration of centering, normalization and whitening.

It is important to emphasize that input preprocessing does not throw away any information because the transformation is invertible: the original inputs can always be recovered from the transformed inputs using $\bar{\mathbf{x}}$ and Σ.

WARNING! Transforming the data to a more convenient format has a hidden trap which easily leads to data snooping.

If you are using a test set to estimate your performance, make sure to determine any input transformation only using the training data. A simple rule: the test data should be kept locked away in its raw form until you are ready to test your final hypothesis. (See Example 5.3 for a concrete illustration of how data snooping can affect your estimate of the performance on your test set if input preprocessing in any way used the test set.) After you determine your transformation parameters from the training data, you should use these same parameters to transform your test data to evaluate your final hypothesis g.

9.2 Dimension Reduction and Feature Selection

The *curse of dimensionality* is a general observation that statistical tasks get exponentially harder as the dimensions increase. In learning from data, this manifests itself in many ways, the most immediate being computational. Simple algorithms, such as optimal (or near-optimal) k-means clustering, or determining the optimal linear separator, have a computational complexity which scales exponentially with dimensionality. Furthermore, a fixed number, N, of data points only sparsely populates a space whose volume is growing exponentially with d. So, simple rules like nearest neighbor get adversely affected because a test point's 'nearest' neighbor will likely be very far away and will not be a good representative point for predicting on the test point. The complexity of a hypothesis set, as could be measured by the VC dimension, will typically increase with d (recall that, for the simple linear perceptron, the VC dimension is $d + 1$), affecting the generalization from in-sample to out-of-sample. The bottom line is that more data are needed to learn in higher-dimensional input spaces.

Exercise 9.5

Consider a data set with two examples,

$$(\mathbf{x}_1^{\scriptscriptstyle\mathrm{T}} = [-1, a_1, \ldots, a_d],\ y_1 = +1); \qquad (\mathbf{x}_2^{\scriptscriptstyle\mathrm{T}} = [1, b_1, \ldots, b_d],\ y_2 = -1),$$

where a_i, b_i are independent random ± 1 variables. Let $\mathbf{x}_{\text{test}}^{\scriptscriptstyle\mathrm{T}} = [-1, -1, \ldots, -1]$. Assume that only the first component of $\mathbf{x}$ is relevant to f. However, the actual measured $\mathbf{x}$ has additional random components in the additional d dimensions. If the nearest neighbor rule is used, show, either mathematically or with an experiment, that the probability of classifying $\mathbf{x}_{\text{test}}$ correctly is $\frac{1}{2} + O(\frac{1}{\sqrt{d}})$ (d is the number of irrelevant dimensions).

What happens if there is a third data point $(\mathbf{x}_3^{\scriptscriptstyle\mathrm{T}} = [1, c_1, \ldots, c_d], y_3 = -1)$?

The exercise illustrates that as you have more and more spurious (random) dimensions, the learned final hypothesis becomes useless because it is dominated by the random fluctuations in these spurious dimensions. Ideally, we should remove all such spurious dimensions before proceeding to the learning. Equivalently, we should retain only the few informative features. For the digits data from Chapter 3, we were able to obtain good performance by extracting just two features from the raw 16×16-pixel input image (size and symmetry), a dimension reduction from 256 raw features to 2 informative ones.

The features $\mathbf{z}$ are simply a transformation of the input $\mathbf{x}$,

$$\mathbf{z} = \Phi(\mathbf{x}),$$

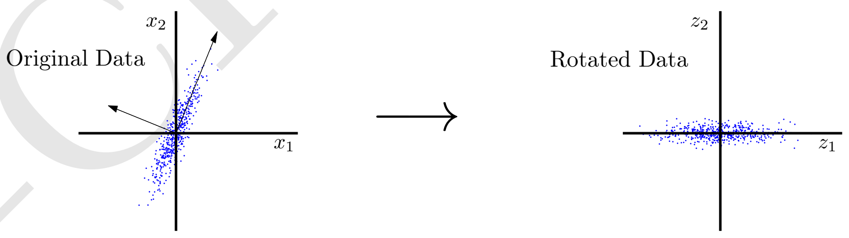
where the number of features is the dimension of $\mathbf{z}$. If the dimension of $\mathbf{z}$ is less than the dimension of $\mathbf{x}$, then we have accomplished dimension reduction. The ideal feature is the target function itself, $z = f(\mathbf{x})$, since if we had this feature, we are done. This suggests that quality feature selection may be as hard as the original learning problem of identifying f.

We have seen features and feature transforms many times before, for example, in the context of linear models and the non-linear feature transform in Chapter 3. In that context, the non-linear feature transform typically *increased* the dimension to handle the fact that the linear hypothesis was not expressive enough to fit the data. In that setting, increasing the dimension through the feature transform was attempting to improve E_{in}, and we did pay the price of poorer generalization. The reverse is also true. If we can *lower* the dimension without hurting E_{in} (as would be the case if we retained all the important information), then we will also improve generalization.

9.2.1 Principal Components Analysis (PCA)

Centering, scaling and whitening all attempt to correct for arbitrary choices that may have been made during data collection. Feature selection, such as PCA, is conceptually different. It attempts to get rid of redundancy or less informative dimensions to help, among other things, generalization. For example, the top right pixel in the digits data is almost always white, so it is a dimension that carries almost no information. Removing that dimension will not hurt the fitting, but will improve generalization.

PCA constructs a small number of *linear* features to summarize the input data. The idea is to rotate the axes (a linear transformation that defines a new coordinate system) so that the important dimensions in this new coordinate system become self evident and can be retained while the less important ones get discarded. Our toy data set can help crystallize the notion.

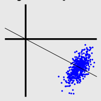
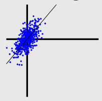


Once the data are rotated to the new 'natural' coordinate system, z_1 stands out as the important dimension of the transformed input. The second dimension, z_2, looks like a bunch of small fluctuations which we ought to ignore, in comparison to the apparently more informative and larger z_1. Ignoring the z_2 dimension amounts to setting it to zero (or just throwing away that coordinate), producing a 1-dimensional feature.

Exercise 9.6

Try to build some intuition for what the rotation is doing by using the illustrations in Figure 9.1 to qualitatively answer these questions.

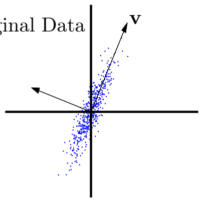
(a) If there is a large offset (or bias) in both measured variables, how will this affect the 'natural axes', the ones to which the data will be rotated? Should you perform input centering before doing PCA?

[*Hint: Consider*  *versus* ]

(b) If one dimension (say x_1) is inflated disproportionately (e.g., income is measured in dollars instead of thousands of dollars). How will this affect the 'natural axes', the ones to which the data should be rotated? Should you perform input normalization before doing PCA?

(c) If you do input whitening, what will the 'natural axes' for the inputs be? Should you perform input whitening before doing PCA?

What if the small fluctuations in the z_2 direction were the actual important information on which f depends, and the large variability in the z_1 dimension are random fluctuations? Though possible, this rarely happens in practice, and if it does happen, then your input is corrupted by large random noise and you are in trouble anyway. So, let's focus on the case where we have a chance and discuss how to find this optimal rotation.



Intuitively, the direction $\mathbf{v}$ captures the largest fluctuations in the data, which could be measured by variance. If we project the input $\mathbf{x}_n$ onto $\mathbf{v}$ to get $z_n = \mathbf{v}^{\mathrm{T}}\mathbf{x}_n$, then the variance of z is $\frac{1}{N}\sum_{n=1}^{N} z_n^2$ (remember $\mathbf{x}_n$ and hence z_n have zero mean).

$$\begin{aligned}\mathsf{var}[z] &= \frac{1}{N}\sum_{n=1}^{N} z_n^2 = \frac{1}{N}\sum_{n=1}^{N} \mathbf{v}^{\mathrm{T}}\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}}\mathbf{v} \\ &= \mathbf{v}^{\mathrm{T}}\left(\frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}}\right)\mathbf{v} \\ &= \mathbf{v}^{\mathrm{T}}\Sigma\mathbf{v}.\end{aligned}$$

To maximize $\mathsf{var}[z]$, we should pick $\mathbf{v}$ as the top eigenvector of Σ, the one with the largest eigenvalue. Before we get more formal, let us address an apparent conflict of interest. Whitening is a way to put your data into a spherically symmetric form so that all directions are 'equal'. This is recommended when you have no evidence to the contrary; you whiten the data because most learning algorithms treat every dimension equally (nearest neighbor, weight decay, etc.). PCA, on the other hand is highlighting specific directions which contain more variance. There is no use doing PCA after doing whitening, since every direction will be on an equal footing after whitening. You use

PCA precisely because the directions are not to be treated equally. PCA helps to identify and throw away the directions where the fluctuations are a result of small amounts of noise. After deciding which directions to throw away, you can now use whitening to put all the retained directions on an equal footing, if you wish.

Our visual intuition works well in 2-dimensions, but in higher dimension, when no single direction captures most of the fluctuation, we need a more principled approach, starting with a mathematical formulation of the task. We begin with the observation that a rotation of the data exactly corresponds to representing the data in a new (rotated) coordinate system.

Coordinate Systems A coordinate system is defined by an orthonormal basis, a set of mutually orthogonal unit vectors. The standard Euclidean coordinate system is defined by the Euclidean basis in d dimensions, $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_d$, where $\mathbf{u}_i$ is the ith standard basis vector which has a 1 in coordinate i and 0 for all other coordinates. The input vector $\mathbf{x}$ has the components $x_i = \mathbf{x}^{\mathrm{T}}\mathbf{u}_i$, and we can write

$$\mathbf{x} = \sum_{i=1}^{d} x_i \mathbf{u}_i = \sum_{i=1}^{d} (\mathbf{x}^{\mathrm{T}}\mathbf{u}_i)\mathbf{u}_i.$$

This can be done for any orthonormal basis $\mathbf{v}_1, \ldots, \mathbf{v}_d$,

$$\mathbf{x} = \sum_{i=1}^{d} z_i \mathbf{v}_i = \sum_{i=1}^{d} (\mathbf{x}^{\mathrm{T}}\mathbf{v}_i)\mathbf{v}_i,$$

where the *coordinates* in the basis $\mathbf{v}_1, \ldots, \mathbf{v}_d$ are $z_i = (\mathbf{x}^{\mathrm{T}}\mathbf{v}_i)$. The goal of PCA is to construct a more intuitive basis where some (hopefully the majority) of the coordinates are small and can be treated as small random fluctuations. These coordinates are going to be discarded, i.e., set to zero. The hope is that we have reduced the dimensionality of the problem while retaining most of the important information.

So, given the vector $\mathbf{x}$ (in the standard coordinate system) and some other coordinate system $\mathbf{v}_1, \ldots, \mathbf{v}_d$, we can define the transformed feature vector whose components are the coordinates $z_1, \ldots, z_d$ in this new coordinate system. Suppose that the first $k \leq d$ of these transformed coordinates are the informative ones, so we throw away the remaining coordinates to arrive at our *dimension-reduced* feature vector

$$\mathbf{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_k \end{bmatrix} = \begin{bmatrix} \mathbf{x}^{\mathrm{T}}\mathbf{v}_1 \\ \vdots \\ \mathbf{x}^{\mathrm{T}}\mathbf{v}_k \end{bmatrix} = \Phi(\mathbf{x}).$$

Exercise 9.7

(a) Show that $\mathbf{z}$ is a linear transformation of $\mathbf{x}$, $\mathbf{z} = \mathrm{V}^{\mathrm{T}}\mathbf{x}$. What are the dimensions of the matrix V and what are its columns?

(b) Show that the transformed data matrix is $\mathrm{Z} = \mathrm{XV}$.

(c) Show that $\sum_{i=1}^{d} z_i^2 = \sum_{i=1}^{d} x_i^2$ and hence that $\|\mathbf{z}\| \le \|\mathbf{x}\|$.

If we kept all the components $z_1, \ldots, z_d$, then we can reconstruct $\mathbf{x}$ via

$$\mathbf{x} = \sum_{i=1}^{d} z_i \mathbf{v}_i.$$

Using only the first k components, the best reconstruction of $\mathbf{x}$ is

$$\hat{\mathbf{x}} = \sum_{i=1}^{k} z_i \mathbf{v}_i.$$

We have lost that part of $\mathbf{x}$ represented by the trailing coordinates of $\mathbf{z}$. The magnitude of the part we lost is captured by the reconstruction error

$$\|\mathbf{x} - \hat{\mathbf{x}}\|^2 = \left\| \sum_{i=k+1}^{d} z_i \mathbf{v}_i \right\|^2 = \sum_{i=k+1}^{d} z_i^2$$

(because $\mathbf{v}_1, \ldots, \mathbf{v}_d$ are *orthonormal*). The new coordinate system is good if the sum of the reconstruction errors over the data points is small. That is, if

$$\sum_{n=1}^{N} \|\mathbf{x}_n - \hat{\mathbf{x}}_n\|^2$$

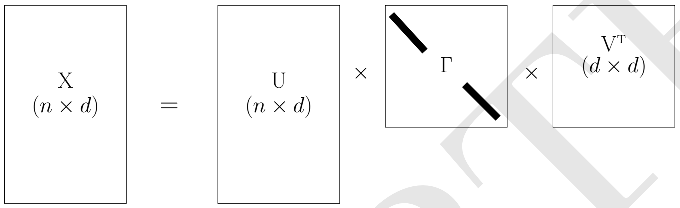
is small. If the $\mathbf{x}_n$ are reconstructed with small error from $\mathbf{z}_n$ (i.e. $\hat{\mathbf{x}}_n \approx \mathbf{x}_n$), then not much information was lost. PCA finds a coordinate system that *minimizes* this total reconstruction error. The trailing dimensions will have the least possible information, and so even after throwing away those trailing dimensions, we can still almost reconstruct the original data. PCA is optimal, which means that no other *linear* method can produce coordinates with a smaller reconstruction error. The first k basis vectors, $\mathbf{v}_1, \ldots, \mathbf{v}_k$, of this optimal coordinate basis are called the top-k principal directions.

So, how do we find this optimal coordinate basis $\mathbf{v}_1, \ldots, \mathbf{v}_d$ (of which we only need $\mathbf{v}_1, \ldots, \mathbf{v}_k$ to compute our dimensionally reduced feature)? The solution to this problem has been known since 1936 when the remarkable *singular value decomposition* (SVD) was invented. The SVD is such a useful tool for learning from data that time spent mastering it will pay dividends (additional background on the SVD is given in Appendix B.2).

The Singular Value Decomposition (SVD). Any matrix, for example, our data matrix X, has a very special representation as the product of three matrices. Assume that $\mathrm{X} \in \mathbb{R}^{N\times d}$ with $N \geq d$ (a similar decomposition holds for X^{T} if $N < d$). Then,

$$\mathrm{X} = \mathrm{U}\Gamma\mathrm{V}^{\mathsf{T}}$$

where $\mathrm{U} \in \mathbb{R}^{n\times d}$ has orthonormal columns, $\mathrm{V} \in \mathbb{R}^{d\times d}$ is an orthogonal matrix[3] and Γ is a non-negative diagonal matrix.[4] The diagonal elements $\gamma_i = \Gamma_{ii}$, where $\gamma_1 \geq \gamma_2 \geq \cdots \geq \gamma_d \geq 0$, are the *singular values* of X, ordered from largest to smallest. The number of non-zero singular values is the rank of X, which we will assume is d for simplicity. Pictorially,



The matrix U contains (as its columns) the *left singular vectors* of X, and similarly V contains (as its columns) the *right singular vectors* of X. Since U consists of orthonormal columns, $\mathrm{U}^{\mathsf{T}}\mathrm{U} = \mathrm{I}_d$. Similarly, $\mathrm{V}^{\mathsf{T}}\mathrm{V} = \mathrm{V}\mathrm{V}^{\mathsf{T}} = \mathrm{I}_d$. If X is square, then U will be square. Just as a square matrix maps an eigenvector to a multiple of itself, a more general non-square matrix maps a left (resp. right) singular vector to a multiple of the corresponding right (resp. left) singular vector, as verified by the following identities:

$$\mathrm{U}^{\mathsf{T}}\mathrm{X} = \Gamma\mathrm{V}^{\mathsf{T}}; \qquad \mathrm{X}\mathrm{V} = \mathrm{U}\Gamma.$$

It is convenient to have column representations of U and V in terms of the singular vectors, $\mathrm{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_d]$ and $\mathrm{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_d]$.

Exercise 9.8

Show $\mathrm{U}^{\mathsf{T}}\mathrm{X} = \Gamma\mathrm{V}^{\mathsf{T}}$ and $\mathrm{X}\mathrm{V} = \mathrm{U}\Gamma$, and hence $\mathrm{X}^{\mathsf{T}}\mathbf{u}_i = \gamma_i\mathbf{v}_i$ and $\mathrm{X}\mathbf{v}_i = \gamma_i\mathbf{u}_i$.

(The ith singular vectors and singular value $(\mathbf{u}_i, \mathbf{v}_i, \gamma_i)$ play a similar role to eigenvector-eigenvalue pairs.)

Computing the Principal Components via SVD. It is no coincidence that we used $\mathbf{v}_i$ for the right singular vectors of X and $\mathbf{v}_i$ in the mathematical formulation of the PCA task. The right singular vectors $\mathbf{v}_1, \ldots, \mathbf{v}_d$ are our optimal coordinate basis so that by ignoring the trailing components we incur the least reconstruction error.

[3]An orthogonal matrix is a square matrix with orthonormal columns, and therefore its inverse is the same as its transpose. So, $\mathrm{V}^{\mathsf{T}}\mathrm{V}=\mathrm{V}\mathrm{V}^{\mathsf{T}}=\mathrm{I}$.

[4]In the traditional linear algebra literature, Γ is typically denoted by Σ and its diagonal elements are the singular values $\sigma_1 \geq \cdots \geq \sigma_d$. We use Γ because we have reserved Σ for the covariance matrix of the input distribution and σ^2 for the noise variance.

Theorem 9.1 (Eckart and Young, 1936)**.** For any k, $\mathbf{v}_1, \ldots, \mathbf{v}_k$ (the top-k right singular vectors of the data matrix X) are a set of top-k principal component directions and the optimal reconstruction error is $\sum_{i=k+1}^{d} \gamma_i^2$.

The components of the dimensionally reduced feature vector are $z_i = \mathbf{x}^{\mathrm{T}}\mathbf{v}_i$ and the reconstructed vector is $\hat{\mathbf{x}} = \sum_{i=1}^{k} z_i \mathbf{v}_i$, which in matrix form is

$$\hat{\mathrm{X}} = \mathrm{X}\mathrm{V}_k\mathrm{V}_k^{\mathrm{T}}, \tag{9.2}$$

where $\mathrm{V}_k = [\mathbf{v}_1, \ldots, \mathbf{v}_k]$ is the matrix of top-k right singular vectors of X.

PCA Algorithm:

Inputs: The *centered* data matrix X and $k \geq 1$.

```
1: Compute the SVD of X: $[\mathrm{U}, \Gamma, \mathrm{V}] = \mathsf{svd}(\mathrm{X})$.
2: Let $\mathrm{V}_k = [\mathbf{v}_1, \ldots, \mathbf{v}_k]$ be the first $k$ columns of V.
3: The PCA-feature matrix and the reconstructed data are
       $\mathrm{Z} = \mathrm{X}\mathrm{V}_k, \qquad \hat{\mathrm{X}} = \mathrm{X}\mathrm{V}_k\mathrm{V}_k^{\mathrm{T}}.$
```

Note that PCA, along with the other input pre-processing tools (centering, rescaling, whitening) are all *unsupervised* - you do not need the y-values. The Eckart-Young theorem is quite remarkable and so fundamental in data analysis that it certainly warrants a proof.

Begin safe skip: You may skip the proof without compromising the logical sequence. A similar green box will tell you when to rejoin.

We will need some matrix algebra preliminaries which are useful general tools. Recall that the reconstructed data matrix $\hat{\mathrm{X}}$ is similar to the data matrix, having the reconstructed input vectors $\hat{\mathbf{x}}_n$ as its rows. The Frobenius norm $\|\mathrm{A}\|_F^2$ of a matrix $\mathrm{A} \in \mathbb{R}^{N\times d}$ is the analog of the Euclidean norm, but for matrices:

$$\|\mathrm{A}\|_F^2 \overset{\mathrm{def}}{=} \sum_{n=1}^{N}\sum_{i=1}^{d} \mathrm{A}_{ij}^2 = \sum_{n=1}^{N} \|\mathrm{row}_n(\mathrm{A})\|^2 = \sum_{i=1}^{d} \|\mathrm{column}_i(\mathrm{A})\|^2.$$

The reconstruction error is exactly the Frobenius norm of the matrix difference between the original and reconstructed data matrices, $\|\mathrm{X} - \hat{\mathrm{X}}\|_F^2$.

Exercise 9.9

Consider an arbitrary matrix A, and any matrices U, V with orthonormal columns ($\mathrm{U}^{\mathrm{T}}\mathrm{U} = \mathrm{I}$ and $\mathrm{V}^{\mathrm{T}}\mathrm{V} = \mathrm{I}$).

(a) Show that $\|\mathrm{A}\|_F^2 = \mathrm{trace}(\mathrm{A}\mathrm{A}^{\mathrm{T}}) = \mathrm{trace}(\mathrm{A}^{\mathrm{T}}\mathrm{A})$.

(b) Show that $\|\mathrm{U}\mathrm{A}\mathrm{V}^{\mathrm{T}}\|_F^2 = \|\mathrm{A}\|_F^2$ (assume all matrix products exist). *[Hint: Use part (a).]*

Proof of the Eckart-Young Theorem. The preceding discussion was for general A. We now set A to be any orthonormal basis A, with columns $\mathbf{a}_1, \ldots, \mathbf{a}_d$,

$$\mathrm{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_d].$$

Since V is an orthonormal basis, we can write

$$\mathrm{A} = \mathrm{V}\boldsymbol{\Psi},$$

where $\boldsymbol{\Psi} = [\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_d]$. Since

$$\mathrm{I} = \mathrm{A}^{\mathrm{T}}\mathrm{A} = \boldsymbol{\Psi}^{\mathrm{T}}\mathrm{V}^{\mathrm{T}}\mathrm{V}\boldsymbol{\Psi} = \boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{\Psi},$$

we see that $\boldsymbol{\Psi}$ is orthogonal. Suppose (without loss of generality) that we will use the first k basis vectors of A for reconstruction. So, define

$$\mathrm{A}_k = [\mathbf{a}_1, \ldots, \mathbf{a}_k] = \mathrm{V}\boldsymbol{\Psi}_k,$$

where $\Psi_k = [\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_k]$. We use A_k to approximate X using the reconstruction $\hat{\mathrm{X}} = \mathrm{X}\mathrm{A}_k\mathrm{A}_k^{\mathrm{T}}$ from (9.2). Then,

$$\begin{aligned}
\|\mathrm{X} - \hat{\mathrm{X}}\|_F^2 &= \|\mathrm{X} - \mathrm{X}\mathrm{A}_k\mathrm{A}_k^{\mathrm{T}}\|_F^2 \\
&= \|\mathrm{U}\Gamma\mathrm{V}^{\mathrm{T}} - \mathrm{U}\Gamma\mathrm{V}^{\mathrm{T}}\mathrm{V}\boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}}\mathrm{V}^{\mathrm{T}}\|_F^2 \\
&= \|\mathrm{U}(\Gamma - \Gamma\boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}})\mathrm{V}^{\mathrm{T}}\|_F^2 \\
&= \|\Gamma(\mathrm{I} - \boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}})\|_F^2,
\end{aligned}$$

where we have used $\mathrm{V}^{\mathrm{T}}\mathrm{V} = \mathrm{I}_d$ and Exercise 9.9(b). By Exercise 9.9(a),

$$\|\Gamma(\mathrm{I} - \boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}})\|_F^2 = \mathrm{trace}(\Gamma(\mathrm{I} - \boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}})^2\Gamma).$$

Now, using the linearity and cyclic properties of the trace and the fact that $\mathrm{I} - \boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}}$ is a projection,

$$\begin{aligned}
\mathrm{trace}\left(\Gamma(\mathrm{I} - \boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}})^2\Gamma\right) &= \mathrm{trace}\left(\Gamma(\mathrm{I} - \boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}})\Gamma\right) \\
&= \mathrm{trace}(\Gamma^2) - \mathrm{trace}\left(\Gamma\boldsymbol{\Psi}_k\boldsymbol{\Psi}_k^{\mathrm{T}}\Gamma\right) \\
&= \mathrm{trace}(\Gamma^2) - \mathrm{trace}\left(\boldsymbol{\Psi}_k^{\mathrm{T}}\Gamma^2\boldsymbol{\Psi}_k\right).
\end{aligned}$$

The first term is independent of $\boldsymbol{\Psi}_k$, so we must maximize the second term. Since $\boldsymbol{\Psi}_k$ has k orthonormal columns, $\|\boldsymbol{\Psi}_k\|_F^2 = k$ (because the Frobenius norm is the sum of squared column norms). Let the rows of $\boldsymbol{\Psi}_k$ be $\mathbf{q}_1^{\mathrm{T}}, \ldots, \mathbf{q}_d^{\mathrm{T}}$. Then $0 \le \|\mathbf{q}_i\|^2 \le 1$ ($\boldsymbol{\Psi}$ has orthonormal rows and $\mathbf{q}_i$ are truncated rows of $\boldsymbol{\Psi}$), and $\sum_{i=1}^{d} \|\mathbf{q}_i\|^2 = k$ (because the Frobenius norm is the sum of squared row-norms). We also have that

$$\boldsymbol{\Psi}_k^{\mathrm{T}}\Gamma^2\boldsymbol{\Psi}_k = [\mathbf{q}_1, \ldots, \mathbf{q}_d] \begin{bmatrix} \gamma_1^2 & & \\ & \ddots & \\ & & \gamma_d^2 \end{bmatrix} \begin{bmatrix} \mathbf{q}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{q}_d \end{bmatrix} = \sum_{i=1}^{d} \gamma_i^2 \mathbf{q}_i\mathbf{q}_i^{\mathrm{T}}.$$

Taking the trace and using linearity of the trace gives

$$\text{trace}(\mathbf{\Psi}_k^{\text{T}}\Gamma^2\mathbf{\Psi}_k) = \sum_{i=1}^{d} \gamma_i^2 \|\mathbf{q}_i\|^2.$$

To maximize $\text{trace}(\mathbf{\Psi}_k^{\text{T}}\Gamma^2\mathbf{\Psi}_k)$, the best possible way to spend our budget of k for $\sum_i \|\mathbf{q}_i\|^2$ is to put as much as possible into $\|\mathbf{q}_1\|$ and then $\|\mathbf{q}_2\|$ and so on, because $\gamma_1 \geq \gamma_2 \geq \cdots$. This will result in the $\|\mathbf{q}_1\|^2 = \cdots = \|\mathbf{q}_k\|^2 = 1$ and all the remaining $\mathbf{q}_i = \mathbf{0}$. Thus,

$$\text{trace}(\mathbf{\Psi}_k^{\text{T}}\Gamma^2\mathbf{\Psi}_k) \leq \sum_{i=1}^{k} \gamma_i^2.$$

We conclude that

$$\begin{aligned}
\|\text{X} - \hat{\text{X}}\|_F^2 &= \text{trace}(\Gamma^2) - \text{trace}\left(\mathbf{\Psi}_k^{\text{T}}\Gamma^2\mathbf{\Psi}_k\right) \\
&= \sum_{i=1}^{d} \gamma_i^2 - \text{trace}\left(\mathbf{\Psi}_k^{\text{T}}\Gamma^2\mathbf{\Psi}_k\right) \\
&\geq \sum_{i=1}^{d} \gamma_i^2 - \sum_{i=1}^{k} \gamma_i^2 \\
&= \sum_{i=k+1}^{d} \gamma_i^2.
\end{aligned}$$

If $\text{A} = \text{V}$ so that $\mathbf{\Psi} = \text{I}$, then indeed $\|\mathbf{q}_1\|^2 = \cdots = \|\mathbf{q}_k\|^2 = 1$ and we attain equality in the bound above, showing that the top-k right singular vectors of X do indeed give an optimal basis, which concludes the proof. ■

The proof of the theorem also shows that the optimal reconstruction error is the sum of squares of the trailing singular values of X, $\|\text{X} - \hat{\text{X}}\|_F^2 = \sum_{i=k+1}^{d} \gamma_i^2$.

End safe skip: Those who skipped the proof are now rejoining us for some examples of PCA in action.

Example 9.2. It is instructive to see PCA in action. The digits training data matrix has 7291 (16×16)-images $(d = 256)$, so $\text{X} \in \mathbb{R}^{7291\times 256}$. Let $\bar{\mathbf{x}}$ be the average image. First center the data, setting $\mathbf{x}_n \leftarrow \mathbf{x}_n - \bar{\mathbf{x}}$, to get a centered data matrix X.

Now, let's compute the SVD of this centered data matrix, $\text{X} = \text{U}\Gamma\text{V}^{\text{T}}$. To do so we may use a built in SVD package that is pretty much standard on any numerical platform. It is also possible to only compute the top-k singular vectors in most SVD packages to obtain $\text{U}_k, \Gamma_k, \text{V}_k$, where U_k and V_k

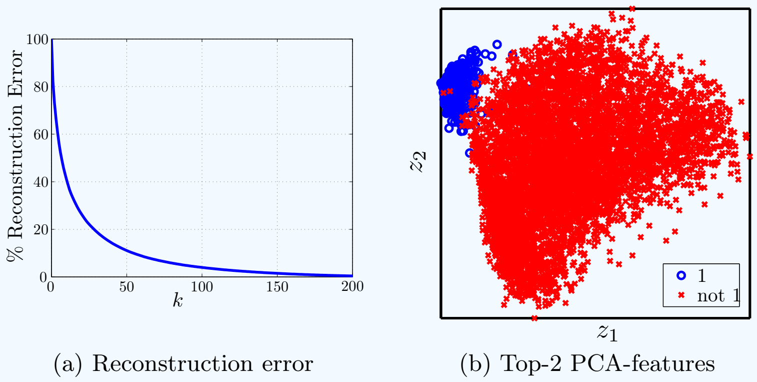


Figure 9.2: PCA on the digits data. (a) Shows how the reconstruction error depends on the number of components k; about 150 features suffice to reconstruct the data almost perfectly. If all principal components are equally important, the reconstruction error would decrease linearly with k, which is not the case here. (b) Shows the two features obtained using the top two principal components. These features look as good as our hand-constructed features of symmetry and intensity from Example 3.5 in Chapter 3.

contain only the top-k singular vectors, and Γ_k the top-k singular values. The reconstructed matrix is

$$\hat{\mathrm{X}} = \mathrm{X}\mathrm{V}_k\mathrm{V}_k^{\mathrm{T}} = \mathrm{U}_k\Gamma_k\mathrm{V}_k^{\mathrm{T}}.$$

By the Eckart-Young theorem, $\hat{\mathrm{X}}$ is the best rank-k approximation to X. Let's look at how the reconstruction error depends on k, the number of features used. To do this, we plot the reconstruction error using k principle components as a percentage of the reconstruction error with zero components. With zero components, the reconstruction error is just $\|\mathrm{X}\|_F^2$. The result is shown in Figure 9.2(a). As can be seen, with just 50 components, the reconstruction error is about 10%. This is a rule of thumb to determine how many components to use: choose k to obtain a reconstruction error of less than 10%. This amounts to treating the bottom 10% of the fluctuations in the data as noise.

Let's reduce the dimensionality to 2 by projecting onto the top two principal components in V_2. The resulting features are shown in Figure 9.2(b). These features can be compared to the features obtained using intensity and symmetry in Example 3.5 on page 106. The features appear to be quite good, and suitable for solving this learning problem. Unlike the size and intensity features which we used in Example 3.5, the biggest advantage of these PCA features is that their construction is fully automated – you don't need to know anything about the digit recognition problem to obtain them. This is also their biggest disadvantage – the features are provably good at reconstructing the input data, but there is no guarantee that they will be useful for solving the learning problem. In practice, a little thought about constructing features, together with such automated methods to then reduce the dimensionality, usually works best.

Finally, suppose you use the feature vector $\mathbf{z}$ in Figure 9.2(b) to build a classifier $\tilde{g}(\mathbf{z})$ to distinguish between the digit 1 and all the other digits. The final hypothesis to be applied to a test input $\mathbf{x}_{\text{test}}$ is

$$g(\mathbf{x}_{\text{test}}) = \tilde{g}(\mathrm{V}_2^{\mathrm{T}}(\mathbf{x}_{\text{test}} - \bar{\mathbf{x}})),$$

where $\tilde{g}$, V_2 and $\bar{\mathbf{x}}$ were constructed from the data: V_2 and $\bar{\mathbf{x}}$ from the data inputs, and $\tilde{g}$ from the targets and PCA-reduced inputs $(\mathrm{Z}, \mathbf{y})$ □

We end this section by justifying the "maximize the variance" intuition that began our discussion of PCA. The next exercise shows that the principal components do indeed represent the high variance directions in the data.

Exercise 9.10

Assume that the data matrix X is centered, and define the covariance matrix $\Sigma = \frac{1}{N}\mathrm{X}^{\mathrm{T}}\mathrm{X}$. Assume all the singular values of X are distinct. (What does this mean for the eigenvalues of Σ?) For a potential principal direction $\mathbf{v}$, we defined $z_n = \mathbf{x}_n^{\mathrm{T}}\mathbf{v}$ and showed that $\text{var}(z_1, \ldots, z_N) = \mathbf{v}^{\mathrm{T}}\Sigma\mathbf{v}$.

(a) Show that the direction which results in the highest variance is $\mathbf{v}_1$, the top right singular vector of X.

(b) Show that we obtain the top-k principal directions, $\mathbf{v}_1, \ldots, \mathbf{v}_k$, by selecting k directions sequentially, each time obtaining the direction with highest variance that is orthogonal to all previously selected directions.

This shows that the top-k principal directions are the directions of highest variance.

(c) If you don't the data matrix X, but only know the covariance matrix Σ, can you obtain the principal directions? If so, how?

9.2.2 Nonlinear Dimension Reduction

Figure 9.3 illustrates one thing that can go wrong with PCA. You can see that the data in Figure 9.3(a) approximately lie on a one-dimensional surface (a curve). However, when we try to reconstruct the data using top-1 PCA, the result is a disaster, shown in Figure 9.3(b). This is because, even though the data lie on a curve, that curve is not linear. PCA can only construct linear features. If the data do not live on a lower dimensional 'linear manifold', then PCA will not work. If you are going to use PCA, here is a checklist that will help you determine whether PCA will work.

1. Do you expect the data to have *linear* structure, for example does the data lie in a linear subspace of lower dimension?

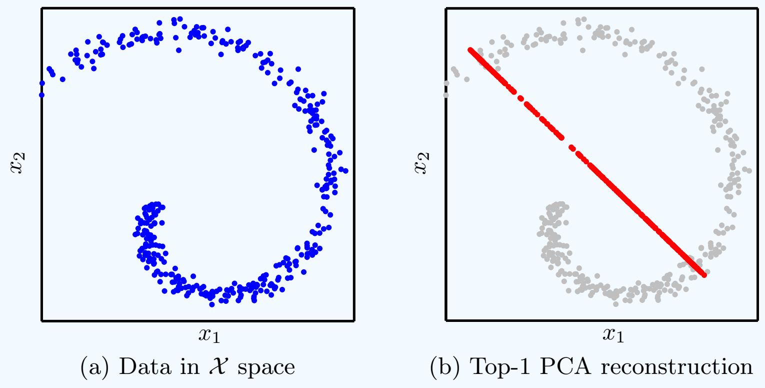


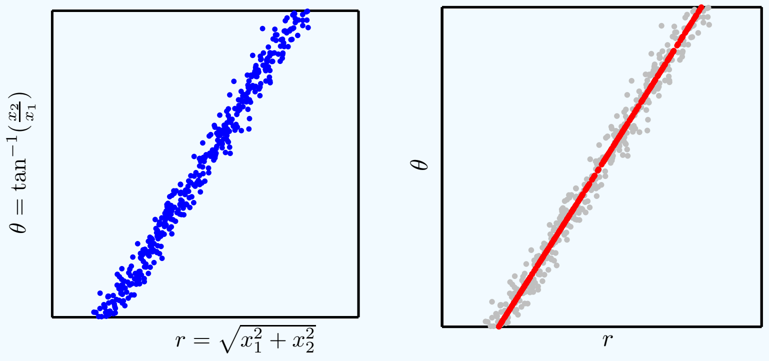
Figure 9.3: (a) The data in $\mathcal{X}$ space does not 'live' in a lower dimensional *linear* manifold. (b) The reconstructed data using top-1 PCA data must lie on a line and therefore cannot accurately represent the original data.

2. Do the bottom principal components contain primarily small random fluctuations that correspond to noise and should be thrown away? The fact that they are small can be determined by looking at the reconstruction error. The fact that they are noise is not much more than a guess.
3. Does the target function f depend primarily on the top principal components, or are the small fluctuations in the bottom principal components key in determining the value of f? If the latter, then PCA will not help the machine learning task. In practice, it is difficult to determine whether this is true (without snooping ☺). A validation method can help determine whether to use PCA-dimension-reduction or not. Usually, throwing away the lowest principal components does not throw away significant information related to the target function, and what little it does throw away is made up for in the reduced generalization error bar because of the lower dimension.

Clearly, PCA will not work for our data in Figure 9.3. However, we are not dead yet. We have an ace up our sleeve, namely the all-powerful nonlinear transform. Looking at the data in Figure 9.3 suggests that the angular coordinate is important. So, lets consider a transform to the nonlinear feature space defined by polar coordinates.

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \xrightarrow{\Phi} \begin{bmatrix} r \\ \theta \end{bmatrix} = \begin{bmatrix} \sqrt{x_1^2 + x_2^2} \\ \tan^{-1}(\frac{x_2}{x_1}) \end{bmatrix}$$

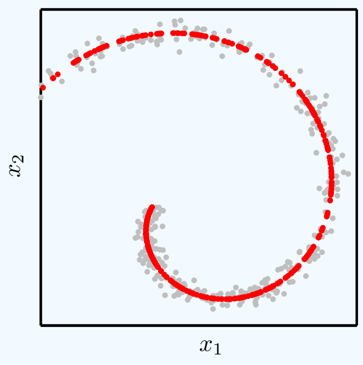
The data using polar-coordinates is shown in Figure 9.4(a). In this space, the data clearly lie on a linear subspace, appropriate for PCA. The top-1 PCA reconstructed data (in the nonlinear feature space) is shown in Figure 9.4(b).



(a) Transformed data in $\mathcal{Z}$ space (b) Reconstruction in $\mathcal{Z}$ space

Figure 9.4: PCA in a nonlinear feature space. (a) The transformed data in the $\mathcal{Z}$ space are approximately on a linear manifold. (b) Shows nonlinear reconstruction of the data in the nonlinear feature space.

We can obtain the reconstructed data in the original $\mathcal{X}$ space by transforming the red reconstructed points in Figure 9.4(b) back to $\mathcal{X}$ space, as shown below.



Exercise 9.11

Using the feature transform $\Phi : \left[\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right] \mapsto \left[\begin{smallmatrix} x_1 \\ x_2 \\ x_1+x_2 \end{smallmatrix}\right]$, you have run top-1 PCA on your data $\mathbf{z}_1, \ldots, \mathbf{z}_n$ in $\mathcal{Z}$ space to obtain $\mathrm{V}_1 = \left[\begin{smallmatrix} 0 \\ 0 \\ 1 \end{smallmatrix}\right]$ and $\bar{\mathbf{z}} = \mathbf{0}$.

For the test point $\mathbf{x} = \left[\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right]$, compute $\mathbf{z}, \hat{\mathbf{z}}, \hat{\mathbf{x}}$.

($\mathbf{z}$ is the test point in $\mathcal{Z}$ space; $\hat{\mathbf{z}}$ is the reconstructed test point in $\mathcal{Z}$ space using top-1 PCA; $\hat{\mathbf{x}}$ is the reconstructed test point in $\mathcal{X}$ space.)

Exercise 9.11 illustrates that you may not always be able to obtain the reconstructed data in your original $\mathcal{X}$ space. For our spiral example, we can obtain the reconstructed data in the $\mathcal{X}$ space because the polar feature transform is *invertible*; invertibility of Φ is essential to reconstruction in $\mathcal{X}$ space. In general, you may want to use a feature transform that is not invertible, for example the 2nd-order polynomial transform. In this case, you will not be able to reconstruct your data in the $\mathcal{X}$ space, but that need not prevent you from using PCA with the nonlinear feature transform, if your goal is prediction. You can transform to your $\mathcal{Z}$ space, run PCA in the $\mathcal{Z}$ space, discard the bottom principal components (dimension reduction), and run your learning algorithm using the top principal components. Finally, to classify a new test point, you first transform it, and then classify in the $\mathcal{Z}$ space. The entire work-flow is summarized in the following algorithm.

```
PCA with Nonlinear Feature Transform:
Inputs: The data $\mathrm{X}, \mathbf{y}$; $k \ge 1$; and, transform $\Phi$.
  1: Transform the data: $\mathbf{z}_n = \Phi(\mathbf{x}_n)$.
  2: Center the data: $\mathbf{z}_n \leftarrow \mathbf{z}_n - \bar{\mathbf{z}}$ where $\bar{\mathbf{z}} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{z}_n$.
  3: Obtain the centered data matrix Z whose rows are $\mathbf{z}_n$.
  4: Compute the SVD of Z: $[\mathrm{U}, \Gamma, \mathrm{V}] = \mathsf{svd}(\mathrm{Z})$.
  5: Let $\mathrm{V}_k = [\mathbf{v}_1, \dots, \mathbf{v}_k]$ be the first $k$ columns of V.
  6: Construct the top-k PCA-feature matrix $\mathrm{Z}_k = \mathrm{Z}\mathrm{V}_k$.
  7: Use the data $(\mathrm{Z}_k, \mathbf{y})$ to learn a final hypothesis $\tilde{g}$.
  8: The final hypothesis is
         $g(\mathbf{x}) = \tilde{g}(\mathrm{V}_k^{\mathrm{T}}(\Phi(\mathbf{x}) - \bar{\mathbf{z}}))$
```

There are other approaches to nonlinear dimension reduction. Kernel-PCA is a way to combine PCA with the nonlinear feature transform without visiting the $\mathcal{Z}$ space. Kernel-PCA uses a kernel in the $\mathcal{X}$ space in much the same way that kernels were used to combine the maximum-margin linear separator with the nonlinear transform to get the Kernel-Support Vector Machine. Two other popular approaches are the Neural-Network auto-encoder (which we discussed in the context of Deep Learning with Neural Networks in Section 7.6) and nonlinear principal curves and surfaces which are nonlinear analogues to the PCA-generated linear principal surfaces. With respect to reconstructing the data onto lower dimensional manifolds, there are also nonparametric techniques like the Laplacian Eigenmap or the Locally Linear Embedding (LLE). The problems explore some of these techniques in greater depth.

9.3 Hints And Invariances

Hints are tidbits of information about the target function that you know ahead of time (before you look at any data). You know these tidbits because you know something about the learning problem. Why do we need hints? If we had an unlimited supply of data (input-output examples), we wouldn't need hints because the data contains all the information we need to learn.[5] In reality, though, the data is finite, and so not all properties about f may be represented in the data. Common hints are invariances, monotonicities and symmetries. For example, rotational invariance applies in most vision problems.

Exercise 9.12

Consider the following three hypothesis sets,

$$\begin{aligned}\mathcal{H}_+ &= \{h|h(x) = \text{sign}(wx + w_0);\ w \geq 0\},\\ \mathcal{H}_- &= \{h|h(x) = \text{sign}(wx + w_0);\ w < 0\},\end{aligned}$$

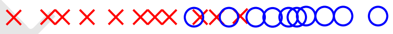
and $\mathcal{H} = \mathcal{H}_+ \cup \mathcal{H}_-$. The task is to perform credit approval based on the income x. The weights w and w_0 must be learned from data.

(a) What are the VC dimensions of $\mathcal{H}$ and $\mathcal{H}_\pm$?

(b) Which model will you pick (before looking at data)? Explain why.

The exercise illustrates *several* important points about learning from data, and we are going to beat these points into the ground. First, the choice between $\mathcal{H}$ and $\mathcal{H}_\pm$ brings the issue of generalization to light. Do you have enough data to use the more complex model, or do you have to use a simpler model to ensure that $E_{\text{in}} \approx E_{\text{out}}$? When learning from data, this is the first order of business. If you cannot expect $E_{\text{in}} \approx E_{\text{out}}$, you are doomed from step 1. Suppose you need to choose one of the simpler models to get good generalization. $\mathcal{H}_+$ and $\mathcal{H}_-$ are of equal 'simplicity'. As far as generalization is concerned, both models are equivalent. Let's play through both choices.

Suppose you choose $\mathcal{H}_-$. The nature of credit approval is that if you have more money, you are less likely to default. So, when you look at the data, it will look something like this (blue circles are $+1$)



No matter how you try to fit this data using hypotheses in $\mathcal{H}_-$, the in-sample error will be approximately $\frac{1}{2}$. Nevertheless, you will output one such final hypothesis g_-. You will have succeeded in that $E_{\text{in}}(g_-) \approx E_{\text{out}}(g_-)$. But you will have failed in that you will tell your client to expect about 50% out-of-sample error, and he will laugh at you.

Suppose, instead, you choose $\mathcal{H}_+$. Now you will be able to select a hypothesis g_+ with $E_{\text{in}}(g_+) \approx 0$; and, because of good generalization, you will be able to tell the client that you expect near perfect out-of-sample accuracy.

As in this credit example, you often have auxiliary knowledge about the problem. In our example, we have reason to pick $\mathcal{H}_+$ over $\mathcal{H}_-$, and by do-

[5]This is true from the informational point of view. However, from the computational point of view, a hint can still help. For example, knowing that the target is linear will considerably speed up your search by focusing it on linear models.

ing so, we are using the *hint* that one's credit status, by its very nature, is monotonically increasing in income.

Moral. Don't throw any old $\mathcal{H}$ at the problem just because your $\mathcal{H}$ is small and will give good generalization. More often than not, you will learn that you failed. Throw the *right* $\mathcal{H}$ at the problem, and if, *unfortunately*, you fail you will know it.

Exercise 9.13

For the problem in Exercise 9.12, why not try a hybrid procedure:

Start with $\mathcal{H}_+$. If $\mathcal{H}_+$ gives low E_{in}, stop; if not, use $\mathcal{H}_-$.

If your hybrid strategy stopped at $\mathcal{H}_+$, can you use the VC bound for $\mathcal{H}_+$? Can you use the VC-bound for $\mathcal{H}$?

(Problem 9.20 develops a VC-type analysis of such hybrid strategies within a framework called Structural Risk Minimization (SRM).)

Hints are pieces of prior information about the learning problem. By prior, we mean prior to looking at the data. Here are some common examples of hints.

Symmetry or Anti-symmetry hints: $f(\mathbf{x}) = f(-\mathbf{x})$ or $f(\mathbf{x}) = -f(-\mathbf{x})$. For example, in a financial application, if a historical pattern $\mathbf{x}$ means you should buy the stock, then the reverse pattern $-\mathbf{x}$ often means you should sell.

Rotational invariance: $f(\mathbf{x})$ depends only on $\|\mathbf{x}\|$.

General Invariance hints: For some transformation $\mathcal{T}$, $f(\mathbf{x}) = f(\mathcal{T}\mathbf{x})$. Invariance to scale, shift and rotation of an image are common in vision applications.

Monotonicity hints: $f(\mathbf{x} + \Delta\mathbf{x}) \geq f(\mathbf{x})$ if $\Delta\mathbf{x} \geq \mathbf{0}$. Sometimes you may only have monotonicity in some of the variables. For example, credit approval should be a monotonic function of income, but perhaps not monotonic in age.

Convexity hint: $f(\eta\mathbf{x} + (1-\eta)\mathbf{x}') \leq \eta f(\mathbf{x}) + (1-\eta)f(\mathbf{x}')$ for $0 \leq \eta \leq 1$.

Perturbation hint: f is close to a known function f', so $f = f' + \delta f$, where δf is small (sometimes called a catalyst hint).

One way to use a hint is to constrain the hypothesis set so that all hypotheses satisfy the hint: start with a hypothesis set $\tilde{\mathcal{H}}$ and throw out all hypotheses which do not satisfy the constraint to obtain $\mathcal{H}$. This will directly lower the size of the hypothesis set, improving your generalization ability. The next exercise illustrates the mechanics of this approach.

Exercise 9.14

Consider the linear model with the quadratic transform in two dimensions. So, the hypothesis set contains functions of the form

$$h(\mathbf{x}) = \text{sign}(w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2 + w_5x_1x_2).$$

Determine constraints on $\mathbf{w}$ so that:

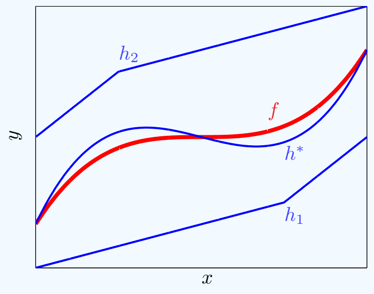
(a) $h(\mathbf{x})$ is monotonically increasing in $\mathbf{x}$.

(b) $h(\mathbf{x})$ is invariant under an arbitrary rotation of $\mathbf{x}$.

(c) The positive set $\{\mathbf{x} : h(\mathbf{x}) = +1\}$ is convex.

Since the hint is a known property of the target, throwing out hypotheses that do not match the hint should not diminish your ability to fit the target function (or the data). Unfortunately, this need not be the case if the target function was not in your hypothesis set to start with. The simple example below explains why. The figure shows a hypothesis set with three hypotheses (blue) and the target function (red) which is *known* to be monotonically increasing.

$$\mathcal{H} = \{h^*, h_1, h_2\}$$



Though f is monotonic, the best approximation to f in $\mathcal{H}$ is h^*, and h^* is not monotonic. So, if you remove all non-monotonic hypotheses from $\mathcal{H}$, all the remaining hypotheses are bad and you will underfit heavily.

The best approximation to f in $\mathcal{H}$ may not satisfy your hint.

When you outright enforce the hint, you certainly ensure that the final hypothesis has the property that f is known to have. But you also may exaggerate deficiencies that were in the hypothesis set to start with. Strictly enforcing a hint is like a hard constraint, and we have seen an example of this before when we studied regularization and the hard-order constraint in Chapter 4: we 'softened' the constraint, thereby giving the algorithm some flexibility to fit the data while still pointing the learning toward simpler hypotheses. A similar situation holds for hints. It is often better to inform the learning about the hint but allow the flexibility to violate the hint a little if that is warranted for fitting the data - that is, to softly enforce the hint. A popular and general way to softly enforce a hint is using *virtual examples.*

9.3.1 Virtual Examples

Hints are useful because they convey additional information that may not be represented in the finite data set. Virtual examples augment the data set so that the hint becomes represented in the data. The learning can then just focus on this augmented data without needing any new machinery to deal explicitly with the hint. The learning algorithm, by trying to fit all the data (the real and virtual examples) can trade-off satisfying the known hint about f (by attempting to fit the virtual examples) with approximating f (by attempting to fit the real examples). In contrast, when you explicitly constrain the model, you cannot take advantage of this trade-off; you simply do the best you can to fit the data while obeying the hint. If by slightly violating the hint you can get a hypothesis that is a supreme fit to the data, it's just too bad, because you don't have access to such a hypothesis. Let's set aside the ideological goal that known properties of f must be preserved and focus on the error (both in and out-of-sample); that is all that matters.

The general approach to constructing virtual examples is to imagine numerically testing if a hypothesis h satisfies the known hint. This will suggest how to build an error function that would equal zero if the hint is satisfied and non-zero values will quantify the degree to which the hint is not satisfied. Let's see how to do this by concrete example.

Invariance Hint. Let's begin with a simple example, the symmetry hint which says that the target function is symmetric: $f(\mathbf{x}) = f(-\mathbf{x})$. How would we test that a particular h has symmetry? Generate a set of arbitrary virtual pairs $\{\mathbf{v}_m, -\mathbf{v}_m\}$, $m = 1, \ldots, M$ ($\mathbf{v}$ for virtual), and test if $h(\mathbf{v}_m) = h(\mathbf{v}'_m)$ for every virtual pair. Thus, we can define the hint error

$$E_{\text{hint}}(h) = \frac{1}{M}\sum_{m=1}^{M}(h(\mathbf{v}_m) - h(-\mathbf{v}_m))^2.$$

Certainly, for the target function, $E_{\text{hint}}(f) = 0$. By allowing for non-zero E_{hint}, we can allow the possibility of slightly violating symmetry, provided that it gives us a much better fit of the data. So, define an augmented error

$$E_{\text{aug}}(h) = E_{\text{in}}(h) + \lambda E_{\text{hint}}(h) \tag{9.3}$$

to quantify the trade-off between enforcing the hint and fitting the data. By minimizing E_{aug}, we are not explicitly enforcing the hint, but encouraging it. The parameter λ controls how much we emphasize the hint over the fit. This looks suspiciously like regularization, and we will say more on this later.

How should one choose λ? The hint conveys information to help the learning. However, overemphasizing the hint by picking too large a value for λ can lead to underfitting just as over-regularizing can. If λ is too large, it amounts to strictly enforcing the hint, and we already saw how that can hurt. The answer: use validation to select λ. (See Chapter 4 for details on validation.)

How should one select virtual examples $\{\mathbf{v}_m, -\mathbf{v}_m\}$ to compute the hint error? As a start, use the real examples themselves. A virtual example for the symmetry hint would be $\{\mathbf{x}_n, -\mathbf{x}_n\}$, for $n = 1, \ldots, N$.

Exercise 9.15

Why might it be a good idea to use the data points to compute the hint error? When might it be better to use more hint examples?

[Hint ☺: Do you care if a hypothesis violates the hint in a part of the input space that has very low probability?]

A general invariance partitions the input space into disjoint regions,

$$\mathcal{X} = \cup_\alpha \mathcal{X}_\alpha.$$

Within any partition, the target function is constant. That is, if $\mathbf{x}, \mathbf{x}' \in \mathcal{X}_\alpha$ then $f(\mathbf{x}) = f(\mathbf{x}')$. A set of transformations $\mathcal{T}$ can be used to implicitly define the regions in the partition: $\mathbf{x}$ and $\mathbf{x}'$ are in the same partition $\mathcal{X}_\alpha$ if and only if $\mathbf{x}'$ can be obtained from $\mathbf{x}$ by applying a composition of transformations from $\mathcal{T}$. As an example, suppose $f(\mathbf{x})$ is invariant under arbitrary rotation of the input $\mathbf{x}$ in the plane. Then the invariant set $\mathcal{X}_r$ is the circle of radius r centered on the origin. For any two points on the same circle, the target function evaluates to the same value. To generate the virtual examples, take each data point $\mathbf{x}_n$ and determine (for example randomly) a virtual example $\mathbf{x}'_n$ that is in the same invariant set as $\mathbf{x}_n$. The hint error would then be

$$E_{\text{hint}}(h) = \frac{1}{N}\sum_{n=1}^{N}(h(\mathbf{x}_n) - h(\mathbf{x}'_n))^2.$$

Notice that we can always compute the hint error, even though we do not know the target function on the virtual examples.

Monotonicity Hint. To test if h is monotonic, for each data point $\mathbf{x}_n$ we construct $\mathbf{x}'_n = \mathbf{x}_n + \Delta\mathbf{x}_n$ with $\Delta\mathbf{x}_n \geq \mathbf{0}$. Monotonicity implies that $h(\mathbf{x}'_n) \geq h(\mathbf{x}_n)$. We can thus construct a hint error of the form

$$E_{\text{hint}}(h) = \frac{1}{N}\sum_{n=1}^{N}(h(\mathbf{x}_n) - h(\mathbf{x}'_n))^2[[h(\mathbf{x}'_n) < h(\mathbf{x}_n)]].$$

Each term in the hint error penalizes violation of the hint, and is set to be zero if the monotonicity is not violated at $\mathbf{x}_n$, regardless of the value of $(h(\mathbf{x}_n) - h(\mathbf{x}'_n))^2$.

Exercise 9.16

Give hint errors for rotational invariance, convexity and perturbation hints.

9.3.2 Hints Versus Regularization

Regularization (Chapter 4) boiled down to minimizing an augmented error

$$E_{\text{aug}}(h) = E_{\text{in}}(h) + \frac{\lambda}{N}\Omega(h),$$

where the regularizer $\Omega(h)$ penalized the 'complexity' of h. Implementing a hint by minimizing an augmented error as in (9.3) looks like regularization with regularizer $E_{\text{hint}}(h)$. Indeed the two are similar, but we would like to highlight the difference. Regularization directly combats the effects of noise (stochastic and deterministic) by encouraging a hypothesis to be simpler: the primary goal is to reduce the variance, and the price is usually a small increase in bias. A hint helps us choose a small hypothesis set that is likely to contain the target function (see Exercise 9.12): the primary motivation is to reduce the bias. By implementing the hint using a penalty term $E_{\text{hint}}(h)$, we will be reducing the variance but there will usually be no price to pay in terms of higher bias. Indirectly, the effect is to choose the right smaller hypothesis set.

Regularization fights noise by pointing the learning toward simpler hypotheses; this applies to any target function. Hints fight bias by helping us choose wisely from among small hypothesis sets; a hint can hurt if it presents an incorrect bit of information about the target.

As you might have guessed, you can use both tools and minimize

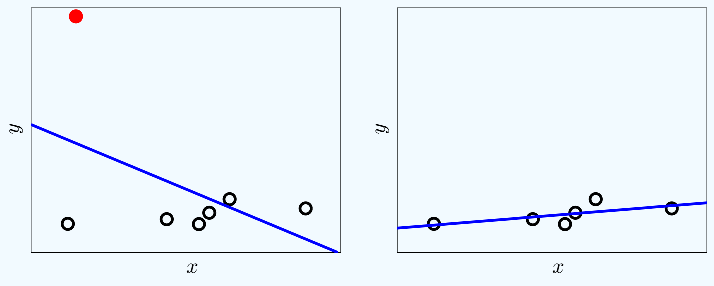
$$E_{\text{aug}}(h) = E_{\text{in}}(h) + \frac{\lambda_1}{N}\Omega(h) + \lambda_2 E_{\text{hint}}(h),$$

thereby combating noise while at the same time incorporating information about the target function. It is possible for the two tools to send the learning in different directions, but this is rare (for example a hint that the target is a 10th order polynomial may get overridden by the regularizer which tells you there is not enough data to learn a 10th order polynomial). It generally pays huge dividends to incorporate properties of the target function, and regularization is a must because there is always noise.

9.4 Data Cleaning

Although having more data is often a good idea, sometimes less is more. Data cleaning attempts to identify and remove noisy or hard examples. A noisy 'outlier' example can seriously lead learning astray (stochastic noise). A complex data point can be just as bad (deterministic noise). To teach a two year old mathematics, you may start with induction or counting $1, 2, 3$. Both are correct 'data points', but the complex concepts of induction will only confuse the child (a simple learner).

Why remove the noisy data as opposed to simply incurring in-sample error on them? You don't just incur in-sample error on the noisy data. You incur additional, unwanted in-sample error on the non-noisy data when the learning is led astray. When you remove the noisy data, you have fewer data points to learn from, but you will gain because these fewer data points have the useful information. The following regression example helps to illustrate.



(a) Fit led astray by outlier (b) Fit without outlier

When we retain the outlier point (red), the linear fit is distorted (even the sign of the slope changes). The principle is not hard to buy into: throw away detrimental data points. The challenge is to identify these detrimental data points. We discuss two approaches that are simple and effective.

9.4.1 Using a Simpler Model to Identify Noisy Data

The learning algorithm sees everything through the lens of the hypothesis set. If your hypothesis set is complex, then you will be able to fit a complex data set, so no data point will look like noise to you. On the other hand, if your hypothesis set is simple, many of the data points could look like noise. We can identify the hard examples by viewing the data through a slightly simpler model than the one you will finally use. One typical choice of the simpler model is a linear model. Data that the simpler model cannot classify are the hard examples, to be disregarded in learning with the complex model. It is a bit of an art to decide on a model that is simple enough to identify the noisy data, but not too simple that you throw away good data which are useful for learning with the more complex final model.

Rather than relying solely on a single simpler hypothesis to identify the noisy data points, it is generally better to have several instances of the simpler hypothesis. If a data point is misclassified by a majority of the simpler hypotheses, then there is more evidence that the data point is hard. One easy way to generate several such simpler hypotheses is to use the same hypothesis set (e.g. linear models) trained on different data sets generated by sampling data points from the original data with replacement (this is called Bootstrap-sampling from the original data set). Example 9.3 illustrates this technique with the digits data from Example 3.1.

Example 9.3. We randomly selected 500 examples of digits data $\mathcal{D}$, and generated more than 10,000 data sets of size 500. To generate a data set, we sampled data points from $\mathcal{D}$ with replacement (Bootstrap sampling).Each bootstrapped data set is used to construct a 'simple' hypothesis using the k-Nearest Neighbor rule for a large k. Each simple hypothesis misclassifies some of the data that was used to obtain that hypothesis. If a particular data point is misclassified by more than half the hypotheses which that data point influenced, we identify that data point as bad (stochastic or deterministic noise). Figure 9.5 shows the bad data for two different choices of the 'simple' model: (a) 5-NN; and (b) 101-NN. The simpler model (101-NN) identifies more con-

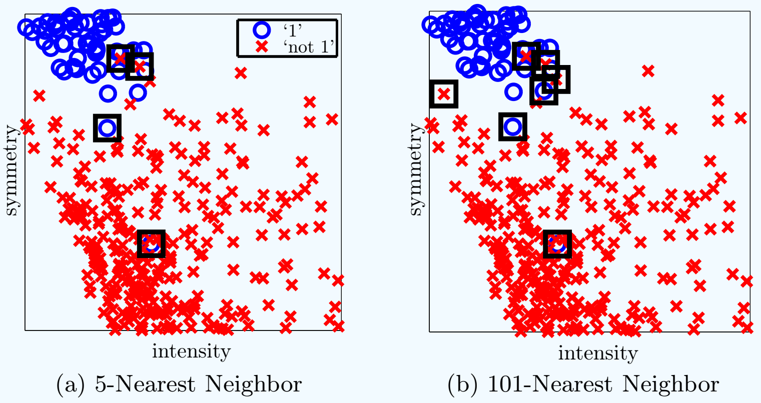


Figure 9.5: Identifying noisy data using the 'simple' nearest neighbor rule. The data in black boxes are identified as noisy (a) using 5-Nearest Neighbor and (b) using 101-Nearest Neighbor. 101-Nearest Neighbor, the simpler of the two models, identifies more noisy data as expected.

fusing data points, as expected. The results depend on our choice of k in the k-NN algorithm. If k is large (generating a constant final hypothesis), then all the '+1'-points will be identified as confusing, because the '−1'-points are in the majority. In Figure 9.5, the points identified as confusing are intuitively so – solitary examples of one class inside a bastion of the other class.

If we choose, for our final classifier, the 'complex' 1-Nearest Neighbor rule, Figure 9.6(a) shows the classifier using all the data (a reproduction of Figure 6.2(a) in Chapter 6). A quick look at the decision boundary for the same 1-Nearest Neighbor rule after removing the bad data (see Figure 9.6(b)) visually confirms that the overfitting is reduced. What matters is the test error, which dropped from 1.7% to 1.1%, a one-third reduction in test error rate.

If computational resources permit, a refinement of this simple and effective method is to remove the noisy points sequentially: first remove the most confusing data point; now rerun the whole process to remove the next data

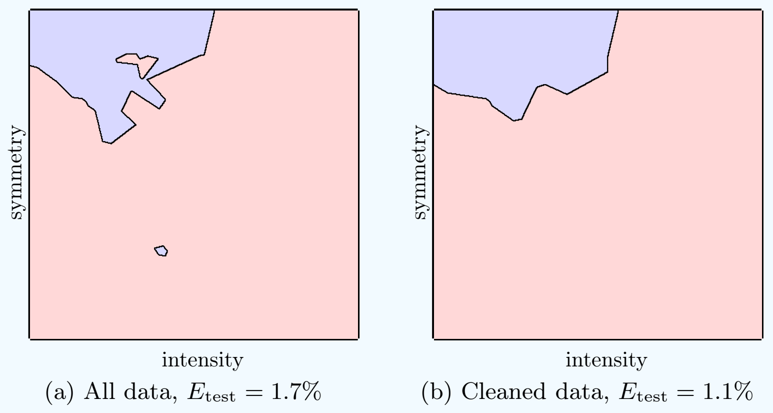


Figure 9.6: The 1-Nearest Neighbor classifier (a) using all the data and (b) after removing the bad examples as determined using 101-Nearest Neighbor. Removing the bad data gives a more believable decision boundary.

point and so on. The advantage of this sequential process is that a slightly confusing example may be redeemed after the true culprit is removed.

In our example, 101-Nearest Neighbor was the 'simple' model, and an example was bad if half the simple hypotheses misclassified that example. The simpler the model, the more outliers it will find. A lower threshold on how often an example is misclassified also results in more outliers. These are implementation choices, and we're in the land of heuristics here. In practice, a slightly simpler model than your final hypothesis and a threshold of 50% for the number of times an example is misclassified are reasonable choices. □

9.4.2 Computing a Validation Leverage Score

If, after removing an example from your data set, you can improve your test error, then by all means remove the example. Easier said than done – how do you know whether the test error will improve when you remove a particular data point? The answer is to use validation to estimate the adverse impact (or leverage) that a data point has on the final hypothesis g. If an example has large leverage with negative impact, then it is a very 'risky' example and should be disregarded during learning.

Let's first define leverage and then see how to estimate it. Recall that using data set $\mathcal{D}$, the learning produces final hypothesis g. When we discussed validation in Chapter 4, we introduced the data set $\mathcal{D}_n$ which contained all the data in $\mathcal{D}$ except $(\mathbf{x}_n, y_n)$. We denoted by g_n^- the final hypothesis that you get from from $\mathcal{D}_n$. We denote the *leverage score* of data point $(\mathbf{x}_n, y_n)$ by

$$\ell_n = E_{\text{out}}(g) - E_{\text{out}}(g_n^-).$$

The leverage score measures how valuable $(\mathbf{x}_n, y_n)$ is. If ℓ_n is large and positive, $(\mathbf{x}_n, y_n)$ is detrimental and should be discarded. To estimate ℓ_n, we need a validation method to estimate E_{out}. Any method for validation can be used. In Chapter 4 we introduced E_{cv}, the cross-validation estimate of $E_{\text{out}}(g)$; E_{cv} is an unbiased estimate of the out-of-sample performance when learning from $N-1$ data points. The algorithm to compute E_{cv} takes as input a data set $\mathcal{D}$ and outputs $E_{\text{cv}}(\mathcal{D})$, an estimate of the out-of-sample performance for the final hypothesis learned from $\mathcal{D}$. Since we get g_n^- from $\mathcal{D}_n$, if we run the cross validation algorithm on $\mathcal{D}_n$, we will get an estimate of $E_{\text{out}}(g_n^-)$. Thus,

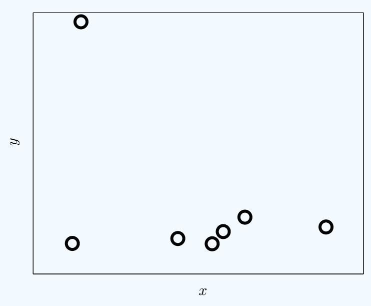
$$\ell_n \approx E_{\text{cv}}(\mathcal{D}) - E_{\text{cv}}(\mathcal{D}_n). \tag{9.4}$$

You can replace the E_{cv} algorithm above with any validation algorithm, provided you run it once on $\mathcal{D}$ and once on $\mathcal{D}_n$. The computation of $E_{\text{cv}}(\mathcal{D})$ only needs to be performed once. Meanwhile, we are asking whether each data point in turn is helping. If we are using cross-validation to determine whether a data point is helping, $E_{\text{cv}}(\mathcal{D}_n)$ needs to be computed for $n = 1, \ldots, N$, requiring you to learn $N(N-1)$ times on data sets of size $N-2$. That is a formidable feat. For linear regression, this can be done more efficiently (see Problem 9.21). Example 9.4 illustrates the use of validation to compute the leverage for our toy regression example that appeared earlier.

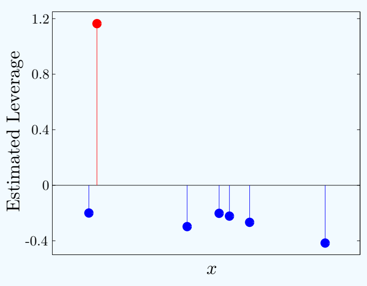
Example 9.4. We use a linear model to fit the data shown in Figure 9.7(a). There are seven data points, so we need to compute

$$E_{\text{cv}}(\mathcal{D}) \qquad \text{and} \qquad E_{\text{cv}}(\mathcal{D}_1), \ldots, E_{\text{cv}}(\mathcal{D}_7).$$

We show the leverage $\ell_n \approx E_{\text{cv}}(\mathcal{D}) - E_{\text{cv}}(\mathcal{D}_n)$ for $n = 1, \ldots, 7$ in Figure 9.7(b).



(a) Data for linear regression



(b) Leverage estimates

Figure 9.7: Estimating the leverage of data points using cross validation. (a) The data points with one outlier (b) The estimated leverage. The outlier has a large positive leverage indicative of a very noisy point.

Only the outlier has huge positive leverage, and it should be discarded. □

9.5 More on Validation

We introduced validation in Chapter 4 as a means to select a good regularization parameter or a good model. We saw in the previous section how validation can be used to identify noisy data points that are detrimental to learning. Validation gives an in-sample estimate for the out-of-sample performance. It lets us certify that the final hypothesis is good, and is such an important concept that many techniques have been developed. Time spent augmenting our tools for validation is well worth the effort, so we end this chapter with a few advanced techniques for validation. Don't fear. The methods are advanced, but the algorithms are simple.

9.5.1 Rademacher Penalties

The Rademacher penalty estimates the optimistic bias of the in-sample error.

$$E_{\text{out}}(g) = E_{\text{in}}(g) + \underbrace{\text{overfit penalty}}_{\text{Rademacher penalty estimates this}}.$$

One way to view the overfit penalty is that it represents the optimism in the in-sample error. You reduced the error all the way to $E_{\text{in}}(g)$. Part of that was real, and the rest was just you fooling yourself. The Rademacher overfit penalty attempts to estimate this optimism.

The intuitive leap is to realize that the optimism in E_{in} is a result of the fitting capability of your hypothesis set $\mathcal{H}$, not because, by accident, your data set was easy to fit. This suggests that if we can compute the optimism penalty for some data set, then we can apply it to other data sets, for example, to the data set $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{\text{N}}, y_{\text{N}})$ that we are given. So let's try to find a data set *for which we can compute the optimism.* And indeed there is one such data set, namely a random one. Consider the data set

$$\mathcal{D}' = (\mathbf{x}_1, r_1), \ldots, (\mathbf{x}_{\text{N}}, r_{\text{N}}),$$

where $r_1, \ldots, r_n$ are generated independently by a random target function, $\mathbb{P}[r_n = +1] = \frac{1}{2}$. The r_n are called Rademacher variables. The inputs are the same as the original data set, so in that sense $\mathcal{D}'$ mimics $\mathcal{D}$. After learning on $\mathcal{D}'$ you produce hypothesis $g_{\mathbf{r}}$ with in-sample error $E'_{\text{in}}(g_{\mathbf{r}})$ (we use the prime $(\cdot)'$ to denote quantities with respect to the random problem). Clearly, $E'_{\text{out}}(g_{\mathbf{r}}) = \frac{1}{2}$, because the target function is random, so the optimism is $\frac{1}{2} - E'_{\text{in}}(g_{\mathbf{r}})$. Using this measure of optimism for our actual problem, we get the Rademacher estimate

$$\hat{E}_{\text{out}}(g) = E_{\text{in}}(g) + \left(\tfrac{1}{2} - E'_{\text{in}}(g_{\mathbf{r}})\right).$$

The Rademacher penalty is easy to compute: generate random targets for your data; perform an in-sample minimization to see how far below $\frac{1}{2}$ you can get the error, and use that as a penalty on your actual in-sample error.

Exercise 9.17

After you perform in-sample minimization to get $g_{\mathbf{r}}$, show that the Rademacher penalty for $E_{\text{in}}(g)$ is given by

$$\tfrac{1}{2} - E'_{\text{in}}(g_{\mathbf{r}}) = \max_{h\in\mathcal{H}} \left\{ \frac{1}{2N} \sum_{n=1}^{N} r_n h(\mathbf{x}_n) \right\},$$

which is proportional to the maximum correlation you can obtain with random signs using hypotheses in the data set.

A better way to estimate the optimism penalty is to compute the expectation over different realizations of the Rademacher variables: $\mathbb{E}_{\mathbf{r}}[\frac{1}{2} - E'_{\text{in}}(g_{\mathbf{r}})]$. In practice, one generates several random Rademacher data sets and takes the average optimism penalty. This means you have to do an in-sample error minimization for each Rademacher data set. The good news is that, with high probability, just a single Rademacher data set suffices.

It is possible to get a theoretical bound for $E_{\text{out}}(g)$ using the Rademacher overfit penalty (just like we did with the VC penalty). The bound is universal (because it holds for any learning scenario – target function and input probability distribution); but, unlike the VC penalty, it is *data dependent*, hence it tends to be tighter in practice. (Remember that the VC bound was independent of the data set because it computed $m_{\mathcal{H}}(N)$ on the worst possible data set of size N.) It is easy to use the Rademacher penalty, since you just need to do a single in-sample error optimization. In this sense the approach is similar to cross-validation, except the Rademacher penalty is computationally cheaper (recall that cross-validation required N in-sample optimizations).

The Rademacher penalty requires in-sample error *minimization* - you compute the worst-case optimism. However, it can also be used with regularization and penalized error minimization, because to a good approximation, penalized error minimization is equivalent to in-sample error minimization using a constrained hypothesis set (see Chapter 4). Thus, when we say "in-sample error minimization," read it as "run the learning algorithm."

9.5.2 The Permutation and Bootstrap Penalties

There are other random data sets for which we can compute the optimism. The permutation and Bootstrap estimates are based off these, and are just as easy to compute using an in-sample error minimization.

Permutation Optimism Penalty. Again, we estimate the optimism by considering how much we can overfit random data, but now we choose the y-values to be a random permutation of the actual y-values in the data, as opposed to random signs. Thus, the estimate easily generalizes to regression as well. The learning problem generated by a random permutation mimics the actual learning problem more closely since it takes both the $\mathbf{x}$ and y values from the data. Consider a randomly permuted data set,

$$\mathcal{D}_{\boldsymbol{\pi}} = (\mathbf{x}_1, y_{\boldsymbol{\pi}_1}), \ldots, (\mathbf{x}_N, y_{\boldsymbol{\pi}_N}),$$

where $\boldsymbol{\pi}$ is a random permutation of $(1, \ldots, N)$. After in-sample error minimization on this randomly permuted data, you produce $g_{\boldsymbol{\pi}}$ with in-sample error on $\mathcal{D}_{\boldsymbol{\pi}}$ equal to $E_{\text{in}}^{\pi}(g_{\boldsymbol{\pi}})$. Unlike with the Rademacher random learning problem, the out-of-sample error for this random learning problem is not simply $\frac{1}{2}$, because the target values are not random signs. We need to be a little more careful and obtain the expected error for the joint $(\mathbf{x}, y)$ target distribution that describes this random learning problem (see Problem 9.12 for the details). However, the final result is the same, namely that the overfit penalty (optimism on the random problem) is proportional to the correlation between the learned function $g_{\boldsymbol{\pi}}(\mathbf{x}_n)$ and the randomly permuted target values $y_{\boldsymbol{\pi}_n}$.[6] The permutation estimate of the out-of-sample error is

$$\hat{E}_{\text{out}}(g) = E_{\text{in}}(g) + \frac{1}{2N}\sum_{n=1}^{N}(y_{\boldsymbol{\pi}_n} - \bar{y})g_{\boldsymbol{\pi}}(\mathbf{x}_n), \tag{9.5}$$

where $\bar{y}$ is the mean target value in the data. The second term is the permutation optimism penalty obtained from a single randomly permuted data set. Ideally, you should average the optimism penalty over several random permutations, even though, as with the Rademacher estimate, one can show that a single permuted data set suffices to give a good estimate with high probability. As already mentioned, the permutation estimate can be applied to regression, in which case it is more traditional not to rescale the sum of squared errors by $\frac{1}{4}$ (which is appropriate for classification). In this case, the $\frac{1}{2N}$ becomes $\frac{2}{N}$.

Bootstrap Optimism Penalty. The Rademacher and permutation data sets occupy two extremes. For the Rademacher data set, we choose the y-values as random signs, independent of the actual values in the data. For the permutation data set, we take the y-values directly from the data and randomly permute them - every target value is represented once, and can be viewed as sampling the target values from the data *without* replacement. The Bootstrap optimism penalty generates a random set of targets, one for each input, by sampling y-values in the data, but *with* replacement. Thus, the y-values represent the observed distribution, but not every y-value may be picked. The functional form of the estimate (9.5) is unchanged; one simply replaces all the terms in the overfit penalty with the corresponding terms for the Bootstrapped data set.

The permutation and Rademacher estimates may be used for model selection just as any other method which constructs a proxy for the out-of-sample

[6] Any such correlation is spurious, as no input-output relationship exists.

error. In some cases, as with leave-one-out cross validation, it is possible to compute the average (over the random data sets) of the overfit penalty analytically, which means that one can use the permutation estimate without having to perform any explicit in-sample error optimizations.

Example 9.5 (Permutation Estimate for Linear Regression with Weight Decay)**.** For linear models, we can compute the permutation estimate without explicitly learning on the randomly permuted data (see Problem 9.18 for details). The estimate for E_{out} is

$$\begin{aligned}\hat{E}_{\text{out}}(g) &= E_{\text{in}}(g) + \mathbb{E}_{\boldsymbol{\pi}}\left[\frac{2}{N}\sum_{n=1}^{N}(y_{\boldsymbol{\pi}_n} - \bar{y})g_{\boldsymbol{\pi}}(\mathbf{x}_n)\right] \\ &= E_{\text{in}}(g) + \frac{2\hat{\sigma}_y^2}{N}\left(\text{trace}(\text{H}) - \frac{\mathbf{1}^{\text{T}}\text{H}\mathbf{1}}{N}\right), \end{aligned} \tag{9.6}$$

where $\hat{\sigma}_y^2 = \frac{N}{N-1}\mathsf{var}(y)$ is the unbiased estimate for the variance of the target values and $\text{H} = \text{X}(\text{X}^{\text{T}}\text{X} + \frac{\lambda}{N}\text{I})^{-1}\text{X}^{\text{T}}$ is the hat-matrix that depends only on the input data. When there is no regularization, $\lambda = 0$, so H is a projection matrix (projecting onto the columns of X), and $\text{trace}(\text{H}) = d + 1$. If the first column of X is a column of ones (the constant term in the regression), then $\text{H}\mathbf{1} = \mathbf{1}$ and the permutation estimate becomes

$$\hat{E}_{\text{out}}(g) = E_{\text{in}}(g) + \frac{2\hat{\sigma}_y^2 d}{N}.$$

This estimate is similar to the AIC estimate from information theory. It is also reminiscent of the test error in Exercise 3.4 which resulted in

$$E_{\text{out}}(g) = E_{\text{in}}(g) + \frac{2\sigma^2(d+1)}{N},$$

where σ^2 was the noise variance. The permutation estimate uses $\hat{\sigma}_y^2$ in place of σ^2. Observe that $\text{trace}(\text{H}) - \frac{1}{N}\mathbf{1}^{\text{T}}\text{H}\mathbf{1}$ plays the role of an effective dimension, d_{eff}. Problem 4.13 in Chapter 4 suggests some alternative choices for an effective dimension. □

9.5.3 Rademacher Generalization Bound

The theoretical justification for the Rademacher optimism penalty is that, as with the VC bound, it can be used to bound E_{out}. A similar though mathematically more complex justification can also be shown for the permutation estimate. The mathematical proof of this is interesting in that it presents new tools for analyzing generalization error. An energetic reader may wish to master these techniques by working through Problem 9.16. We present the result for a hypothesis set that is closed under negation ($h \in \mathcal{H}$ implies $-h \in \mathcal{H}$).

Theorem 9.6 (Rademacher Bound)**.** With probability at least $1-\delta$,

$$E_{\text{out}}(g) \le E_{\text{in}}(g) + \max_{h\in\mathcal{H}} \left\{ \tfrac{1}{N} \textstyle\sum_{n=1}^{N} r_n h(\mathbf{x}_n) \right\} + O\left(\sqrt{\tfrac{1}{N} \log \tfrac{1}{\delta}} \right).$$

The probability is with respect to the data set and a single realization of the Rademacher variables.

The overfit penalty in the bound is twice the Rademacher optimism penalty (we can ignore the third term which does not depend on the hypothesis set and is small). The bound is similar to the VC bound in that it holds for any target function and input distribution. Unlike the VC bound, it depends on the data set and can be significantly tighter than the VC bound.

9.5.4 Out-of-Sample Error Estimates for Regression

We close this chapter with a discussion of validation methods for regression, and a model selection experiment in the context of regression, to illustrate some of the issues to think about when using validation for model selection. Cross-validation and the permutation optimism penalty are general in that they can be applied to regression and only require in-sample minimization. They also require no assumptions on the data distribution to work. We call these types of approaches sampling approaches because they work using samples from the data.

The VC bound only applies to classification. There is an analog of the VC bound for regression. The statistics community has also developed a number of estimates. The estimates are multiplicative in form, so $E_{\text{out}} \approx (1+\Omega(p))E_{\text{in}}$, where $p = \frac{N}{d_{\text{eff}}}$ is the number of data points per effective degree of freedom (see Example 9.5 for one estimate of the effective degrees of freedom in a linear regression setting).

VC penalty factor	$E_{\text{out}} \le \dfrac{\sqrt{p}}{\sqrt{p} - \sqrt{1 + \ln p + \frac{\ln N}{2d_{\text{eff}}}}} E_{\text{in}}$
Akaike's FPE	$E_{\text{out}} \approx \dfrac{p+1}{p-1} E_{\text{in}}$
Schwartz criterion	$E_{\text{out}} \approx \left(1 + \dfrac{\ln N}{p-1}\right) E_{\text{in}}$
Craven & Wahba's GCV	$E_{\text{out}} \approx \dfrac{p^2}{(p-1)^2} E_{\text{in}}$

For large p, FPE (final prediction error) and GCV (generalized cross validation) are similar: $E_{\text{out}} = (1 + 2p^{-1} + O(p^{-2}))E_{\text{in}}$. This suggests that the simpler estimator $E_{\text{out}} = (1 + \frac{2d_{\text{eff}}}{N})E_{\text{in}}$ might be good enough, and, from the practical point of view, it is good enough.

The statistical estimates (FPE, Schwartz and GCV) make model assumptions and are good asymptotic estimates modulo those assumptions. The VC penalty is more generally valid. However, it tends to be very conservative. In practice, the VC penalty works quite well, as we will see in the experiments.

Validation Experiments with Regression. The next exercise sets up an experimental design for studying validation and model selection. We consider two model selection problems. The first is to determine the right order of the polynomial to fit the data. In this case, we have models $\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2, \ldots$, corresponding to the polynomials of degree $0, 1, 2, \ldots$. The model selection task is to select one of these models. The second is to determine the right value of the regularization parameter λ in the weight decay regularizer. In this case, we fix a model (say) $\mathcal{H}_Q$ and minimize the augmented error $E_{\text{aug}}(\mathbf{w}) = E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N}\mathbf{w}^{\text{T}}\mathbf{w}$; we have models $\mathcal{H}_Q(\lambda_1), \mathcal{H}_Q(\lambda_2), \mathcal{H}_Q(\lambda_3), \ldots$, corresponding to the choices $\lambda_1, \lambda_2, \lambda_3, \ldots$ for the regularization parameter. The model selection task is to select one of these models, which corresponds to selecting the appropriate amount of regularization.

Exercise 9.18 [Experimental Design for Model Selection]

Use the learning set up in Exercise 4.2.

(a) **Order Selection.** For a single experiment, generate $\sigma^2 \in [0, 1]$ and the target function degree in $\{0, \ldots, 30\}$. Set $N = 100$ and generate a data set $\mathcal{D}$. Consider the models up to order 20: $\mathcal{H}_0, \ldots, \mathcal{H}_{20}$. For these models, obtain the final hypotheses $g_0, \ldots, g_{20}$ and their test errors $E_{\text{out}}^{(1)}, \ldots, E_{\text{out}}^{(20)}$. Now obtain estimates of the out-of-sample errors, $\hat{E}_{\text{out}}^{(1)}, \ldots, \hat{E}_{\text{out}}^{(20)}$, using the following validation estimates:

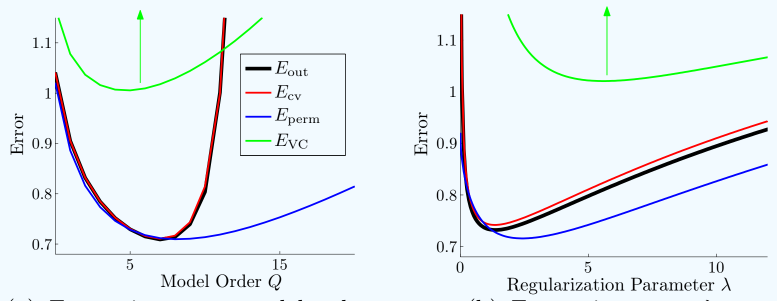
VC penalty; LOO-CV; Permutation estimate; FPE.

Plot the out-of-sample error and the validation estimates versus the model order, averaged over many experiments.

(b) λ **Selection.** Repeat the previous exercise to select the regularization parameter. Fix the order of the model to $Q = 5$. Randomly generate $\sigma^2 \in [0, 1]$ and the order of the target from $\{0, \ldots, 10\}$, and set $N = 15$. Use different models with $\lambda \in [0, 300]$.

(c) For model selection, you do not need a perfect error estimate. What is a weaker, but sufficient, requirement for a validation estimate to give good model selection?

(d) To quantify the quality of an error estimate, we may use it to select a model and compute the actual out-of-sample error of the selected model, versus the model with minimum out-of-sample error (after training with the same data). The regret is the relative increase in error incurred by using the selected model versus the optimum model. Compare the regret of different validation estimates, taking the average over many runs.



(a) E_{out} estimate vs. model order (b) E_{out} estimate vs. λ

Figure 9.8: Validation estimates of E_{out}. (a) polynomial models of different order Q; (b) models with different regularization parameter λ. The estimates and E_{out} are averaged over many runs; the 3 validation methods shown are E_{CV} (leave-one-out cross validation estimate), E_{perm} (permutation estimate), and E_{VC} (VC penalty bound). The VC penalty bound is *far* above the plot because it is so loose (indicated by the arrow); we have shifted it down to show its shape. The VC estimate is not useful for estimating E_{out} but can be useful for model selection.

The average performance of validation estimates of E_{out} from our implementation of Exercise 9.18 are shown in Figure 9.8. As can be seen from the figure, when it comes to estimating the out-of-sample error, the cross validation estimate is hands down favorite (on average). The cross validation estimate has a systematic bias because it is actually estimating the out-of-sample error when learning with $N-1$ examples, and the actual out-of-sample error with N examples will be lower. The out-of-sample error is asymmetric about its minimum – the price paid for overfitting increases more steeply than the price paid for underfitting. As can be seen by comparing where the minimum of these curves lies, the VC estimate is the most conservative. The VC overfit penalty strongly penalizes complex models, so that the optimum model (according to the VC estimate) is much simpler.

We now compare the different validation estimates for model selection (Exercise 9.18(d)). We look at the *regret*, the average percentage increase in E_{out} from using an E_{out}-estimate to pick a model versus picking the optimal model. These results are summarized in the table of Figure 9.9. The surprising thing is that the permutation estimate and the VC estimate seem to dominate, even though cross validation gives the best estimate of E_{out} on average. This has to do with the asymmetry of E_{out} around the best model. It pays to be conservative, and err on the side of the simpler model (underfitting) than on the side of the more complex model (overfitting), because the price paid for overfitting is far steeper than the price paid for underfitting. The permutation estimate tends to underfit, and the VC estimate even more so; these estimates are paying the 'small' price for underfitting most of the time. On the other hand,

E_{out} Estimate	Order Selection		λ Selection	
	Regret	Avg. Order	Regret	Avg. λ
E_{out}	0	10.0	0	7.93
E_{in}	43268	20.0	117	0
E_{CV}	540	9.29	18.8	23.1
E_{perm}	**185**	**7.21**	5.96	9.57
E_{FPE}	9560	11.42	51.3	18.1
E_{VC}	508	5.56	**3.50**	**125**

Figure 9.9: Using estimates of E_{out} model selection (Exercise 9.18(d)). The best validation estimate is highlighted in bold. For order selection, the order $Q \in \{0, \ldots, 20\}$, and for regularization $\lambda \in [0, 300]$. All results are averaged over many thousands of experiments. (Regret $= \frac{E_{\text{out}}(g) - E_{\text{out}}(\text{optimal})}{E_{\text{out}}(\text{optimal})}$.)

cross validation picks the correct model most of the time, but occasionally goes for too complex a model and pays a very steep price. In the end, the more conservative strategy wins. Using E_{in} is always the worst, no surprise.

You always need regularization, so $\lambda = 0$ is bad. Suppose we remove that case from our set of available models. Now repeat the experiment for selecting the λ in the range $[0.1, 300]$, as opposed to the range $[0, 300]$, a very minor difference in the available choices for λ. The results become

$\lambda \in [0.1, 300]$	E_{out}	E_{in}	E_{CV}	E_{perm}	E_{FPE}	E_{VC}
Regret	0	1.81	0.44	**0.39**	0.87	0.42

The permutation estimate is best. Cross validation is now much better than before. The in-sample error is still by far the worst. Why did all methods get much better? Because we curtailed their ability to err in favor of too complex a model. What we were experiencing is 'overfitting' during the process of model selection, and by removing the choice $\lambda = 0$, we 'regularized' the model selection process. The cross validation estimate is the most accurate, but also the most sensitive, and has a high potential to be led astray, thus it benefited most from the regularization of the model selection. The FPE estimate is a statistical estimate. Such statistical estimates typically make model assumptions and are most useful in the large N limit, so their applicability in practice may be limited. The statistical estimate fares okay, but, for this experiment, it is not really competitive with the permutation approach, cross validation, or the VC penalty.

Our recommendation. Do not arbitrarily pick models to select among. One can 'overfit' during model selection. Carefully decide on a set of models and use a robust validation method to select one. There is no harm in using multiple methods for model selection, and we recommend both the permutation estimate (robust, easy to compute and generally applies) and cross validation (general, easy to use and usually the most accurate for estimating E_{out}).

9.6 Problems

Problem 9.1 Consider data $(0,0,+1),(0,1,+1),(5,5,-1)$ (in 2-D), where the first two entries are x_1, x_2 and the third is y.

(a) Implement the nearest neighbor method on the raw data and show the decision regions of the final hypothesis.

(b) Transform to whitened coordinates and run the nearest neighbor rule, showing the final hypothesis in the original space.

(c) Use principal components analysis and reduce the data to 1 dimension for your nearest neighbor classifier. Again, show the decision regions of the final hypothesis in the original space.

Problem 9.2 We considered three forms of input preprocessing: centering, normalization and whitening. The goal of input processing is to make the learning robust to unintentional choices made during data collection.

Suppose the data are $\mathbf{x}_n$ and the transformed vectors $\mathbf{z}_n$. Suppose that during data collection, $\mathbf{x}'_n$, a mutated version of $\mathbf{x}_n$, were measured, and input preprocessing on $\mathbf{x}'_n$ produces $\mathbf{z}'_n$. We would like to study when the $\mathbf{z}'_n$ would be the same as the $\mathbf{z}_n$. In other words, what kinds of mutations can the data vectors be subjected to without changing the result of your input processing. The learning will be robust to such mutations.

Which input processing methods are robust to the following mutations:

(a) Bias: $\mathbf{x}'_n = \mathbf{x}_n + \mathbf{b}$ where $\mathbf{b}$ is a constant vector.

(b) Uniform scaling: $\mathbf{x}'_n = \alpha\mathbf{x}_n$, where $\alpha > 0$ is a constant.

(c) Scaling: $\mathbf{x}'_n = \mathrm{A}\mathbf{x}_n$ where A is a diagonal non-singular matrix.

(d) Linear transformation: $\mathbf{x}'_n = \mathrm{A}\mathbf{x}_n$ where A is a non-singular matrix.

Problem 9.3 Let Σ be a symmetric positive definite matrix with eigendecomposition $\Sigma = \mathrm{U}\Gamma\mathrm{U}^{\mathrm{T}}$ (see the Appendix), where U is orthogonal and Γ is positive diagonal. Show that

$$\Sigma^{\frac{1}{2}} = \mathrm{U}\Gamma^{\frac{1}{2}}\mathrm{U}^{\mathrm{T}} \qquad \text{and} \qquad \Sigma^{-\frac{1}{2}} = \mathrm{U}\Gamma^{-\frac{1}{2}}\mathrm{U}^{\mathrm{T}}.$$

What are $\Gamma^{\frac{1}{2}}$ and $\Gamma^{-\frac{1}{2}}$?

Problem 9.4 If A and V are orthonormal bases, and $\mathrm{A} = \mathrm{V}\psi$, show that $\psi = \mathrm{V}^{\mathrm{T}}\mathrm{A}$ and hence that ψ is an orthogonal matrix.

Problem 9.5 Give the SVD of matrix A defined in each case below.

(a) A is a diagonal matrix.

(b) A is a matrix with pairwise orthogonal rows.

(c) A is a matrix with pairwise orthogonal columns.

(d) Let A have SVD $U\Gamma V^{\mathrm{T}}$ and $Q^{\mathrm{T}}Q = I$. What is the SVD of QA.

(e) A has blocks A_i along the diagonal, where A_i has SVD $U_i\Gamma_i V_i^{\mathrm{T}}$,

$$A = \begin{bmatrix} \boxed{A_1} & & & \\ & \boxed{A_2} & & \mathbf{0} \\ & \mathbf{0} & \ddots & \\ & & & \boxed{A_m} \end{bmatrix}.$$

Problem 9.6 For the digits data, suppose that the data were not centered first in Example 9.2. Perform PCA and obtain a 2-dimensional feature vector (give a plot). Are the transformed data whitened?

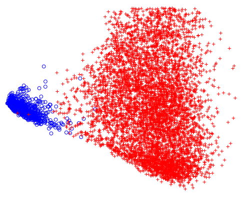
Problem 9.7 Assume that the input matrix X is centered, and construct the reduced feature vector $\mathbf{z}_n = \Gamma_k^{-1}V_k^{\mathrm{T}}\mathbf{x}_n$ where V_k is the matrix of top-k right singular vectors of X. Show that the feature vectors $\mathbf{z}_n$ are whitened.

Problem 9.8 One critique of PCA is that it selects features without regard to target values. Thus, the features may not be so useful in solving the learning problem, even though they represent the input data well.

One heuristic to address this is to do a PCA separately for each class, and use the top principal direction for each class to construct the features. Use the digits data. Construct a two dimensional feature as follows.

```
1: Center all the data.
2: Perform PCA on the +1 data. Let $\mathbf{v}_1$ be the top
   principal direction.
3: Perform PCA on the -1 data. Let $\mathbf{v}_1$ be the top
   principal direction.
4: Use $\mathbf{v}_1$ and $\mathbf{v}_2$ to obtain the features

         $z_1 = \mathbf{v}_1^{\mathrm{T}}\mathbf{x}$   and   $z_2 = \mathbf{v}_2^{\mathrm{T}}\mathbf{x}$.
```



We applied the algorithm to the digits data, giving the features shown above.

(a) Give a scatter plot of your resulting two features.

(b) Are the directions $\mathbf{v}_1$ and $\mathbf{v}_2$ necessarily orthogonal?

(c) Let $\mathrm{Z} = \mathrm{X}\hat{\mathrm{V}}$ be the feature matrix, where $\hat{\mathrm{V}} = [\mathbf{v}_1, \mathbf{v}_2]$. Show that the best reconstruction of X from Z is

$$\hat{\mathrm{X}} = \mathrm{Z}\hat{\mathrm{V}}^{\dagger} = \mathrm{X}\hat{\mathrm{V}}\hat{\mathrm{V}}^{\dagger},$$

where $\hat{\mathrm{V}}^{\dagger}$ is the pseudo-inverse of$\hat{\mathrm{V}}$.

(d) Is this method of constructing features supervised or unsupervised?

Problem 9.9 Let $\mathrm{X} = \mathrm{U}\Gamma\mathrm{V}^{\mathrm{T}}$ be the SVD of X. Give an alternative proof of the Eckart-Young theorem by showing the following steps.

(a) Let $\hat{\mathrm{X}}$ be a reconstruction of X whose rows are the data points in X projected onto k basis vectors. What is the rank($\hat{\mathrm{X}}$)?

(b) Show that $\|\mathrm{X} - \hat{\mathrm{X}}\|_F^2 \geq \|\mathrm{U}^{\mathrm{T}}(\mathrm{X} - \hat{\mathrm{X}})\mathrm{V}\|_F^2 = \|\Gamma - \mathrm{U}^{\mathrm{T}}\hat{\mathrm{X}}\mathrm{V}\|_F^2$.

(c) Let $\hat{\Gamma} = \mathrm{U}^{\mathrm{T}}\hat{\mathrm{X}}\mathrm{V}$. Show that rank($\hat{\Gamma}$) $\leq k$.

(d) Argue that the optimal choice for $\hat{\Gamma}$ must have all off-diagonal elements zero. *[Hint: What is the definition of the Frobenius norm?]*

(e) How many non-zero diagonals can $\hat{\Gamma}$ have.

(f) What is the best possible choice for $\hat{\Gamma}$.

(g) Show that $\hat{\mathrm{X}} = \mathrm{X}\mathrm{V}\mathrm{V}^{\mathrm{T}}$ results in such an optimal choice for $\hat{\Gamma}$.

Problem 9.10 **Data Snooping with Input Preprocessing.** You are going to run PCA on your data X and select the top-k PCA features. Then, use linear regression with these k features to produce your final hypothesis. You want to estimate E_{out} using cross validation. Here are two possible algorithms for getting a cross-validation error.

Algorithm 1
```
1: SVD: $[\mathrm{U}, \Gamma, \mathrm{V}] = \mathrm{svd}(\mathrm{X})$.
2: Get features $\mathrm{Z} = \mathrm{X}\mathrm{V}_k$
3: for $n = 1 : N$ do
4:     Leave out $(\mathbf{z}_n, y_n)$ to obtain data $\mathrm{Z}_n, \mathbf{y}_n$.
5:     Learn $\mathbf{w}_n^-$ from $\mathrm{Z}_n, \mathbf{y}_n$
6:     Error $\mathrm{e}_n = (\mathbf{x}_n^{\mathrm{T}}\mathbf{w}_n^- - y_n)^2$.
7: $E_1 = \mathrm{average}(\mathrm{e}_1, \ldots, \mathrm{e}_N)$.
```

Algorithm 2
```
1: for $n = 1 : N$ do
2:     Leave out $(\mathbf{x}_n, y_n)$ to obtain data $\mathrm{X}_n, \mathbf{y}_n$.
3:     SVD: $[\mathrm{U}^-, \Gamma^-, \mathrm{V}^-] = \mathrm{svd}(\mathrm{X}_n)$.
4:     Get features $\mathrm{Z}_n = \mathrm{X}_n\mathrm{V}_k^-$.
5:     Learn $\mathbf{w}_n^-$ from $\mathrm{Z}_n, \mathbf{y}_n$.
6:     Error $\mathrm{e}_n = (\mathbf{x}_n^{\mathrm{T}}\mathrm{V}_2^-\mathbf{w}_n^- - y_n)^2$
7: $E_2 = \mathrm{average}(\mathrm{e}_1, \ldots, \mathrm{e}_N)$.
```

In both cases, your final hypothesis is $g(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathrm{V}_k\mathbf{w}$ where $\mathbf{w}$ is learned from $\mathrm{Z} = \mathrm{X}\mathrm{V}_k$ form Algorithm 1 and $\mathbf{y}$. (Recall V_k is the matrix of top-k singular singular vectors.) We want to estimate $E_{\text{out}}(g)$ using either E_1 or E_2.

(a) What is the difference between Algorithm 1 and Algorithm 2?

(b) Run an experiment to determine which is a better estimate of $E_{\text{out}}(g)$.

(i) Set the dimension $d = 5$, the number of data points $N = 40$ and $k = 3$. Generate random target function weights $\mathbf{w}_f$, normally distributed. Generate N random normally distributed d-dimensional inputs $\mathbf{x}_1, \ldots, \mathbf{x}_N$. Generate targets $y_n = \mathbf{w}_f^{\mathrm{T}}\mathbf{x}_n + \epsilon_n$ where ϵ_n is independent Gaussian noise with variance 0.5.

(ii) Use Algorithm 1 and 2 to compute estimates of $E_{\text{out}}(g)$.

(iii) Compute $E_{\text{out}}(g)$.

(iv) Report $E_1, E_2, E_{\text{out}}(g)$.

(v) Repeat parts (i)–(iv) 10^6 times and report averages $\bar{E}_1, \bar{E}_2, \bar{E}_{\text{out}}$.

(c) Explain your results from (b) part (v).

(d) Which is the correct estimate for $E_{\text{out}}(g)$.

Problem 9.11 From Figure 9.8(a), performance degrades rapidly as the order of the model increases. What if the target function has a high order? One way to allow for higher order, but still get good generalization is to fix the effective dimension d_{eff}, and try a variety of orders. Given X, d_{eff} depends on λ. Fixing d_{eff} (to say 7) requires changing λ as the order increases.

(a) For fixed d_{eff}, when the order increases, will λ increase or decrease?

(b) Implement a numerical method for finding $\lambda(d_{\text{eff}})$ using one of the measures for the effective number of parameters (e.g., $d_{\text{eff}} = \text{trace}(\mathrm{H}^2(\lambda))$, where $\mathrm{H}(\lambda) = \mathrm{X}(\mathrm{X}^{\mathrm{T}}\mathrm{X}+\lambda\mathrm{I})^{-1}\mathrm{X}^{\mathrm{T}}$ is the hat-matrix from Chapter 4). The inputs are d_{eff}, and X and the output should be $\lambda(d_{\text{eff}}, \mathrm{X})$.

(c) Use the experimental design in Exercise 9.18 to evaluate this approach. Vary the model order in $[0, 30]$ and set d_{eff} to 7. Plot the expected out-of-sample error versus the order.

(d) Comment on your results. How should your plot behave if d_{eff} alone controlled the out-of-sample error? What is the best model order using this approach?

Problem 9.12 In this problem you will derive the permutation optimism penalty in Equation (9.5). We compute the optimism for a particular data distribution and use that to penalize E_{in} as in (9.5). The data is $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, and the permuted data set is

$$\mathcal{D}_{\boldsymbol{\pi}} = (\mathbf{x}_1, y_{\boldsymbol{\pi}_1}), \ldots, (\mathbf{x}_N, y_{\boldsymbol{\pi}_N}),$$

Define the 'permutation input distribution' to be uniform over $\mathbf{x}_1, \ldots, \mathbf{x}_N$. Define a 'permutation target function' $f_{\boldsymbol{\pi}}$ as follows: to compute target values $f_{\boldsymbol{\pi}}(\mathbf{x}_n)$, generate a random permutation $\boldsymbol{\pi}$ and set $f_{\boldsymbol{\pi}}(\mathbf{x}_n) = y_{\boldsymbol{\pi}_n}$. So,

$$\mathbb{P}[f_{\boldsymbol{\pi}}(\mathbf{x}_n) = y_m] = \frac{1}{N},$$

for $m = 1, \ldots, N$, independent of the particular $\mathbf{x}_n$. We define $E^{\boldsymbol{\pi}}_{\text{out}}(h)$ with respect to this target function on the inputs $\mathbf{x}_1, \ldots, \mathbf{x}_N$.

(a) Define $E^{\boldsymbol{\pi}}_{\text{out}}(h) = \frac{1}{4}\,\mathbb{E}_{\mathbf{x},\boldsymbol{\pi}}\left[(h(\mathbf{x}) - f_{\boldsymbol{\pi}}(\mathbf{x}))^2\right]$, where expectation is with respect to the permutation input and target joint distribution. Show that

$$E^{\boldsymbol{\pi}}_{\text{out}}(h) = \frac{1}{4N}\sum_{n=1}^{N} \mathbb{E}_{\boldsymbol{\pi}}[(h(\mathbf{x}_n) - f_{\boldsymbol{\pi}}(\mathbf{x}_n))^2].$$

(b) Show that

$$E^{\boldsymbol{\pi}}_{\text{out}}(h) = \frac{s_y^2}{4} + \frac{1}{4N}\sum_{n=1}^{N}(h(\mathbf{x}_n) - \bar{y})^2,$$

where $\bar{y} = \frac{1}{N}\sum_n y_n$ and $s_y^2 = \frac{1}{N}\sum_n (y_n - \bar{y})^2$ are the mean and variance of the target values.

(c) In-sample error minimization on $\mathcal{D}_{\boldsymbol{\pi}}$ yields $g_{\boldsymbol{\pi}}$. What is $E^{\boldsymbol{\pi}}_{\text{in}}(g_{\boldsymbol{\pi}})$?

(d) The permutation optimism penalty is $E^{\boldsymbol{\pi}}_{\text{out}}(g_{\boldsymbol{\pi}}) - E^{\boldsymbol{\pi}}_{\text{in}}(g_{\boldsymbol{\pi}})$. Show:

$$\text{permutation optimism penalty} = \frac{1}{2N}\sum_{n=1}^{N}(y_{\boldsymbol{\pi}_n} - \bar{y})g_{(\boldsymbol{\pi})}(\mathbf{x}_n). \qquad (9.7)$$

(e) Show that the permutation optimism penalty is proportional to the correlation between the randomly permuted targets $y_{\boldsymbol{\pi}_n}$ and the learned function $g_{\boldsymbol{\pi}}(\mathbf{x}_n)$.

Problem 9.13 Repeat Problem 9.12, but , instead of defining the target distribution for the random problem using a random permutation (sampling the targets without replacement), use sampling of the targets with replacement (the Bootstrap distribution).
[Hint: Almost everything stays the same.]

Problem 9.14 In this problem you will investigate the Rademacher optimism penalty for the perceptron in one dimension, $h(x) = \text{sign}(x - w_0)$.

(a) Write a program to estimate the Rademacher optimism penalty:
  (i) Generate inputs $x_1, \ldots, x_N$ and random targets $r_1, \ldots, r_N$.
  (ii) Find the perceptron $g_{\mathbf{r}}$ with minimum in-sample error $E'_{\text{in}}(g_{\mathbf{r}})$.
  (iii) Compute the optimism penalty as $\frac{1}{2} - E'_{\text{in}}(g_{\mathbf{r}})$.

  Run your program for $N = 1, 2, \ldots, 10^3$ and plot the penalty versus N.

(b) Repeat part (a) 10,000 times to compute the average Rademacher penalty and give a plot of the penalty versus N.

(c) On the same plot show the function $1/\sqrt{N}$; how does the Rademacher penalty compare to $1/\sqrt{N}$? What is the VC-penalty for this learning model and how does it compare with the Rademacher penalty?

Problem 9.15 The Rademacher optimism penalty is

$$E'_{\text{out}}(g_{\mathbf{r}}) - E'_{\text{in}}(g_{\mathbf{r}}) = \frac{1}{2} - E'_{\text{in}}(g_{\mathbf{r}}).$$

Let $\mathcal{H}$ have growth function $m_{\mathcal{H}}(N)$. Show that, with probability at least $1-\delta$,

$$\text{Rademacher optimism penalty} \le \sqrt{\frac{1}{2N}\ln\frac{2m_{\mathcal{H}}(N)}{\delta}}.$$

[Hint: For a single hypothesis gives $\mathbb{P}[|E_{\text{out}}(h) - E_{\text{in}}(h)| > \epsilon] \le 2e^{-2N\epsilon^2}$*.]*

Problem 9.16 *[Hard]* **Rademacher Generalization Bound.** The Rademacher optimism penalty bounds the test error for binary classification (as does the VC-bound). This problem guides you through the proof.

We will need McDiarmid's inequality (a generalization of the Hoeffding bound):

Lemma 9.1 (McDiarmid, 1989). *Let $X_i \in A_i$ be independent random variables, and Q a function, $Q : \prod_i A_i \mapsto \mathbb{R}$, satisfying*

$$\sup_{\substack{x\in\prod_i A_i\\ z\in A_j}} |Q(\mathbf{x}) - q(x_1,\ldots,x_{j-1},z,x_{j+1},\ldots,x_n)| \le c_j,$$

for $j = 1,\ldots,n$. Then, $t > 0$,

$$\mathbb{P}[Q(X_1,\ldots,X_n) - \mathbb{E}[Q(X_1,\ldots,X_n)] \ge t] \le \exp\left(-\frac{2t^2}{\sum_{i=1}^n c_i^2}\right).$$

Corollary 9.2. *With probability at least $1-\delta$,*

$$\Big|Q(X_1,\ldots,X_n) - \mathbb{E}[Q(X_1,\ldots,X_n)]\Big| \le \sqrt{\frac{1}{2}\sum_{i=1}^n c_i^2 \ln\frac{2}{\delta}}.$$

(a) Assume that the hypothesis set is symmetric ($h \in \mathcal{H}$ implies $-h \in \mathcal{H}$). Prove that with probability at least $1-\delta$,

$$E_{\text{out}}(g) \le E_{\text{in}}(g) + \mathbb{E}_{r,\mathcal{D}}\left[\max_{h\in\mathcal{H}} \frac{1}{N}\sum_{n=1}^N r_n h(\mathbf{x}_n)\right] + \sqrt{\frac{1}{2N}\log\frac{2}{\delta}}. \quad (9.8)$$

To do this, show the following steps.

(i) $E_{\text{out}}(g) \le E_{\text{in}}(g) + \max\limits_{h\in\mathcal{H}}\{E_{\text{out}}(h) - E_{\text{in}}(h)\}$.

(ii) $\max\limits_{h\in\mathcal{H}}\{E_{\text{out}}(h) - E_{\text{in}}(h)\} = \frac{1}{2}\max\limits_{h\in\mathcal{H}}\left\{\frac{1}{N}\sum_{n=1}^N y_n h(\mathbf{x}_n) - \mathbb{E}_{\mathbf{x}}[f(\mathbf{x})h(\mathbf{x})]\right\}$.

(iii) Show that $\frac{1}{2}\max\limits_{h\in\mathcal{H}}\left\{\frac{1}{N}\sum_{n=1}^N y_n h(\mathbf{x}_n) - \mathbb{E}_{\mathbf{x}}[f(\mathbf{x})h(\mathbf{x})]\right\}$ is upper bounded with probability at least $1-\delta$ by

$$\tfrac{1}{2}\,\mathbb{E}_{\mathcal{D}}\left[\max_{h\in\mathcal{H}}\left\{\tfrac{1}{N}\textstyle\sum_{n=1}^N y_n h(\mathbf{x}_n) - \mathbb{E}_{\mathbf{x}}[f(\mathbf{x})h(\mathbf{x})]\right\}\right] + \sqrt{\tfrac{1}{2N}\log\tfrac{2}{\delta}}$$

[Hint: Use McDairmid's inequality with

$$Q(\mathbf{x}_1,\ldots,\mathbf{x}_N,y_1,\ldots,y_N)=\max_{h\in\mathcal{H}}\left\{\tfrac{1}{N}\textstyle\sum_{n=1}^N y_nh(\mathbf{x}_n)-\mathbb{E}_{\mathbf{x}}[f(\mathbf{x})h(\mathbf{x})]\right\}.$$

Show that perturbing data point $(\mathbf{x}_n,y_n)$ changes Q by at most $\frac{2}{N}$.]

(iv) Let $\mathcal{D}'=(\mathbf{x}'_1,y'_1),\ldots,(\mathbf{x}'_N,y'_N)$ be an independent (ghost) data set. Show that

$$\mathbb{E}_{\mathbf{x}}[f(\mathbf{x})h(\mathbf{x})]=\tfrac{1}{N}\textstyle\sum_{n=1}^N\mathbb{E}_{\mathcal{D}'}[y'_nh(\mathbf{x}'_n)].$$

Hence, show that $\max\limits_{h\in\mathcal{H}}\left\{\frac{1}{N}\sum_{n=1}^N y_nh(\mathbf{x}_n)-\mathbb{E}_{\mathbf{x}}[f(\mathbf{x})h(\mathbf{x})]\right\}$ equals

$$\max_{h\in\mathcal{H}}\left\{\mathbb{E}_{\mathcal{D}'}\left[\tfrac{1}{N}\textstyle\sum_{n=1}^N(y_nh(\mathbf{x}_n)-y'_nh(\mathbf{x}'_n))\right]\right\}.$$

By convexity, $\max_h\{\mathbb{E}[\cdot]\}\le\mathbb{E}[\max_h\{\cdot\}]$, hence show that

$$\begin{aligned}&\max_{h\in\mathcal{H}}\left\{\tfrac{1}{N}\textstyle\sum_{n=1}^N y_nh(\mathbf{x}_n)-\mathbb{E}_{\mathbf{x}}[f(\mathbf{x})h(\mathbf{x})]\right\}\\ \le\ &\mathbb{E}_{\mathcal{D}'}\left[\max_{h\in\mathcal{H}}\tfrac{1}{N}\textstyle\sum_{n=1}^N(y_nh(\mathbf{x}_n)-y'_nh(\mathbf{x}'_n))\right].\end{aligned}$$

Conclude that

$$E_{\text{out}}(g)\le E_{\text{in}}(g)+\tfrac{1}{2}\mathbb{E}_{\mathcal{D},\mathcal{D}'}\left[\max_{h\in\mathcal{H}}\frac{1}{N}\sum_{n=1}^N(y_nh(\mathbf{x}_n)-y'_nh(\mathbf{x}'_n))\right]+\sqrt{\tfrac{1}{2N}\log\tfrac{2}{\delta}}.$$

The remainder of the proof is to bound the second term on the RHS.

(v) Let $r_1,\ldots,r_N$ be *arbitrary* ±1 Rademacher variables. Show that

$$\begin{aligned}&\mathbb{E}_{\mathcal{D},\mathcal{D}'}\left[\max_{h\in\mathcal{H}}\left\{\tfrac{1}{N}\textstyle\sum_{n=1}^N(y_nh(\mathbf{x}_n)-y'_nh(\mathbf{x}'_n))\right\}\right]\\ =\ &\mathbb{E}_{\mathcal{D},\mathcal{D}'}\left[\max_{h\in\mathcal{H}}\left\{\tfrac{1}{N}\textstyle\sum_{n=1}^N r_n(y_nh(\mathbf{x}_n)-y'_nh(\mathbf{x}'_n))\right\}\right]\\ \le\ &2\,\mathbb{E}_{\mathcal{D}}\left[\max_{h\in\mathcal{H}}\left\{\tfrac{1}{N}\textstyle\sum_{n=1}^N r_ny_nh(\mathbf{x}_n)\right\}\right].\end{aligned}$$

[Hint: Argue that $r_n=-1$ effectively switches $\mathbf{x}_n$ with $\mathbf{x}'_n$ which is just a relabeling of variables in the expectation over $\mathbf{x}_n,\mathbf{x}'_n$. For the second step, use $\max\{A-B\}\le\max|A|+\max|B|$ and the fact that $\mathcal{H}$ is symmetric.]

(vi) Since the bound in (v) holds for any $\mathbf{r}$, we can take the expectation over independent r_n with $\mathbb{P}[r_n=+1]=\frac{1}{2}$. Hence, show that

$$\begin{aligned}&\mathbb{E}_{\mathcal{D},\mathcal{D}'}\left[\max_{h\in\mathcal{H}}\left\{\tfrac{1}{N}\textstyle\sum_{n=1}^N(y_nh(\mathbf{x}_n)-y'_nh(\mathbf{x}'_n))\right\}\right]\\ \le\ &2\,\mathbb{E}_{\mathbf{r},\mathcal{D}}\left[\max_{h\in\mathcal{H}}\left\{\tfrac{1}{N}\textstyle\sum_{n=1}^N r_nh(\mathbf{x}_n)\right\}\right],\end{aligned}$$

and obtain (9.8). *[Hint: what is the distribution of r_ny_n?]*

(b) In part (a) we obtained a generalization bound in terms of twice the *expected* Rademacher optimism penalty. To prove Theorem 9.6, show that this expectation can be well approximated by a single realization.

(i) Let $Q(r_1,\ldots,r_N,\mathbf{x}_1,\ldots,\mathbf{x}_n) = \max\limits_{h\in\mathcal{H}}\left\{\frac{1}{N}\sum_{n=1}^{N} r_n h(\mathbf{x}_n)\right\}$. Show that if you change an input of Q, its value changes by at most $\frac{2}{N}$.

(ii) Show that with probability at least $1-\delta$,

$$\mathbb{E}_{\mathbf{r},\mathcal{D}}\left[\max_{h\in\mathcal{H}}\left\{\tfrac{1}{N}\sum_{n=1}^{N} r_n h(\mathbf{x}_n)\right\}\right] \le \max_{h\in\mathcal{H}}\left\{\tfrac{1}{N}\sum_{n=1}^{N} r_n h(\mathbf{x}_n)\right\}+\sqrt{\tfrac{2}{N}\ln\tfrac{2}{\delta}}.$$

(iii) Apply the union bound to show that with probability at least $1-\delta$,

$$E_{\text{out}}(g) \le E_{\text{in}}(g) + \max_{h\in\mathcal{H}}\left\{\tfrac{1}{N}\sum_{n=1}^{N} r_n h(\mathbf{x}_n)\right\} + \sqrt{\tfrac{9}{2N}\ln\tfrac{4}{\delta}}.$$

Problem 9.17 *[Hard]* **Permutation Generalization Bound.** Prove that with probability at least $1-\delta$,

$$E_{\text{out}}(g) \le E_{\text{in}}(g) + \mathbb{E}_{\boldsymbol{\pi}}\left[\max\nolimits_{h\in\mathcal{H}} \tfrac{1}{N}\sum_{n=1}^{N} y_n^{(\boldsymbol{\pi})} h(\mathbf{x}_n)\right] + O\left(\sqrt{\tfrac{1}{N}\log\tfrac{1}{\delta}}\right).$$

The second term is similar to the permutation optimism penalty, differing by $\bar{y}\,\mathbb{E}_{\boldsymbol{\pi}}\left[\bar{g}_{\boldsymbol{\pi}}\right]$, which is zero for balanced data.

[Hint: Up to introducing the r_n, you can follow the proof in Problem 9.16; now pick a distribution for $\mathbf{r}$ to mimic permutations. For some helpful tips, see "A Permutation Approach to Validation," Magdon-Ismail, Mertsalov, 2010.]

Problem 9.18 **Permutation Penalty for Linear Models.** For linear models, the predictions on the data are $\hat{\mathbf{y}}^{(\boldsymbol{\pi})} = \mathrm{H}\mathbf{y}^{(\boldsymbol{\pi})}$, H is the hat matrix, $\mathrm{H}(\lambda) = \mathrm{X}(\mathrm{X}^{\mathrm{T}}\mathrm{X}+\lambda\mathrm{I})^{-1}\mathrm{X}^{\mathrm{T}}$, which is independent of $\boldsymbol{\pi}$. For regression, the permutation optimism penalty from (9.7) is $\frac{2}{N}\sum_{n=1}^{N}(y_{\boldsymbol{\pi}_n}-\bar{y})g_{(\boldsymbol{\pi})}(\mathbf{x}_n)$. (we do not divide the squared error by 4 for regression).

(a) Show that for a single permutation, permutation penalty is:

$$\frac{2}{N}\sum_{m,n=1}^{N} \mathrm{H}_{mn}(y_{\boldsymbol{\pi}_m}y_{\boldsymbol{\pi}_n} - \bar{y}y_{\boldsymbol{\pi}_n}).$$

(b) Show that: $\mathbb{E}_{\boldsymbol{\pi}}[y_{\boldsymbol{\pi}_n}] = \bar{y}$, and $\mathbb{E}_{\boldsymbol{\pi}}[y_{\boldsymbol{\pi}_m}y_{\boldsymbol{\pi}_n}] = \begin{cases}\bar{y}^2+s_y^2 & m=n,\\ \bar{y}^2-\frac{1}{N-1}s_y^2 & m\ne n.\end{cases}$

($\bar{y}$ and s_y^2 are defined in Problem 9.12(b).)

(c) Take the expectation of the penalty in (a) and prove Equation (9.6):

$$\text{permutation optimism penalty} = \frac{2\hat{\sigma}_y^2}{N}\left(\text{trace}(\mathrm{S}) - \frac{\mathbf{1}^{\mathrm{T}}\mathrm{S}\mathbf{1}}{N}\right),$$

where $\hat{\sigma}_y^2 = \frac{N}{N-1}s_y^2$ is the unbiased estimate of the target variance.

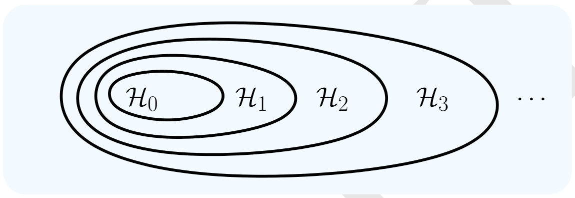
Problem 9.19 Repeat Problem 9.18 for the Bootstrap optimism penalty and show that

$$\text{Bootstrap optimism penalty} = \frac{2s_y^2}{N}\text{trace}(\mathrm{H}).$$

Compare this to the permutation optimism penalty in Problem 9.18.

[Hint: How does Problem 9.18(b) change for the Bootstrap setting?]

Problem 9.20 This is a repeat of Problem 5.2. Structural Risk Minimization (SRM) is a useful framework for model selection that is related to Occam's Razor. Define a *structure* – a nested sequence of hypothesis sets:



The SRM framework picks a hypothesis from each $\mathcal{H}_i$ by minimizing E_{in}. That is, $g_i = \underset{h\in\mathcal{H}_i}{\text{argmin}}\, E_{\text{in}}(h)$. Then, the framework selects the final hypothesis by minimizing E_{in} *and* the model complexity penalty Ω. That is, $g^* = \underset{i=1,2,\cdots}{\text{argmin}}(E_{\text{in}}(g_i) + \Omega(\mathcal{H}_i))$. Note that $\Omega(\mathcal{H}_i)$ should be non-decreasing in i because of the nested structure.

(a) Show that the in-sample error $E_{\text{in}}(g_i)$ is non-increasing in i.

(b) Assume that the framework finds $g^* \in \mathcal{H}_i$ with probability p_i. How does p_i relate to the complexity of the target function?

(c) Argue that the p_i's are unknown but $p_0 \le p_1 \le p_2 \le \cdots \le 1$.

(d) Suppose $g^* = g_i$. Show that

$$\mathbb{P}\left[|E_{\text{in}}(g_i) - E_{\text{out}}(g_i)| > \epsilon \mid g^* = g_i\right] \le \frac{1}{p_i}\cdot 4m_{\mathcal{H}_i}(2N)e^{-\epsilon^2 N/8}.$$

Here, the conditioning is on selecting g_i as the final hypothesis by SRM. *[Hint: Use the Bayes theorem to decompose the probability and then apply the VC bound on one of the terms]*

You may interpret this result as follows: if you use SRM and end up with g_i, then the generalization bound is a factor $\frac{1}{p_i}$ worse than the bound you would have gotten had you simply started with $\mathcal{H}_i$.

Problem 9.21 Cross-Validation Leverage for Linear Regression.] In this problem, compute an expression for the leverage defined in Equation (9.4) for linear regression with weight decay regularization. We will use the same notation from Problem 4.26, so you may want to review that problem and some of the tools developed there.

To simulate leaving out the data point $(\mathbf{z}_m, y_m)$, set the mth row of Z and the mth entry of $\mathbf{y}$ to zero, to get data matrix $\mathrm{Z}^{(m)}$ and target vector $\mathbf{y}^{(m)}$, so $\mathcal{D}_m = (\mathrm{Z}^{(m)}, \mathbf{y}^{(m)})$. We need to compute the cross validation error for this data set $E_{\text{cv}}(\mathcal{D}_m)$. Let $\hat{\mathrm{H}}^{(m)}$ be the hat matrix you get from doing linear regression with the data $\mathrm{Z}^{(m)}, \mathbf{y}^{(m)}$.

(a) Show that $E_{\text{cv}}(\mathcal{D}_m) = \dfrac{1}{N-1}\displaystyle\sum_{n\neq m}\left(\frac{\hat{y}_n^{(m)} - y_n}{1 - \mathrm{H}_{nn}^{(m)}}\right)^2$.

(b) Use the techniques from Problem 4.26 to show that

$$\mathrm{H}_{nk}^{(m)} = \mathrm{H}_{nk} + \frac{\mathrm{H}_{mn}\mathrm{H}_{mk}}{1 - \mathrm{H}_{mm}}.$$

(c) Similarly, show that

$$\hat{y}_n^{(m)} = \hat{y}_n + \left(\frac{\hat{y}_m - y_m}{1 - \mathrm{H}_{mm}}\right)\mathrm{H}_{mn}.$$

[Hint: Use part (c) of Problem 4.26.]

(d) Show that $E_{\text{cv}}(\mathcal{D}_m)$ is given by

$$\frac{1}{N-1}\sum_{n=1}^{N}\left(\frac{\hat{y}_n - y_n + \left(\frac{\hat{y}_m - y_m}{1-\mathrm{H}_{mm}}\right)\mathrm{H}_{mn}}{1 - \mathrm{H}_{nn} - \frac{\mathrm{H}_{mn}^2}{1-\mathrm{H}_{mm}}}\right)^2 + \frac{1}{N-1}\left(\frac{\hat{y}_m - y_m}{1 - 2\mathrm{H}_{mm}}\right)^2.$$

(e) Give an expression for the leverage ℓ_m. What is the running time to compute the leverages $\ell_1, \ldots, \ell_N$.

Problem 9.22 The data set for Example 9.3 is

$$\mathrm{X} = \begin{bmatrix} 1 & 0.51291 \\ 1 & 0.46048 \\ 1 & 0.3504 \\ 1 & 0.095046 \\ 1 & 0.43367 \\ 1 & 0.70924 \\ 1 & 0.11597 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 0.36542 \\ 0.22156 \\ 0.15263 \\ 0.10355 \\ 0.10015 \\ 0.26713 \\ 2.3095 \end{bmatrix}.$$

Implement the algorithm from Problem 9.21 to compute the leverages for all the data points as you vary the regularization parameter λ.

Give a plot of the leverage of the last data point as a function of λ. Explain the dependence you find in your plot.

Problem 9.23 For a sample of 500 digits data (classifying "1" versus "not 1"), use linear regression to compute the leverage of the data points (even though the problem is a classification problem). You can use the algorithm in Problem 9.21.

Give a similar plot to Figure 9.5 where you highlight in a black box all the data points with the top-10 largest (positive) leverages.

Give some intuition for which points are highlighted.

Problem 9.24 **The Jackknife Estimate.** The jackknife is a general statistical technique used to reduce the bias of an estimator, based on the assumption that the estimator is asymptotically (as N increases) unbiased. Let $\mathcal{Z} = \mathbf{z}_1, \ldots, \mathbf{z}_N$ be a sample set. In the case of learning, the sample set is the data, so $\mathbf{z}_n = (\mathbf{x}_n, y_n)$. We wish to estimate a quantity t using an estimator $\hat{t}(\mathcal{Z})$ (some function of the sample $\mathcal{Z}$). Assume that $\hat{t}(\mathcal{Z})$ is asymptotically unbiased, satisfying

$$\mathbb{E}_{\mathcal{Z}}[\hat{t}(\mathcal{Z})] = t + \frac{a_1}{N} + \frac{a_2}{N^2} + \cdots.$$

The bias is $O\left(\frac{1}{N}\right)$. Let $\mathcal{Z}_n = \mathbf{z}_1, \ldots, \mathbf{z}_{n-1}, \mathbf{z}_n, \mathbf{z}_{n+1}, \ldots, \mathbf{z}_N$ be the leave one out sample sets (similar to cross validation), and consider the estimates using the leave one out samples $\hat{t}_n = \hat{t}(\mathcal{Z}_n)$.

(a) Argue that $\mathbb{E}_{\mathcal{Z}}[\hat{t}_n] = t + \dfrac{a_1}{N-1} + \dfrac{a_2}{(N-1)^2} + \cdots$.

(b) Define $\hat{\tau}_n = N\hat{t}(\mathcal{Z}) - (N-1)\hat{t}_n$. Show that $\mathbb{E}_{\mathcal{Z}}[\hat{\tau}_n] = t - \frac{a_2}{N(N-1)} + O(\frac{1}{N^2})$. ($\hat{\tau}_n$ has an asymptotically asymptotically smaller bias than $t(\mathcal{Z})$.) The $\hat{\tau}_n$ are called pseudo-values because they have the "correct" expectation. A natural improvement is to take the average of the pseudo-values, and this is the jackknife estimate: $\hat{t}_J(\mathcal{Z}) = \frac{1}{N}\sum_{n=1}^{N} \hat{\tau}_n = N\hat{t}(\mathcal{Z}) - \frac{N-1}{N}\sum_{n=1}^{N} \hat{t}(\mathcal{Z}_n)$.

(c) *Applying the Jackknife to variance estimation.* Suppose that the sample is a bunch of independent random values $x_1, \ldots, x_N$ from a distribution whose variance σ^2 we wish to estimate. We suspect that the sample variance, $s^2 = \frac{1}{N}\sum_{n=1}^{N} x_n^2 - \frac{1}{N^2}\left(\sum_{n=1}^{N} x_n\right)^2$, should be a good estimator, i.e., it has the assumed form for the bias (it does). Let s_n^2 be the sample variances on the leave one out samples. Show that

$$s_n^2 = \tfrac{1}{N-1}\left(\sum_{m=1}^{N} x_m^2 - x_n^2\right) - \tfrac{1}{(N-1)^2}\left(\sum_{m=1}^{N} x_m - x_n\right)^2.$$

Hence show that jackknife estimate $Ns^2 - \frac{N-1}{N}\sum_n s_n^2$ is $s_J^2 = \frac{N}{N-1}s^2$, which is the well known unbiased estimator of the variance. In this particular case, the jackknife has completely removed the bias (automatically).

(d) What happens to the jackknife if $t(\mathcal{Z})$ has an asymptotic bias?

(e) If the leading order term in the bias was $\frac{1}{N^2}$, does the jackknife estimate have a better or worse bias (in magnitude)?

Problem 9.25 The Jackknife for Validation. (see also the previous problem) If E_{in} is an asymptotically unbiased estimate of E_{out}, we may use the jackknife to reduce this bias and get a better estimate. The sample is the data. We want to estimate the expected out-of-sample error when learning from N examples, $\mathcal{E}_{\textsf{out}}(N)$. We estimate $\mathcal{E}_{\textsf{out}}(N)$ by the in-sample error $E_{\text{in}}(g^{(\mathcal{D})})$.

(a) What kind of bias (+ve or -ve) will E_{in} have. When do you expect it to be asymptotically unbiased?

(b) Assume that the bias has the required form: $\mathbb{E}_{\mathcal{D}}[E_{\text{in}}(g^{(\mathcal{D})})] = \mathcal{E}_{\textsf{out}}(N) + \frac{a_1}{N} + \frac{a_2}{N^2} + \cdots$. Now consider one of the leave one out data sets $\mathcal{D}_n$, which would produce the estimates $E_{\text{in}}(g^{(\mathcal{D}_n)})$. Show that:

i. the pseudo-values are $NE_{\text{in}}(g^{(\mathcal{D})}) - (N-1)E_{\text{in}}(g^{(\mathcal{D}_n)})$;

ii. the jackknife estimate is: $E_J = NE_{\text{in}}(g^{(\mathcal{D})}) - \frac{N-1}{N}\sum\limits_{n=1}^{N} E_{\text{in}}(g^{(\mathcal{D}_n)})$.

(c) Argue that $\mathbb{E}_{\mathcal{D}}[E_{\text{in}}(g^{(\mathcal{D}_n)})] = \mathcal{E}_{\textsf{out}}(N-1) + \frac{a_1}{N-1} + \frac{a_2}{(N-1)^2} + \cdots$, and hence show that the expectation of the jackknife estimate is given by

$$\mathbb{E}_{\mathcal{D}}[E_J] = \mathcal{E}_{\textsf{out}}(N) + (N-1)(\mathcal{E}_{\textsf{out}}(N) - \mathcal{E}_{\textsf{out}}(N-1)) + O\left(\frac{1}{N^2}\right).$$

(d) If the learning curve converges, having a form $\mathcal{E}_{\textsf{out}}(N) = E + \frac{b_1}{N} + \frac{b_2}{N^2} + \cdots$, then show that $\mathbb{E}_{\mathcal{D}}[E_J] = \mathcal{E}_{\textsf{out}}(N) + \frac{b_1}{N} + O\left(\frac{1}{N^2}\right)$.

The jackknife replaced the term $\frac{a_1}{N}$ in the bias by $\frac{b_1}{N}$. (In a similar vein to cross validation, the jackknife replaces the bias of the in-sample estimate with the bias in the learning curve.) When will the jackknife be helpful?

Problem 9.26 The Jackknife Estimate for Linear Models. The jackknife validation estimate can be computed analytically for linear models. Define the matrix $\mathrm{H}^{(\delta)} = \mathrm{H}^2 - \mathrm{H}$, and let $\mathbf{y}^{(\delta)} = \mathrm{H}^{(\delta)}\mathbf{y}$.

(a) Show that the jackknife estimate is given by

$$E_J = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{(\hat{y}_n - y_n)^2}{1-\mathrm{H}_{nn}} - \frac{2y_n^{(\delta)}(\hat{y}_n - y_n)}{1-\mathrm{H}_{nn}} - \frac{\mathrm{H}_{nn}^{(\delta)}(\hat{y}_n - y_n)^2}{(1-\mathrm{H}_{nn})^2}\right). \quad (9.9)$$

[Hint: you may find some of the formulas used in deriving the cross validation estimate for linear models useful from Problem 4.26.]

(b) When $\lambda = 0$, what is E_J? Compare to E_{cv} in (4.13). Show that $E_{\text{in}} \le E_J \le E_{\text{cv}}$. *[Hint: H is a projection matrix, so $\mathrm{H}^2 = \mathrm{H}$ when $\lambda = 0$; also, $\mathrm{H}_{nn} = \mathbf{x}_n^{\mathrm{T}}(\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathbf{x}_n$, so show that $0 \le \mathrm{H}_{nn} \le 1$ (show that for any positive definite (invertible) matrix A, $\mathbf{x}_n^{\mathrm{T}}(\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}} + A)^{-1}\mathbf{x}_n \le 1$).]*

Problem 9.27 **Feature Selection.** Let $x \in \mathbb{R}^d$ and let $\mathcal{H}$ be the perceptron model $\text{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x})$. The *zero norm*, $\|\mathbf{w}\|_0$, is the number of non-zero elements in $\mathbf{w}$. Let $\mathcal{H}_k = \{\mathbf{w} : \|\mathbf{w}\|_0 \leq k\}$. The hypothesis set $\mathcal{H}_k$ contains the classifiers which use only k of the dimensions (or *features*) to classify the input. Model selection among the $\mathcal{H}_k$ corresponds to selecting the optimal number of features. Picking a particular hypothesis in $\mathcal{H}_k$ corresponds to picking the best set of k features.

(a) Show that order selection is a special case of feature selection.

(b) Show that $d_{\text{VC}}(\mathcal{H}_k) \leq \max(d+1, (2k+1)\log_2 d) = O(k \log d)$, which for $k = o(d)$ is an improvement over the trivial bound of $d+1$.
[Hint: Show that $d_{\text{VC}} \leq M$ for any M which satisfies $2^M > M^{k+1}\binom{d}{k}$; to show this, remember that $\mathcal{H}_k$ is the union of smaller models (how many? what is their VC-dimension?), and use the relationship between the VC-dimension of a model and the maximum number of dichotomies it can implement on any M points. Now use the fact that $\binom{d}{k} \leq \left(\frac{ed}{k}\right)^k$]

Problem 9.28 **Forward and Backward Subset Selection (FSS and BSS)** For selecting k features, we fix $\mathcal{H}_k$, and ask for the hypothesis which minimizes E_{in} (typically $k \ll d$). There is no known efficient algorithm to solve this problem (it is a NP-hard problem to even get close to the minimum E_{in}). FSS and BSS are greedy heuristics to solve the problem.

FSS: Start with $k = 1$ and find the best single feature dimension i_1 yielding minimum E_{in}. Now fix this feature dimension and find the next feature dimension i_2 which when coupled with the feature dimension i_1 yields minimum E_{in}. Continue in this way, adding one feature at a time, until you have k features.

BSS: Start with all d features and now remove one feature at a time until you are down to k features. Each time you remove the feature which results in minimum increase in E_{in}.

(a) Which of FSS and BSS do you expect to be more efficient. Suppose that to optimize with i features and N data points takes $O(iN)$ time. What is the asymptotic run time of FSS and BSS. For what values of k would you use FSS over BSS and vice versa. *[Hint: show that FSS takes $N\sum_{i=1}^{k} i(d+1-i)$ time and BSS takes $N\sum_{i=1}^{d-k}(d-i)(d+1-i)$.]*

(b) Show that $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_d$ are a structure. (See Problem 9.20.)

(c) Implement FSS. Use the bound on the VC-dimension given in the Problem 9.27 to perform SRM using the digits data to classify 1 (class $+1$) from $\{2, 3, 4, 5, 6, 7, 8, 9, 0\}$ (all in class -1).

(d) What are the first few most useful features? What is the optimal number of features (according to SRM)?