Chapter 3

Linear Models for Outlier Detection

"My nature is to be linear, and when I'm not, I feel really proud of myself." – Cynthia Weil

3.1 Introduction

The attributes in real data are usually highly correlated. Such dependencies provide the ability to predict attributes from one another. The notions of prediction and anomaly detection are intimately related. Outliers are, after all, values that deviate from expected (or *predicted*) values on the basis of a particular model. Linear models focus on the use of inter-attribute dependencies to achieve this goal. In the classical statistics literature, this process is referred to as *regression modeling*.

Regression modeling is a parametric form of correlation analysis. Some forms of correlation analysis attempt to predict dependent variables from other independent variables, whereas other forms summarize the entire data in the form of latent variables. An example of the latter is the method of *principal component analysis*. Both forms of modeling can be very useful in different scenarios of outlier analysis. The former is more useful for complex data types such as time-series (see Chapters 9 and 11), whereas the latter is more useful for the conventional multidimensional data type. A unified discussion of these two forms of linear modeling also lays the foundations for some of the discussions in later chapters.

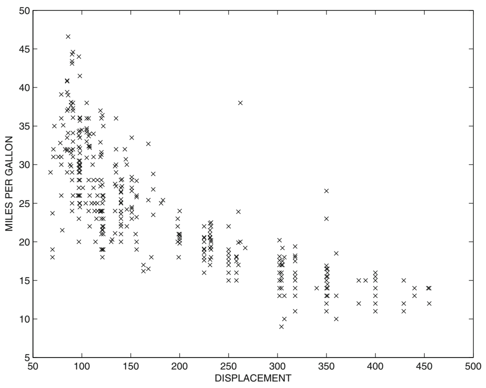
The main assumption in linear models is that the (normal) data is embedded in a lower-dimensional subspace. Data points that do not naturally fit this embedding model are, therefore, regarded as outliers. In the case of proximity-based methods, which will be discussed in the next chapter, the goal is to determine specific *regions of the space* in which outlier points behave very differently from other points. On the other hand, in linear methods, the goal is to find *lower-dimensional subspaces*, in which the outlier points behave very differently from other points. This can be viewed as an orthogonal point of view to clustering- or nearest-neighbor methods, which try to summarize the data *horizontally* (i.e., on the rows or data values), rather than *vertically* (i.e., on the columns or dimensions). As will be discussed in the chapter on high-dimensional outlier detection, it is, in principle, possible to combine these methods to obtain more general local subspace models, which can identify outliers by integrating the horizontal and vertical criteria in a holistic way.

The assumption of linear correlations is a critical leap of faith used by linear models. This may or may not be true for a particular data set, which will have a crucial impact on modeling effectiveness. In order to explain this point, we will use the *Autompg* and *Arrythmia* data sets from the *UCI Machine Learning Repository* [203]. The first data set contains features describing various car measurements and the corresponding mileage (mpg). The second data set contains features derived from electrocardiogram (ECG) readings of human patients.

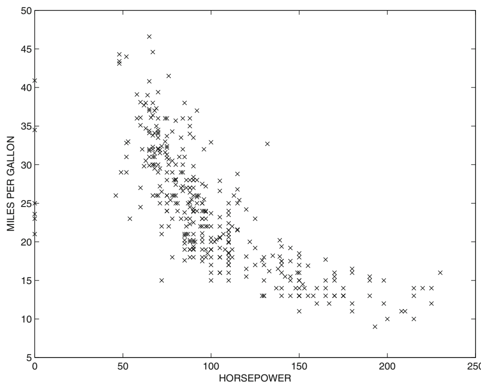
In Figures 3.1(a) and (b), the dependence of the *Miles per Gallon* attribute has been shown on each of the *displacement* and *horsepower* attributes, respectively, for the *Autompg* data set. It is evident that these attributes are highly correlated. Although a significant level of noise is also present in this particular data set, the linear dependence between the attributes is readily apparent. In fact, it can be shown for this data set, that with increasing dimensionality (by selecting more attributes from the data set), the data can be aligned along much lower-dimensional planes. This is also evident in the 3-dimensional plot of Figure 3.1(e). On the other hand, when various views along three of the measured dimensions of the *Arrythmia* data set (Figures 3.1(c), (d) and (f)) are examined, it is evident that the data separates out into two clusters, one of which is slightly larger than the other. Furthermore, it is rather hard to embed this type of data distribution into a lower-dimensional subspace. This data set is more suitable for proximity-based analysis, which will be presented in Chapter 4. The reason for introducing this example is to revisit the point made in the first chapter about the impact of the choices made during the crucial phase of selecting the correct model of normal data. In this particular case, it is evident that the linear model is more appropriate for data sets in which the data naturally aligns along lower-dimensional hyperplanes.

For unsupervised problems like outlier detection, it is hard to guarantee that a specific model of normal data will be effective. For example, in some data sets, different subsets of attributes may be suitable for different models. Such data sets are best addressed with the use of subspace methods discussed in Chapter 5, which can combine the power of row and column selection for outlier analysis. However, in many cases, simplified models such as linear models or proximity-based models are sufficient, without incurring the complexity of subspace methods. From a model-selection perspective, exploratory and visual analysis of the data is rather critical in the first phase of outlier detection in order to find out whether a particular data model is suitable for a particular data set. This is particularly true in the case of unsupervised problems like outlier detection in which ground-truth is not available in order to test the effectiveness of various models.

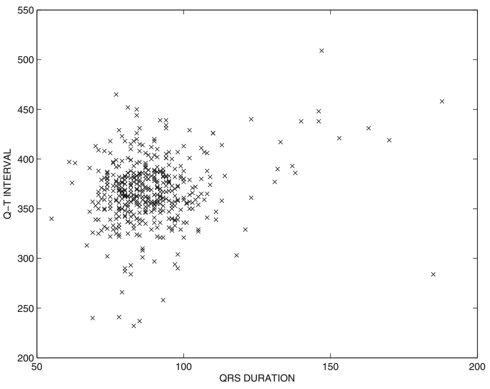
In this chapter, two main classes of linear models will be studied. The first class of models uses statistical regression modeling between dependent and independent variables in order to determine *specific types of dependencies* in the data. Such forms of modeling are more useful when some of the attributes are naturally predictive of others. For example, it is natural to predict the *last* value of a time-series based on a window of previous history of values in the time-series. In such cases, dependency-oriented regression modeling is leveraged by creating a derived data set of dependent and independent variables and quantifying deviations in the observed value of the dependent variable from its predicted value. Even for multidimensional data, we will show in section 7.7 of Chapter 7 that dependency-oriented models for regression can be used to decompose the unsupervised outlier detection problem into d different regression modeling problems for d-dimensional data. The second class of models uses principal component analysis to determine the lower-dimensional subspaces



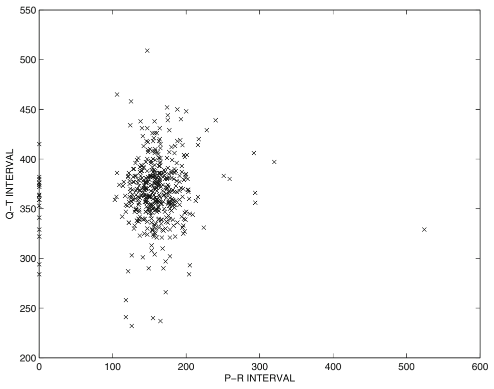
(a) View 1 (*Autompg*)



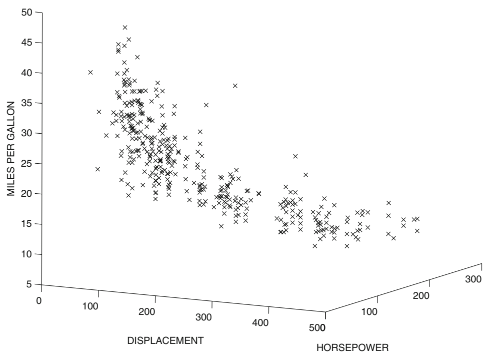
(b) View 2 (*Autompg*)



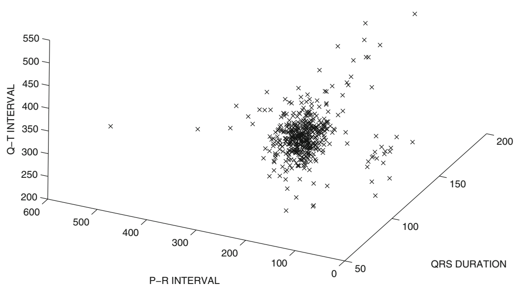
(c) View 1 (*Arrythmia*)



(d) View 2 (*Arrythmia*)



(e) 3-d View (*Autompg*)



(f) 3-d View (*Arrythmia*)

Figure 3.1: Effectiveness of linear assumption is data-set dependent

of projection. These models are useful for traditional multidimensional data because all attributes are treated in a homogeneous way with no distinction between dependent and independent variables. At a technical and mathematical level, both forms of modeling are quite similar, and use very similar methods in order to derive the optimal lower-dimensional representations. The main difference is in how the objective function of the two models is formulated. Some of the dependency-oriented models discussed in this chapter will also be useful in formulating outlier-detection models in the dependency-oriented data types discussed in later chapters.

It should be emphasized that regression-analysis is used extensively to detect anomalies in time-series data, and many of the basic techniques discussed in this chapter are applicable to that scenario as well. This techniques will be discussed in more detail in Chapter 9. However, since the time-series aspect of the problem is also based on dependencies of *temporally adjacent* data values, there are a number of subtle differences in the modeling process for multidimensional data and that for time-series data. Therefore, in this chapter, the much simpler case of multidimensional outlier analysis will be addressed. At the same time, the discussion will be general enough, so that the technical machinery required for applying regression analysis to the time-series scenario (Chapter 9) is introduced.

This chapter is organized as follows. In section 3.2, the basic linear regression models for outlier analysis will be introduced. In section 3.3, the principal component method for outlier analysis will be discussed. This can be considered an important special case of linear regression models, which is used frequently in outlier analysis, and it uses a similar optimization model. Therefore it is given a dedicated treatment in its own section. We discuss both the hard and soft versions of principal component analysis and show that the latter is equivalent to the Mahalanobis method discussed in the previous chapter. Furthermore, this technique can be easily extended to the nonlinear case with kernel methods. The related problem of one-class support-vector machines is discussed in section 3.4. A matrix-factorization view of linear models is discussed in section 3.5. Neural networks are introduced in section 3.6. Section 3.7 will study the limitations of linear models for outlier analysis. Section 3.8 contains the conclusions and summary.

3.2 Linear Regression Models

In linear regression, the observed values in the data are modeled using a linear system of equations. Specifically, the different dimensions in the data are related to one another using a set of linear equations, in which the coefficients need to be learned in a data-driven manner. Since the number of observed values is typically much larger than the dimensionality of the data, this system of equations is an *over-determined* one and cannot be solved exactly (i.e., zero error). Therefore, these models learn the coefficients that minimize the squared error of the deviations of data points from values predicted by the linear model. The exact choice of the error function determines whether a particular variable is treated specially (i.e., error of predicted variable value), or whether variables are treated homogeneously (i.e., error distance from estimated lower-dimensional plane). These different choices of the error function do *not* lead to the same model. In fact, the discussion in this chapter will show that the models can be qualitatively very different *especially in the presence of outliers.*

Regression analysis is generally considered an important application in its own right in the field of statistics. In classical instantiations of this application, it is desired to learn the value of a specific dependent variable from a set of independent variables. This is a common scenario in time-series analysis, which will be discussed in detail in Chapter 9.

Thus, a specific variable is treated *specially* from the other variables. Independent variables are also referred to as *explanatory variables*. This is a common theme with *contextual data types*, in which some attributes (e.g., time, spatial location, or adjacent series values) are treated as independent, whereas others (e.g., temperature or environmental measurements) are treated as dependent. Therefore, much of the framework in this section will also set the stage for the analysis of dependency-oriented data types in later chapters. However, for the straightforward multidimensional data type, all dimensions are treated in a homogeneous way, and the best-fitting linear relation among *all* the attributes is estimated. Therefore, there are subtle differences in the modeling (i.e., optimization formulation) of these two different settings.

Consider a domain such as temporal and spatial data, in which the attributes are partitioned into *contextual* and *behavioral* attributes. In such cases, a particular behavioral attribute value is often predicted as a linear function of the behavioral attributes in its *contextual* neighborhood in order to determine deviations from expected values. This is achieved by constructing a multidimensional data set from the temporal or spatial data in which a particular behaviorial attribute value (e.g., temperature at current time) is treated as the dependent variable, and its contextual neighborhood behaviorial values (e.g., temperatures in previous window) are treated as the independent variables. Therefore, the importance of predicting the dependent variable is paramount in estimating deviations and thereby quantifying outlier scores. In such cases, outliers are defined on the basis of the error of predicting the dependent variable, and anomalies within the interrelationships of independent variables are considered less important. Therefore, the focus of the optimization process is on minimizing the predicted error of the *dependent* variable(s) in order to create the model of normal data. Deviations from this model are flagged as outliers.

The special case of regression analysis with dependent variables will be studied first. The discussion of this case also sets the stage for a more detailed discussion for the cases of time-series data in Chapter 9 and spatial data in Chapter 11. Furthermore, the use of this technique for decomposition of the multidimensional outlier detection problem into a set of regression modeling problems will be studied in section 7.7 of Chapter 7. The identification of outliers in such cases is also very useful for *noise reduction* in regression modeling [467], which is an important problem in its own right.

In a later section, we will introduce a more general version of linear models in which no distinction is made between dependent and independent variables. The basic idea here is that the (normal) data are assumed to lie on a lower-dimensional hyperplane in the feature space. The normalized deviation of a point from this lower-dimensional hyperplane (in a direction perpendicular to the hyperplane) is used to compute the outlier score. As we will show later, these simple linear models can be greatly enriched with the use of *data transformations* that are able to map these linear models to more complex nonlinear models. Interestingly, certain types of transformations even allow the use of these linear models for outlier detection in more complex data types such as time-series data, discrete sequence data, and graph data. Furthermore, many other linear and nonlinear models such as one-class support-vector machines and neural networks can be viewed as variants of these models.

3.2.1 Modeling with Dependent Variables

A variable y can be modeled as a linear function of d dependent variables (or dimensions) as follows:

$$y = \sum_{i=1}^{d} w_i \cdot x_i + w_{d+1}$$

The variable y is the response variable or the dependent variable, and the variables $x_1 \ldots x_d$ are the independent or the explanatory variables. The coefficients $w_1 \ldots w_{d+1}$ need to be learned from the training data. The data set might contain N different instances denoted by $\overline{X_1} \ldots \overline{X_N}$. The d dimensions in the jth point $\overline{X_j}$ are denoted by $(x_{j1} \ldots x_{jd})$. The jth response variable is denoted by y_j. These N instance-pairs $(\overline{X_j}, y_j)$ provide examples of how the dependent variable is related to the d independent variables with a linear function. The parameters of the linear function are learned with an optimization model that minimizes the aggregate squared error of predicting the dependent variable. The key assumption here is that the dependent variable is central to the application at hand, and therefore errors are defined only with respect to this variable. Outliers are defined as those data points in which the linear function learned from the N instances provides an unusually large error. Therefore, the jth instance of the response variable is related to the explanatory variables as follows:

$$y_j = \sum_{i=1}^{d} w_i \cdot x_{ji} + w_{d+1} + \epsilon_j \quad \forall j \in \{1, \ldots, N\} \tag{3.1}$$

Here, ϵ_j represents the error in modeling the jth instance, and it provides the *outlier score* of the pair $(\overline{X_j}, y_j)$. In *least-squares regression*, the goal is to determine the regression coefficients $w_1 \ldots w_{d+1}$ that minimize the error $\sum_{j=1}^{N} \epsilon_j^2$.

We will first introduce the notations needed to convert the system of N equations in Equation 3.1 into matrix form. The $N \times (d+1)$-matrix whose jth row is $(x_{j1} \ldots x_{jd}, 1)$ is denoted by D, and the $N \times 1$ matrix (column vector) of the different values of $y_1 \ldots y_N$ is denoted by $\overline{y}$. The value of 1 is included as the final dimension in each row of D in order to address the effect of the constant term w_{d+1}. Thus, the first d dimensions of D can be considered a d-dimensional data set containing the N instances of the independent variables, and $\overline{y}$ is a corresponding N-dimensional column-vector of response variables. The $(d+1)$-dimensional row-vector of coefficients $w_1 \ldots w_{d+1}$ is denoted by $\overline{W}$. This creates an *over-determined* system of equations corresponding to the rows of the following matrix-wise representation of Equation 3.1:

$$\overline{y} \approx D\overline{W}^T \tag{3.2}$$

Note the use of "≈" because we have not included the error term ϵ_j of Equation 3.1. The least-squares error $\sum_{j=1}^{N} \epsilon_j^2$ *of predicting the response variable* is optimized by minimizing the squared error $||D\overline{W}^T - \overline{y}||^2$ in order to learn the coefficient vector $\overline{W}$. By applying differential calculus to the objective function, we obtain the following condition:

$$2D^T D\overline{W}^T - 2D^T\overline{y} = 0 \tag{3.3}$$

Therefore, the optimal coefficients for this minimization problem are provided by the following equation:

$$\overline{W}^T = (D^T D)^{-1} D^T \overline{y} \tag{3.4}$$

Note that $D^T D$ is a $(d+1) \times (d+1)$ matrix, which needs to be inverted in order to solve this system of equations. In cases where the matrix D is of rank less than $(d+1)$, the matrix

D^TD will not be invertible. In such cases, Equation 3.2 represents an *under-determined* system, which has infinitely many solutions with zero error (outlier scores). In such cases, it is impossible to score the in-sample data points in a reasonable way because of paucity of data. Nevertheless, it is still possible to score out-of-sample data points by using the learned coefficients. To minimize overfitting, regularization is used. Given a regularization parameter $\alpha > 0$, we add the term $\alpha||W||^2$ to the objective function. Therefore, the new objective function J is as follows:

$$J = ||D\overline{W}^T - \overline{y}||^2 + \alpha||W||^2 \tag{3.5}$$

This optimization problem can be solved in a similar way by setting the gradient of J with respect to $\overline{W}$ to 0.

$$2(D^TD + \alpha I)\overline{W}^T - 2D^T\overline{y} = 0 \tag{3.6}$$

Here, I represents a $(d+1) \times (d+1)$ identity matrix. The regularized solution is as follows:

$$\overline{W}^T = (D^TD + \alpha I)^{-1}D^T\overline{y} \tag{3.7}$$

The closed-form solution to this problem is particularly convenient, and is one of the cornerstones of regression analysis in classical statistics. The outlier scores then correspond to the absolute values of the elements in the N-dimensional error vector $\overline{\epsilon} = \overline{y} - D\overline{W}^T$. However, it is important to note that using different attributes as the dependent variable will provide different outlier scores.

To understand this point, it is useful to examine the special case of 2-dimensional data:

$$Y = w_1 \cdot X + w_2 \tag{3.8}$$

In this case, the estimation of the coefficient w_1 has a particularly simple form, and it can be shown that the best estimate for w_1 is as follows:

$$w_1 = \frac{Cov(X,Y)}{Var(X)}$$

Here, $Var(\cdot)$ and $Cov(\cdot)$ correspond to the variance and covariance of the underlying random variables. The value w_2 can further be easily estimated, by plugging in the means of X and Y into the linear dependence, once w_1 has been estimated. In general, if X is regressed on Y instead of the other way around, one would have obtained $w_1 = \frac{Cov(X,Y)}{Var(Y)}$. Note that the regression dependencies would have been different for these cases.

The set of coefficients $w_1 \ldots w_{d+1}$ define a lower-dimensional hyperplane which fits the data as well as possible in order to optimize the error in the dependent variable. This hyperplane may be different for the same data set, depending upon which variable is chosen as the dependent variable. In order to explain this point, let us examine the behavior of two attributes from the *Auto-Mpg* data set of the *UCI Machine Learning Repository* [203].

Specifically, the second and the third attributes of the *Auto-Mpg* data set correspond to the *Displacement* and *Horsepower* attributes in a set of records corresponding to cars. The scatter plot for this pair of attributes is illustrated in Figure 3.2. Three regression planes have been shown in this figure, which are as follows:

- One regression plane is drawn for the case, when the *Horsepower (Y-axis)* is dependent on the *Displacement (X-axis)*. The residual in this case is the error of prediction of the *Horsepower* attribute. The sum of squares of this residual over various data points is optimized.

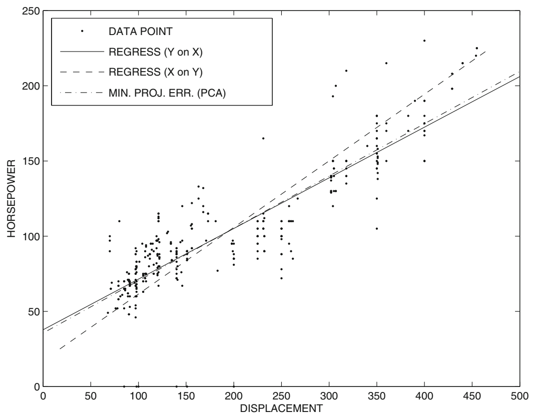


Figure 3.2: Optimal regression plane depends upon the choice of residual which is optimized

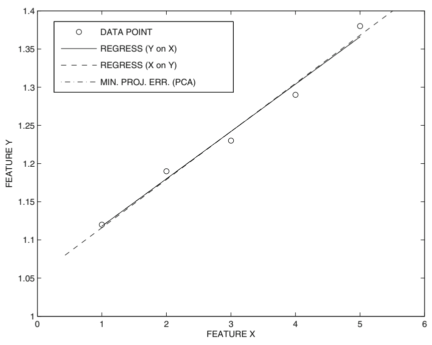
- The second regression plane is drawn for the case, when the *Displacement (X-axis)* is dependent on the *Horsepower (Y-axis)*. The residual in this case is the error in prediction of the *Displacement* attribute.

- In the last case, the goal is to estimate a hyperplane that optimizes the aggregate mean-squared distance (i.e., residual error) of the data points from it. Thus, the residual in this case is the distance of each point to the hyperplane in a normal direction to the hyperplane, and it provides an outlier score. The computation of such a hyperplane will be discussed in a later section on principal component analysis (PCA).

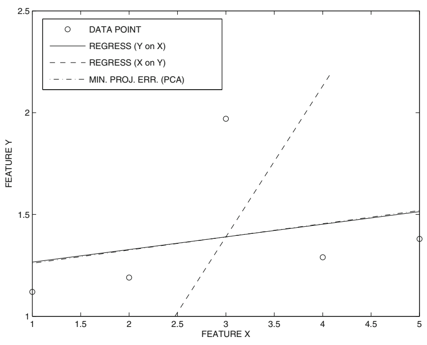
It is evident from Figure 3.2 that the optimal hyperplanes in these different cases are quite different. While the optimization of the mean square projection distance produces a hyperplane which is somewhat similar to the case of Y-on-X regression, the two are not the same. This is because these different cases correspond to different choices of errors on the residuals that are optimized, and therefore correspond to different best fitting hyperplanes. It is also noteworthy that the three projection planes are collinear and pass through the mean of the data set.

When the data fits the linear assumption well, all these hyperplanes are likely to be similar. However, the presence of noise and outliers can sometimes result in drastic negative effects on the modeling process. In order to illustrate this point, a variation of an example from [467] is used. In Figure 3.3, the different regression planes for two sets of five data points have been presented corresponding to different dependent variables. The two sets of five data points in Figures 3.3(a) and (b) are different by only one point, in which the Y-coordinate is distorted during data collection. As a result, this point does not fit the remaining data very well.

The original data set in Figure 3.3(a) fits the linear assumption very well. Therefore, all the three regression planes tend to be very similar to one another. However, after the perturbation of a single data point, the resulting projection planes are drastically perturbed. In particular, the X on Y-regression plane is significantly perturbed so as to no longer represent the real trends in the underlying data set. It is also noteworthy that the optimal projection plane is closer to the more stable of the two regression models. This is a general property of optimal projection planes, since they optimize their orientation in a stable way



(a) Regression with no outliers



(b) Regression with one perturbed point

Figure 3.3: Drastic effects of outliers on quality of regression analysis

so as to globally fit the data well. The determination of such planes will be discussed in the next section.

The residual ϵ_j provides useful information about the outlier score of the data point j. The mean of these residuals is expected to be 0, and the variance of these residuals can be estimated directly from the data. The Z-values of these residuals can also be used as outlier scores. The most common assumption in regression modeling is to assume that the error term ϵ_i is a normal distribution, which is centered at zero. Then, the t-value test discussed in Chapter 2 can be used directly on the different residuals, and the outlying observations can be subsequently removed. The normal assumption on the residuals implies that the vector of coefficients is also normally distributed with mean and variances, as discussed earlier.

One problem with the approach is its instability to the presence of outliers. When the outliers have drastic effects on the regression, such as in the case of the X-on-Y regression in Figure 3.3(b), the removal of outliers is likely to result in the removal of the wrong observations, since the regression parameters are drastically incorrect. Ironically, the presence of outliers prevents the proper modeling of outlier scores. One approach to address this problem is to use ensemble methods in which different subsets of points are sampled repeatedly to score the data points differently and average the scores. Note that all points can be scored with a given model because out-of-sample points can also be scored with the linear regression model. In cases where the number of training points is small, only out-of-sample points are scored for the averaging process. In other words, half the points are sampled to construct the model and the remaining half are scored. This process is repeated multiple times in order to score all the data points and provide the averaged predictions.

3.2.1.1 Applications of Dependent Variable Modeling

These scores can be used in a variety of applications associated with dependency-oriented data types. For example, in contextual data types, such as spatial and temporal data, these scores can be used to discover anomalous contextual outliers. These techniques will be discussed in detail in Chapters 9 and 11. These scores can also be used to remove those data points that are detrimental for learning and regression modeling applications. For example, in a regression modeling application, one can remove training data points with large training errors in order to improve the learning model.

It turns out that such models are useful even for *unsupervised* outlier detection of multidimensional data [429]. The basic idea is to repeatedly apply the regression modeling approach by fixing one of the attributes as the dependent variable and the remaining attributes as the independent variables. Thus, for a d-dimensional data set, a total of d regression models can be created. The squared errors of predicting each attribute are added in a weighted way in order to define the outlier score of a data point. This model is discussed in detail in section 7.7 of Chapter 7.

3.2.2 Linear Modeling with Mean-Squared Projection Error

The previous section discussed the case in which a particular variable (behavioral attribute) is considered special, and the optimal plane is determined in order to minimize the mean-squared error of the residuals for this variable. A more general form of regression modeling is one in which all variables are treated in a similar way, and the optimal regression plane is determined the minimize the *projection error* of the data to the plane. In the following discussion, it will be assumed that the data set containing N points is mean-centered, so that the mean of each dimension is 0.

The projection error of the data to the plane is the sum of the squares of the distances of the points to their projection into the plane. The projection of a point 'A' on the plane is the closest point on the plane to 'A,' and is computed using a line passing through 'A' in a normal direction to the plane. The point at which this line intersects the plane is the projection point. Thus, in this case, let us assume that we have a set of variables $x_1 \ldots x_d$, and the corresponding regression plane is as follows:

$$w_1 \cdot x_1 + \ldots + w_d \cdot x_d = 0 \tag{3.9}$$

It is assumed that the data is mean-centered, and therefore the constant term w_{d+1} is missing in this case. Each variable is associated with a coefficient, and the "special" (dependent) variable y is missing in this case. Consider a setting in which we have N instances corresponding to the d-dimensional data points, denoted by $\overline{X_1} \ldots \overline{X_N}$. If the number of points N is greater than the dimensionality d, the system of equations is over-determined and all the points in the data may not satisfy Equation 3.9, no matter how the vector of coefficients $\overline{W} = (w_1 \ldots w_d)$ is selected. Therefore, we need to allow for an error value ϵ_j:

$$\overline{W} \cdot \overline{X_j} = \epsilon_j \quad \forall j \in \{1 \ldots N\} \tag{3.10}$$

Therefore, the goal is to determine the row vector $\overline{W} = (w_1 \ldots w_d)$ of coefficients, so that the sum of the squared errors $\sum_{i=1}^{N} \epsilon_j^2$ is minimized. In order to address issues of scaling and avoid the trivial solution $\overline{W} = 0$, a normalization constraint is assumed:

$$||\overline{W}||^2 = \sum_{i=1}^{d} w_i^2 = 1 \tag{3.11}$$

Note that this scaling constraint ensures that the absolute value of $\overline{W} \cdot \overline{X_j} = \epsilon_j$ is exactly equal to the distance of data point $\overline{X_j}$ from the hyperplane in Equation 3.9.

As before, let D be an $N \times d$ matrix containing d-dimensional points in its rows, which are denoted by $\overline{X_1} \ldots \overline{X_N}$. One can succinctly write the N-dimensional column vector of distances of the different data points to this regression plane as $\overline{\epsilon} = D\overline{W}^T$ according to Equation 3.10. This column vector contains the outlier scores. The L_2-norm $||D\overline{W}^T||^2$ of the column vector of distances is the objective function, which needs to be minimized to determine the optimal coefficients $w_1 \dots w_d$. This is a constrained optimization problem because of the normalization constraint in Equation 3.11. One can set the gradient of the Lagrangian relaxation $||D\overline{W}^T||^2 - \lambda(||\overline{W}||^2 - 1)$ to 0 in order to derive a constraint on the optimal parameter vector $\overline{W}$. This constraint turns out to be in eigenvector form:

$$[D^T D]\overline{W}^T = \lambda \overline{W}^T \tag{3.12}$$

Which eigenvector of the positive semi-definite matrix $D^T D$ provides the lowest objective function value? Substituting for $[D^T D]\overline{W}^T$ from Equation 3.12 in the original objective function evaluates it to $\lambda||\overline{W}||^2 = \lambda$. *The aggregate squared error along a particular eigenvector is equal to the eigenvalue.* Therefore, the optimal vector $\overline{W}$ is the smallest eigenvector of $D^T D$. Note that the matrix $D^T D$ is a scaled version of the covariance matrix Σ of the mean-centered data matrix D:

$$\Sigma = \frac{D^T D}{N} \tag{3.13}$$

The scaling of a matrix does not affect its eigenvectors but it scales the eigenvalues to the *variances* along those directions instead of the *aggregate errors*. The data is therefore assumed to be compactly represented by the $(d-1)$-dimensional hyperplane, which is perpendicular to the smallest eigenvector of the covariance matrix Σ. The outlier scores correspond to the distances of the data points from their nearest point on (i.e., perpendicular distance to) this hyperplane. However, a dimensionality of $(d-1)$ is often too large to discover sufficiently discriminative outlier scores. The aforementioned solution, nevertheless, provides a first step to an effective (and more general) solution to the problem, which is referred to as *principal component analysis (PCA)*. The PCA method generalizes the aforementioned solution to a k-dimensional hyperplane, in which the value of k can vary at any value between $\{1 \dots d-1\}$. Because of its importance to outlier analysis, this method will be discussed in a dedicated section of its own along with corresponding applications.

3.3 Principal Component Analysis

The least-squares formulation of the previous section simply tries to find a $(d-1)$-dimensional hyperplane which has an optimum fit to the data values and the score is computed along the remaining orthogonal direction. Principal component analysis can be used to solve a general version of this problem. Specifically, it can find optimal representation hyperplanes of *any* dimensionality. In other words, the PCA method can determine the k-dimensional hyperplane (for any value of $k < d$) that minimizes the squared projection error over the remaining $(d-k)$ dimensions. The optimization solution in the previous section is a special case of principal component analysis, which is obtained by setting $k = d - 1$.

In principal component analysis, the $d \times d$ covariance matrix over d-dimensional data is computed, where the (i,j)th entry is equal to the covariance between the dimensions i and j for the set of N observations. As discussed in the previous section, consider a multidimensional data set D of dimensionality d and size N. The N different rows of D are denoted by $\overline{X_1} \dots \overline{X_N}$. Each row is a d-dimensional instance in the data. The individual dimensions of the ith row are denoted by $\overline{X_i} = [x_{i1} \dots x_{id}]$, where x_{ij} is the jth dimension of the ith instance $\overline{X_i}$. Let us denote the $d \times d$ covariance matrix of the data set by Σ, in which the (i,j)th entry is the covariance between the ith and jth dimensions. Since the data set D is mean-centered, the covariance matrix Σ may be expressed as follows:

$$\Sigma = \frac{D^T D}{N} \tag{3.14}$$

This matrix can be shown to be symmetric and positive semi-definite. It can therefore be diagonalized as follows:

$$\Sigma = P \Delta P^T$$

Here, Δ is a diagonal matrix, and P is an a matrix, whose columns correspond to the (orthonormal) eigenvectors of Σ. The corresponding entries in the diagonal matrix Δ provide the eigenvalues. In the aforementioned section, we stated that the *normal hyperplane to* the smallest eigenvector of Σ provides the $(d-1)$-dimensional hyperplane that approximates the data with the least-squared error. Another way of stating this is that the subspace *that is a linear combination of* the $(d-1)$ *largest* eigenvectors provides an axis system in which the data can be approximately represented with very little loss. It is possible to generalize this argument with the k largest eigenvectors to define a corresponding k-dimensional subspace of (approximate) representation. Here, k can be any value in $\{1 \dots d-1\}$ and it might not be set to only $(d-1)$. Outliers are data points for which the error of this approximation is high.

Therefore, in principal component analysis, the first step is to transform the data into a new axis system of representation. The orthonormal eigenvectors provides the axes directions along which the data should be projected. The key properties of principal component analysis, which are relevant to outlier analysis, are as follows:

Property 3.3.1 (PCA Properties) *Principal component analysis provides a set of eigenvectors satisfying the following properties:*

- *If the data is transformed to the axis-system corresponding to the orthogonal eigenvectors, the variance of the transformed data along each axis (eigenvector) is equal to the corresponding eigenvalue. The covariances of the transformed data in this new representation are 0.*

- *Since the variances of the transformed data along the eigenvectors with small eigenvalues are low, significant deviations of the transformed data from the mean values along these directions may represent outliers.*

More details and properties of PCA may be found in [33, 296]. PCA provides a more general solution than the 1-dimensional optimal solution of the previous section, because the PCA-solution provides a recursive solution of *any* dimensionality depending on the choice of parameter k. It is also noteworthy that the PCA approach potentially uses *all* the solutions of Equation 3.12 rather than using only the smallest eigenvector.

The data can be transformed to the new axis system of orthonormal eigenvectors, with transformed d-dimensional records denoted by $\overline{X_1}' \dots \overline{X_N}'$. This can be achieved by using the product between the original row-vector representation $\overline{X_i}$ and the matrix P containing the new axes (orthonormal eigenvectors) in its columns:

$$\overline{X_i}' = [x'_{i1} \dots x'_{id}] = \overline{X_i} P \tag{3.15}$$

Let D' be the transformed data matrix for which the ith row contains the transformed point $\overline{X_i}'$. The transformed data matrix D' in the de-correlated axis-system can be expressed in terms of the original data matrix D as follows:

$$D' = DP \tag{3.16}$$

In this new representation, the inter-attribute covariances in the new data matrix D' are zero, and the variances along the individual attributes correspond to the coordinates along

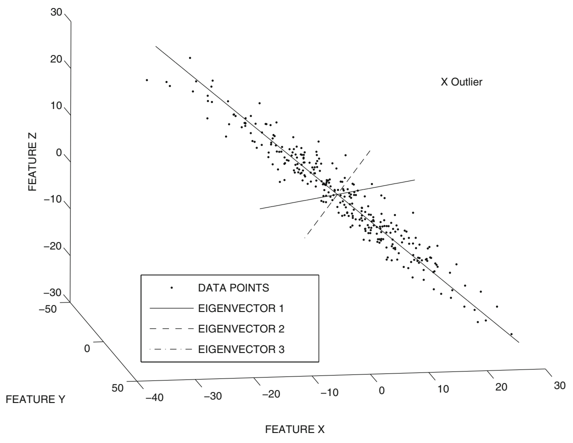


Figure 3.4: Eigenvectors correspond to directions of correlations in the data. A small number of eigenvectors can capture most of the variance in the data.

the eigenvectors eigenvalues. For example, if the jth eigenvalue is very small, then the value of x'_{ij} in this new transformed representation does not vary much over the different values of i and fixed j. For mean-centered data, these values of x'_{ij} are approximately equal to the mean value of 0, unless the data point is an outlier.

The beautiful part about PCA is that the largest eigenvectors provide the key directions of *global* correlation in a single shot. These directions are also referred to as the *principal components*, which are uncorrelated and retain most of the data variance. In real settings, it is common for a large fraction of the eigenvalues to be very small. This means that most of the data aligns along a *much lower-dimensional subspace.* This is very convenient from the perspective of outlier analysis, because the observations that do not respect this alignment can be assumed to be outliers. For example, for an eigenvector j that has a small eigenvalue, a large deviation of x'_{ij} for the ith record from other values of x'_{kj} is indicative of outlier behavior. This is because the values of x'_{kj} do not vary much, when j is fixed and k is varied. Therefore, the value x'_{ij} may be deemed to be unusual in these settings.

The effectiveness of principal component analysis in exposing outliers from the underlying data set can be illustrated with an example. Consider the scatterplot of the 3-dimensional data illustrated in Figure 3.4. In this case, the corresponding eigenvectors have been ordered by decreasing eigenvalues (variances), although this is not immediately obvious from the figure in this 2-dimensional perspective. In this case, the standard deviation along the first eigenvector is three times that along the second eigenvector and nine times that along the third eigenvector. Thus, most of the variance would be captured in the lower-dimensional subspace formed by the top two eigenvectors, although a significant amount of variance would also be captured by selecting only the first eigenvector. If the distances of the original data points to the 1-dimensional line corresponding to the first eigenvector (and passing through the mean of the data) are computed, the data point 'X' in the figure would be immediately exposed as an outlier. In the case of high-dimensional data, most of the variance of the data can be captured along a much lower k-dimensional subspace. The residuals for the data points can then be computed by examining the projection distances to this k-dimensional hyperplane passing through the mean of the data points. Data points that have very large distances from this hyperplane can be identified as outliers. Although it is possible to use this distance as an outlier score, it is possible to sharpen the score further with normalization as follows. The squared Euclidean distance to this hyperplane can be decomposed into the sum of the squares of the $(d-k)$ distances along the smallest eigenvectors. Each of these $(d-k)$ squared distances should be divided by the corresponding eigenvalue (for variance standardization) and the scaled values should be added to provide the final result. The intuition behind this scaling is that the variance along an eigenvector is its eigenvalue, and therefore a large deviation along a smaller eigenvalue should be rewarded to a greater degree.

3.3.1 Connections with the Mahalanobis Method

In the aforementioned method, we remove the k largest eigenvectors in a *hard* way and compute a weighted sum of these squared distances along the remaining $(d-k)$ distances as the outlier score. A simpler special case would be to use a *soft* approach for weighting the distances along *all* the different eigenvectors rather than *selecting* a particular set of eigenvectors in a *hard* way. This special case turns out to be the same as the Mahalanobis method in section 2.3.4 of Chapter 2.

This computation is achieved by evaluating the normalized distance of the data point to the centroid along the direction of *each principal component*. Let $\overline{e_j}$ be the jth eigenvector with a variance (eigenvalue) of λ_j along that direction. The overall normalized outlier score of a data point $\overline{X}$, to the centroid[1] $\overline{\mu}$ of the data is given by the sum of squares of these values:

$$\text{Score}(\overline{X}) = \sum_{j=1}^{d} \frac{|(\overline{X} - \overline{\mu}) \cdot \overline{e_j}|^2}{\lambda_j} \tag{3.17}$$

Note the presence of λ_j in the denominator, which provides a soft weighting. A simple algorithm to compute these scores for the rows of an $n \times d$ data matrix D is as follows:

1. Compute the covariance matrix Σ of the original data matrix D and diagonalize it as $\Sigma = P \Delta P^T$.

2. Transform the data D to a new de-correlated axis-system as $D' = DP$.

3. Standardize each column of D' to unit variance by dividing it with its standard deviation.

4. For each row of D', report its (squared) Euclidean distance from the centroid of D' as its outlier score.

It is important to note that most of the contribution to the outlier score in Equation 3.17 is provided by deviations along the principal components with small values of λ_j, when a data point deviates significantly along such directions. This is also captured by the important standardization step in the aforementioned description. This step recognizes that the principal components represent the independent concepts in the data and *implements a form of soft weighting of the transformed dimensions by standardization* rather than selecting a subset of transformed dimensions in a hard way. The sum of the squares of the Euclidean distances of the point from the centroid along the transformed dimensions is a χ^2-distribution with d degrees of freedom. The value of the aggregate residual is compared

[1] Although the centroid is the origin for mean-centered data, it is also possible to transform non-centered data once the eigenvectors of the covariance matrix have been computed. The expression of Equation 3.17 continues to hold for settings in which $\overline{\mu}$ is not necessarily 0.

to the cumulative distribution for the χ^2-distribution in order to determine a probability value for the level of anomalousness. The aforementioned approach was first used in [493].

Although it might not be immediately apparent, the score computed above is the same as the Mahalanobis method discussed in section 2.3.4 of Chapter 2 (cf. Exercise 11). Specifically, the Mahalanobis distance value between $\overline{X}$ and $\overline{\mu}$ computed in Equation 2.21 of that section is *exactly the same* as the score in Equation 3.17 above, except that the eigenvector analysis above provides a better understanding of how this score is decomposed along the different directions of correlation. Another advantage of this interpretation of the method is that it can be naturally extended to settings in which the data is distributed on a *non-linear manifold* rather than the linear setting of Figure 3.4. This extension will be discussed in section 3.3.8.

3.3.2 Hard PCA versus Soft PCA

The Mahalanobis method can be viewed as a type of soft PCA in which the principal components are weighted rather than pre-selected. The PCA decomposition also allows the option of ignoring the large eigenvectors and using only the smallest $\delta < d$ eigenvectors in order to compute the outlier scores. However, the benefits of doing so are unclear because the Mahalanobis approach already performs a kind of soft pruning with the inverse weighting of the contributions of each eigenvector by the eigenvalues. It is not uncommon for a rare value to align along a long eigenvector when all attributes are extreme values in a correlated data set. By explicitly pruning the long eigenvectors, such outliers can be missed. Therefore, hard PCA focuses only on finding dependency-oriented outliers, whereas the Mahalanobis method (soft PCA) can discover both dependency-oriented outliers and extreme values. In this sense, the Mahalanobis method can be viewed as a more elegant generalization of hard PCA. Hard PCA focuses on the *reconstruction* error of representing the data in a low-dimensional space, which introduces an additional parameter of selecting the dimensionality of the representation space. Introducing parameters always results in data-specific unpredictability of results in unsupervised settings, where there is no guided way of tuning the parameters.

3.3.3 Sensitivity to Noise

Principal component analysis is generally more stable to the presence of a few outliers than the dependent variable analysis methods. This is because principal component analysis computes the errors with respect to the *optimal hyperplane*, rather than a *particular variable*. When more outliers are added to the data, the optimal hyperplane usually does not change too drastically. Nevertheless, in some settings, the presence of outliers can cause challenges. In such cases, several techniques exist for performing robust PCA. For example, this approach can be used in order to determine the obvious outliers in the first phase. In the second phase, these outliers can be removed, and the covariance matrix can be constructed more robustly with the remaining data. The scores are then recomputed with the adjusted covariance matrix. This approach can also be applied iteratively. In each iteration, the obvious outliers are removed, and a more refined PCA model is constructed. The final outlier scores are the deviation levels in the last iteration.

3.3.4 Normalization Issues

PCA can sometimes yield poor results when the scales of the different dimensions are very different. For example, consider a demographic data set containing attributes such as *Age* and *Salary*. The *Salary* attribute may range in the tens of thousands, whereas the *Age* attribute is almost always less than a hundred. The use of PCA would result in the principal components being dominated by the high-variance attributes. For example, for a 2-dimensional data set containing only *Age* and *Salary*, the largest eigenvector will be almost parallel to the *Salary* axis, irrespective of very high correlations between the *Age* and *Salary* attributes. This can reduce the effectiveness of the outlier detection process. Therefore, a natural solution is to normalize the data, so that the variance along each dimension is one unit. This is achieved by dividing each dimension with its standard deviation. This implicitly results in the use of a *correlation matrix* rather than the *covariance matrix* during principal component analysis. Of course, this issue is not unique to linear modeling, and it is often advisable to use such pre-processing for most outlier detection algorithms.

3.3.5 Regularization Issues

When the number of data records N is small, the covariance matrix cannot be estimated very accurately, or it might be ill-conditioned. For example, a data set containing a small number of records might have zero variance along some of the dimensions, which might underestimate the true variability. In such cases, it is desirable to use regularization to avoid overfitting. This type of regularization intuitively similar to the notion of Laplacian smoothing discussed for the EM method in Chapter 2. The basic idea is to adjust the covariance matrix Σ by adding αI to it, where I is a $d \times d$ identity matrix and $\alpha > 0$ is a small regularization parameter. The eigenvectors of $(\Sigma + \alpha I)$ are then used for computing the scores. The basic effect of this modification is intuitively equivalent to adding a small amount of independent noise with variance α to each dimension before computing the covariance matrix.

Alternatively, one can use cross-validation for scoring. The data is divided into m folds, and the PCA model is constructed only on $(m-1)$ folds. The remaining fold is scored. This process is repeated for each fold. An ensemble-centric variant is to sample a subset of the data to build the model, and score all the points. The process is repeated over multiple samples and the scores are averaged. The effectiveness of this approach is shown in [35].

3.3.6 Applications to Noise Correction

Most of this book is devoted to either removal of outliers as noise or identification of outliers as anomalies. However, in some applications, it may be useful to *correct* the errors in outliers so that they conform more closely to the broad patterns in the data. PCA is a natural approach for achieving this goal because the principal components represent the broad patterns. The basic idea is that the *projection of the data point on the k-dimensional hyperplane corresponding to the largest eigenvalues (and passing through the data mean) provides a corrected representation.* Obviously such an approach is likely to correct the outlier points significantly more than most of the other normal data points. Some theoretical and experimental results on why such an approach is likely to improve data quality is provided in [21].

A similar approach to PCA, referred to as *Latent Semantic Indexing (LSI)* has also been used for text data to improve retrieval quality [162, 425]. Text representations are inherently noisy because the same word may mean multiple things (*synonymy*) or the same concept can be represented with multiple words (*polysemy*). This leads to numerous challenges in virtually all similarity-based applications. In particular, it has been observed in [425] that the use of such dimensionality reduction methods in text data significantly improves the effectiveness of similarity computations, because of the reduction in the noise effects of *synonymy* and *polysemy*. It has been observed [425] that such dimensionality reduction methods significantly improve similarity computations in text because of the reduction in the noise effects of *synonymy* and *polysemy*. The technique of LSI [162] is a variant of PCA, which was originally developed for efficient indexing and retrieval. However, it was eventually observed that the quality of similarity computations improves as well [425]. This observation was taken to its logical conclusion in [21], where significant noise reduction was shown both theoretically and experimentally with PCA-based techniques.

An even more effective approach to noise correction is to combine outlier removal and re-insertion with the correction process. The first step is to perform PCA, and remove the top outliers on the basis of a t-test with respect to the optimal plane of representation. Subsequently, PCA is performed again on this cleaner data set in order to generate the projection subspaces more accurately. The projections can then be performed on this corrected subspace. This process can actually be repeated iteratively, if desired in order to provide further refinement. A number of other approaches to perform regression analysis and outlier removal in a robust way are presented in [467].

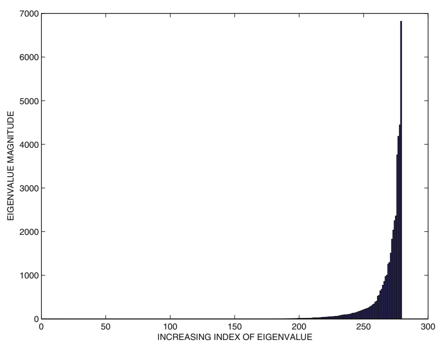
3.3.7 How Many Eigenvectors?

As discussed earlier, the eigenvectors with the largest variance provide the most informative subspaces for data representation, outlier analysis, and noise correction. In some cases, it is not necessary to select a subset of the dimensions in a hard way because a soft inverse weighting with eigenvalues is sufficiently effective. However, in many applications such as noise correction, the data needs to be projected into a subspace of lower dimensionality by selecting a specific number of eigenvectors. Therefore, a natural question arises, as to how the dimensionality k of the projection subspace should be determined.

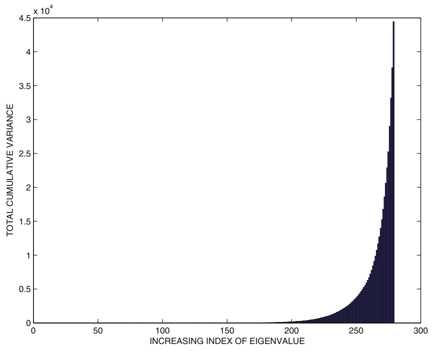
One observation in most real data sets is that the vast number of eigenvalues are relatively small, and most of the variance is concentrated in a few eigenvectors. An example illustrated in Figure 3.5 shows the behavior of the 279 eigenvectors of the *Arrythmia* data set of the *UCI Machine Learning Repository* [203]. Figure 3.5(a) shows the absolute magnitude of the eigenvalues in increasing order, whereas Figure 3.5(b) shows the total amount of variance retained in the top-k eigenvalues. In essence, Figure 3.5(b) is derived by using the cumulative sum over the eigenvalues in Figure 3.5(a). While it was argued at the beginning of the chapter that the *Arrythmia* data set is weakly correlated along many of the dimensions, on a pairwise basis, it is interesting to note that it is still possible[2] to find a small number of directions of global correlation along which most of the variance is retained. In fact, it can be shown that the smallest 215 eigenvalues (out of 279) *cumulatively* contain less than 1% of the variance in the data set.

In other words, most eigenvalues are very small. Therefore, it pays to retain the eigenvectors corresponding to extremely large values, with respect to the average behavior of the

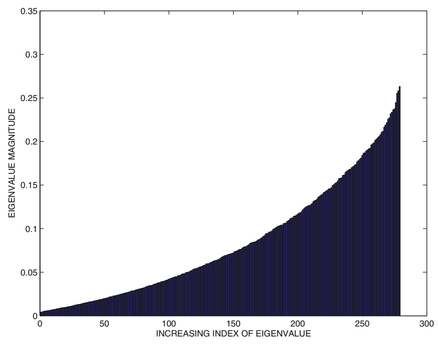
[2]This is partially because the data set is relatively small with only 452 records. For example, the results in Figures 3.5(c) and (d) show that even for a uniformly distributed data set of the same size, it is possible to find some skews in the eigenvalues because of (undesirable) overfitting. On the other hand, a strength of PCA is that the cumulative effects of even weak correlations become magnified with increasing dimensionality, and it becomes possible to find a much lower-dimensional subspace contain the informative projections.



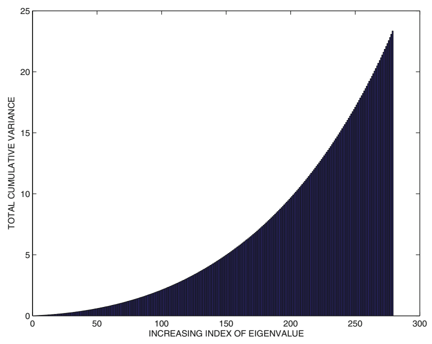
(a) Magnitude of eigenvalues (Increasing index): *Arrythmia*



(b) Variance in smallest k eigenvalues: *Arrythmia*

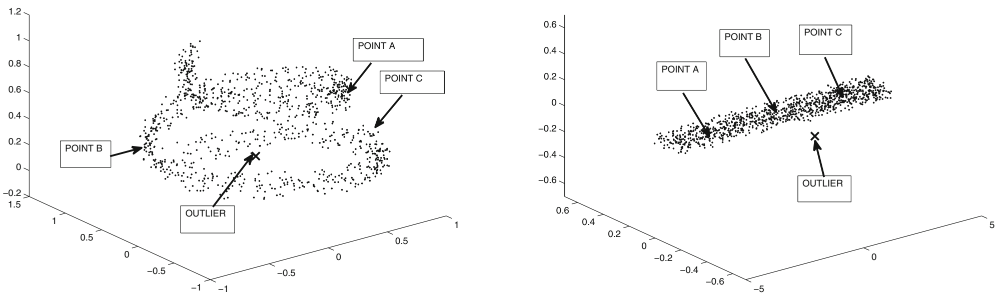


(c) Magnitude of eigenvalues *Uniform: 452 records only*



(d) Variance in smallest k eigenvalues *Uniform: 452 records only*

Figure 3.5: A few eigenvalues contain most of the variance (*Arrythmia*)



(a) Nonlinear manifold with outlier (b) Transformed representation of (a)

Figure 3.6: Nonlinear transformations can expose outliers that are hard to discover with linear models

eigenvalues. How to determine, what is “extremely large?” This is a classical case of extreme value analysis methods, which were introduced in Chapter 2. Therefore, each eigenvalue is treated as a data sample, and the statistical modeling is used to determine the large values with the use of hypothesis testing. A challenge in this case is that the sample sizes are often small unless the data set is very high dimensional. Even for relatively high-dimensional data sets (e.g., 50-dimensional data sets), the number of samples (50 different eigenvalues) available for hypothesis testing is relatively small. Therefore, this is a good candidate for the t-value test. The t-value test can be used in conjunction with a particular level of significance and appropriate degrees of freedom to determine the number of selected eigenvectors.

3.3.8 Extension to Nonlinear Data Distributions

Although the PCA methodology works well for linearly aligned data distributions such that those discussed in Figure 3.4, the vanilla version of PCA is not effective in cases where the data is aligned along nonlinear manifolds. An example of such a manifold is illustrated in Figure 3.6(a). In such cases, PCA can be extended to nonlinear settings with the use of a technique known as the *kernel trick*. The basic idea in nonlinear PCA is to use a data transformation so that the data aligns along a linear hyperplane in the transformed space. In other words, the mapping unfurls the non-linear manifold into a linear embedding. An illustrative example of such an embedding is illustrated in Figure 3.6(b). After such an embedding has been constructed, we can use the score computation method of Equation 3.17.

How can such embeddings be constructed? A naive approach would be to use *explicit* polynomial transformations of the data. For example, a second-order polynomial transformation would represent each point with $d+d^2$ dimensions. In other words, the d original dimensions are included and d^2 new dimensions are added corresponding to pairwise products of data values. Subsequently, *all nonzero* eigenvectors can be extracted from the covariance matrix of this new representation. Note that the number of nonzero eigenvectors can be larger than d in this setting because we do not know the dimensionality of the transformed space. However, the number of nonzero eigenvectors is always no larger than the number of data points. Equation 3.17 can be applied to this new representation in order to determine the scores. Unfortunately, this is not a practical approach because it expands the number of dimensions to $O(d^2)$, and the size of the covariance matrix is $O(d^4)$. Such an approach would be impractical from a computational point of view. Fortunately, these transformations need not be done *explicitly*; rather, they can be done implicitly with the use of carefully designed $N \times N$ similarity matrices between all pairs of data points. This technique is also referred to as the *kernel trick*, although a generalized point of view encompasses various methods such as *spectral embeddings* [297] and ISOMAP [542]. The approach described in this section is a direct nonlinear generalization of the Mahalanobis method with kernel functions, and is introduced in [35] as an ensemble-centric adaptation. Some of these ideas are also related to the concept of *kernel whitening* [541], which advocates scaling of all kernel features to unit variance before applying any learning method.

In order to simplify further exposition, we will first describe an alternative method for extracting (linear) PCA from a dot-product *similarity matrix* rather than a covariance matrix. This alternative description of linear PCA is easier to generalize to the nonlinear case. In our earlier description of PCA, we first construct the covariance matrix $\Sigma = \frac{D^T D}{N}$ from the mean-centered matrix D and diagonalize it to extract its $d \times d$ eigenvector matrix P, the columns of which are the new basis vectors. Subsequently, the transformed representation of data is given by projecting the data on these basis vectors as $D' = DP$. After the new representation has been determined, the outlier score is simply equal to the distance from the centroid, after standardizing the transformed data to unit variance along each (transformed) dimension. Therefore, the key is to extract the new *embedding* D' of the data representation in order to extract the outlier scores. The basis representation in the columns of P only provides an intermediate step for computing the embedding D', and is itself not essential for computing the outlier scores. The similarity-matrix approach to PCA extracts this embedding D' directly without computing these basis vectors.

In the similarity-matrix approach to PCA, the $N \times N$ dot product matrix $S = DD^T$ is constructed instead of the $d \times d$ covariance matrix $D^T D$. Note that S is a dot-product similarity matrix in which the (i, j)th entry is equal to the dot-product between the ith and jth data points. Subsequently, the positive-semidefinite similarity matrix S is diagonalized as follows:

$$S = DD^T = Q\Lambda^2 Q^T \tag{3.18}$$

Here, Q is an $N \times N$ matrix whose columns contain orthonormal eigenvectors, although only the top-d eigenvectors have non-zero eigenvalues because the matrix DD^T has rank at most d. Λ^2 is a diagonal matrix containing the non-negative eigenvalues of the positive semi-definite matrix. *Then, in the alternative similarity-matrix approach to PCA, it can be shown that the top-d scaled eigenvectors $Q\Lambda$ (i.e., first d columns of $Q\Lambda$) yield the new $N \times d$ data representation D' [515].* Therefore, this alternative approach[3] to PCA *directly* extracts the embedding from the eigenvectors and eigenvalues of the $N \times N$ dot-product similarity matrix $S = DD^T$ without computing a basis matrix P. This is particularly useful for nonlinear dimensionality reduction because it is often hard to interpret the transformed basis in terms of the original features.

The key idea in nonlinear PCA is that by replacing the dot-product similarity matrix $S = DD^T$ with a carefully chosen *kernel similarity matrix* in the aforementioned approach, nonlinear manifolds are mapped on linear hyperplanes such as the mapping from Figure 3.6(a) (data matrix D) to Figure 3.6(b) (transformed embedding $Q\Lambda$). Such similarity matrices will be discussed in section 3.3.8.1. This approach is the essence of a well-known technique known as the *kernel trick* and the corresponding approach is referred to as *kernel*

[3]This equivalence can be shown using the SVD decomposition of matrix D. In fact, the nonzero eigenvalues of DD^T and $D^T D$ are the same.

PCA. The key is to assume that the (i, j)th value of S is equal to $\Phi(\overline{X_i}) \cdot \Phi(\overline{X_j})$ in some transformed space according to some unknown transformation $\Phi(\overline{X_i})$ of higher dimensionality. Therefore, the similarity matrix S can be used to compute the Mahalanobis scores as follows:

1. Construct the positive semi-definite $N \times N$ kernel similarity function S using an off-the-shelf kernel function or other similarity matrix computation (e.g., sparsified and normalized similarity matrix used in spectral methods [378]).

2. Diagonalize the matrix $S = Q\Lambda^2 Q^T$ and set the new $N \times k$ embedding D' to the first k columns of $Q\Lambda$ corresponding to largest eigenvalues. The default assumption is to set k so that all nonzero eigenvectors of S are included. Note that k can be larger than d in the nonlinear setting.

3. Standardize each column of D' to unit variance by dividing it with its standard deviation.

4. For each row of D', report its (squared) Euclidean distance from the centroid of D' as its outlier score.

Why did we use all nonzero eigenvectors? In problems like outlier detection, aggregate trends are not sufficient because one is looking for exceptions. In fact, most outliers are emphasized along the *smaller* eigenvectors. Even a single data point deviating significantly along a small eigenvector is important for outlier detection. It is important to select all nonzero eigenvectors, so that important outliers are not missed. Assume that the columns of Q and Λ are sorted in order of decreasing eigenvalue. One should select the (unique) value of k such that $\Lambda_{kk} > 0$, and $\Lambda_{k+1,k+1} = 0$. In practice, the value of the latter will not be exactly zero because of numerical errors during eigenvalue computation. Furthermore, such computational errors magnify the inaccuracy in scoring along small eigenvectors. Therefore, some conservative threshold ϵ on the eigenvalue (e.g., 10^{-8}) can be used to select the cut-off point. Another caveat is that the approach can give poor results because of overfitting when all N eigenvectors are nonzero; in such cases, either dropping the *largest* eigenvectors or using an out-of-sample implementation (see section 3.3.8.2) is helpful.

3.3.8.1 Choice of Similarity Matrix

How does one choose the similarity matrix S? The (i, j)th entry of S is defined by the kernel similarity $K(\overline{X_i}, \overline{X_j})$ between the data points $\overline{X_i}$ and $\overline{X_j}$. There are a number of off-the-shelf kernel functions that are commonly used in such settings, although the general consensus is in favor of the Gaussian kernel [270, 541]. Some common choices of the kernel function $S = [K(\overline{X_i}, \overline{X_j})]$ are as follows:

Function	**Form**
Linear Kernel	$K(\overline{X_i}, \overline{X_j}) = \overline{X_i} \cdot \overline{X_j}$ (Defaults to PCA)
Gaussian Radial Basis Kernel	$K(\overline{X_i}, \overline{X_j}) = e^{-\|\|\overline{X_i} - \overline{X_j}\|\|^2/\sigma^2}$
Polynomial Kernel	$K(\overline{X_i}, \overline{X_j}) = (\overline{X_i} \cdot \overline{X_j} + c)^h$
Sigmoid Kernel	$K(\overline{X_i}, \overline{X_j}) = \tanh(\kappa \overline{X_i} \cdot \overline{X_j} - \delta)$

Note that the special case of linear PCA without any transformation (section 3.3) corresponds to $K(\overline{X_i}, \overline{X_j}) = \overline{X_i} \cdot \overline{X_j}$, and the corresponding kernel is also referred to as the linear kernel. Since PCA is technically defined for mean-centered kernel matrices, it is possible to mean-center[4] the kernel matrix. However, it is not essential in either the linear or nonlinear setting, and can sometimes cause unexpected problems in the nonlinear setting when all eigenvectors are nonzero.

Many of these kernel functions have parameters associated with them. The shape of the embedding will depend on the choice of the kernel function and its parameters. Note that the use of an arbitrary kernel function might result in a embedding somewhat different from Figure 3.6(b), which is only illustrative in nature. Furthermore, different choices of kernel functions or parameters might have different levels of effectiveness in exposing the outliers. Unfortunately, in unsupervised methods there is no meaningful way of choosing such kernel functions or parameters. The use of the Gaussian[5] kernel is particularly common because of its stability across different data sets. In that case, the parameter σ should be of the same order of magnitude as the pairwise distances between points. Smaller values of σ can model complex distributions but also cause overfitting. Large values of σ yield results similar to the linear Mahalanobis method. For small data sets with less than 1000 points, values of σ between two and three times the median pairwise distance between points might be desirable [35].

Kernel functions provide only one of many ways of computing the similarity matrix S. Some other ways of computing the similarity matrix are as follows:

1. One can sparsify the similarity matrix by keeping only the mutual k-nearest neighbors of each data point. All other entries in the similarity matrix are set to 0. This type of similarity matrix is used in spectral methods [378] and it tends to favor local embeddings. Furthermore, the (i, j)th entry can be normalized by the geometric mean of the sum of row i and row j in order to provide local normalization [297]. Such local normalization is used in many settings such as the LOF method discussed in section 4.2.1 of Chapter 4.

2. One can use the pairwise distance-computation between points by using the geodesic methodology of ISOMAP. Subsequently, the pairwise distances are transformed to pairwise similarities with the use of the cosine law. This approach is discussed in [33]. Unlike spectral methods, this type of approach will tend to favor global embeddings.

By selecting a particular type of similarity matrix one can effectively control the types of outliers that one will discover in various applications.

3.3.8.2 Practical Issues

It is noteworthy that the kernel embedding can be N-dimensional for a data set containing N points. In the case of the Gaussian kernel, the embedding dimensionality is often more than the dimensionality of the input data set. Each eigenvector defines a dimension of the embedding. Although large eigenvectors are more important for applications like clustering, the variations along the *small* eigenvectors are more important for outlier detection.

[4]A kernel matrix can be mean-centered by using the update $S \leftarrow (I - U/N)S(I - U/N)$, where I is an identity matrix and U is an $N \times N$ matrix containing only 1s.

[5]It is the noteworthy that a factor of 2 is omitted from the denominator in the exponent of the Gaussian kernel function for simplicity. All discussions on choice of σ are consistent with this convention.

The standardization of the dimensions of the transformed representation to unit variance emphasizes the importance of these eigenvectors, as in the case of the linear Mahalanobis method. The conservative approach is to use all the nonzero eigenvectors (while excluding those nonzero eigenvectors caused by numerical errors). However, when nearly all N eigenvalues are nonzero, it is sometimes the result of overfitting, and the results can be poor. A second issue is that the size of the kernel matrix S is $O(N^2)$ which can be very large. For example, for a data set containing a hundred thousand points, this type of approach is not practical.

One way of resolving both problems is to use an ensemble-centric adaptation [35] based on variable subsampling [32]. One can repeatedly draw a random sample from the data of size s between 50 and 1000 points. Each such sample is used to (efficiently) construct an embedding of each of the N points in a space whose dimensionality is at most s.

The first step is to construct the $s \times s$ similarity matrix S using the sample. Subsequently, we extract the s eigenvectors and eigenvalues of S as $S = Q\Lambda^2 Q^T$, where Q and Λ are both $s \times s$ matrices. However, we drop the zero eigenvectors of Q and therefore retain only the $k < s$ nonzero[6] eigenvectors of Q. The corresponding $s \times k$ matrices are Q_k and Λ_k, respectively. Therefore, $Q_k \Lambda_k$ provides the embedding of the s in-sample points.

For the remaining $(N - s)$ out-of-sample points, an $(N - s) \times s$ kernel similarity S_o is constructed[7] between the out-of-sample points and in-sample points. In other words, the (i, j)th entry in this matrix is the kernel similarity between the ith out-of-sample point and the jth in-sample point, which can also be expressed as the dot product between the points in transformed space. We will show how to use this matrix in order to derive the k-dimensional embedding of each out-of-sample data point that is consistent with the aforementioned embedding of the in-sample points.

Let the unknown $(N - s) \times k$ matrix containing the k-dimensional embedding of the out-of-sample points (in its rows) be denoted by V_k. Since the matrix V_k is unknown, the goal is to use S_o and the in-sample embedding $Q_k \Lambda_k$ in order to estimate it. Note that each entry in the similarity matrix S_o can be approximately expressed as the pairwise dot product of the out-of-sample points (rows of V_k) and in-sample points (rows of $Q_k \Lambda_k$) in transformed space. One can express this relationship using matrix products as $S_o = V_k (Q_k \Lambda_k)^T$. Then, by post-multiplying this relationship with $Q_k \Lambda_k^{-1}$, it is evident that the $(N - s) \times k$ embedding matrix of these out-of-sample points is given [481] by $V_k = S_o Q_k \Lambda_k^{-1}$. The $s \times k$ matrix $Q_k \Lambda_k$ (which is the embedding of in-sample points) is stacked with the $(N - s) \times k$ matrix $V_k = S_o Q_k \Lambda_k^{-1}$ to create a single $N \times k$ embedding matrix E of all N data points:

$$E = \begin{pmatrix} Q_k \Lambda_k \\ S_o Q_k \Lambda_k^{-1} \end{pmatrix} \tag{3.19}$$

Each column of E is standardized to zero mean and unit variance. This embedding is used to score all data points in one shot. Note that the mean row-vector of the standardized matrix E is a vector of zeros because of the standardization operation. The squared distance of each row of E to this row-wise mean of E (which is the origin) is reported as the outlier score of that row. As a result, each point will receive one outlier score. The score is divided by the dimensionality of E to ensure that the average score over all points in each ensemble

[6]Some eigenvectors might be nonzero or negative due to numerical errors. In such cases, it is advisable to use a very conservative threshold like 10^{-8} to distinguish truly nonzero values from those caused by numerical errors. This is also important because numerical errors hurt the computations along small eigenvalues more severely due to re-scaling of D'.

[7]For some data-dependent kernels like spectral methods and ISOMAP, it is not possible to construct out-of-sample similarity matrices like S_o exactly, although approximate methods exist [79].

component is 1 (because of standardization). The entire process is repeated m times with different sample sets, so that each point receives m different outlier scores. The average score of each point is reported as the final result.

3.3.8.3 Application to Arbitrary Data Types

One of the beautiful aspects of this approach is that *as long as a positive semidefinite similarity matrix exists between a set of N objects*, one can always use this approach by extracting a multidimensional embedding from the similarity matrix. The main technical obstacle is that any matrix S, which can be expressed as DD^T to obtain the embedding D in transformed space, can be shown to be positive semi-definite. For example, if we computed the $N \times N$ similarity matrix between a set of N time-series using some domain-specific function and did not find the similarity matrix S to be positive semi-definite, it means that *no embedding exists* whose dot products will give you the required similarities. There are two ways in which we can avoid this problem:

- Kernel similarity functions have been designed for a variety of data types such as time-series, strings, and graphs. These similarity functions are special because they are guaranteed to be positive semi-definite. A discussion of these different types of kernel functions may be found in [33]. Note that it is not always possible to use kernels because the similarity matrix might be defined in a domain-dependent way.

- For an arbitrary similarity matrix S with negative eigenvalues, we can drop the eigenvectors with negative eigenvalues, if their absolute magnitude is relatively small. One can view this approach as an indirect way of slightly adjusting the similarity matrix to force it to satisfy the requirements of a multidimensional embedding. The level of approximation of the similarity matrix can be shown to be dependent on the *absolute* magnitude of the dropped eigenvalues. Therefore, it is desirable for the absolute magnitudes of the negative eigenvalues to be small. An alternative approach is to regularize S to the positive semidefinite matrix $S + \alpha I$, where $\alpha > 0$ is the magnitude of the most negative eigenvalue. This approach restricts itself to increasing only the self-similarity (norms) of data points without changing inter-point similarity.

The second approach should be used in cases in which one cannot use an off-the-shelf kernel, and the similarity function is domain-dependent. Nevertheless, the dropping of eigenvectors does have the drawback that some outliers might be missed.

3.4 One-Class Support Vector Machines

One-class support vector machines (SVMs) can be viewed as variants of linear and logistic regression models in which the notion of *margin* is used to avoid overfitting, just as regularization is used in regression models. These error penalties are computed with the notion of *slack variables*. Although it is possible to use squared loss (as in regression), it is more common to use other forms of the loss function (such as *hinge-loss*) in support vector machines. This section assumes a familiarity with support-vector machines for classification. The uninitiated reader is referred to [33] for an introduction to the basics of the support-vector machines.

The main problem in using support-vector machines for outlier detection is that the model is primarily designed for *two* classes which need to be separated with the use of a decision boundary (and their associated margin hyperplanes). However, in outlier detection, the data are not labeled, and therefore (a possibly noisy) model is constructed assuming that all the provided examples belong to the normal class. In order to address this issue, it is assumed that *the origin of a kernel-based transformed representation belongs to the outlier class.* Note that the quality of the separation between the normal class and the outliers in a particular data domain and corresponding transformation will depend significantly on the validity of this assumption.

For now, we will assume that the data point $\overline{X}$ is transformed to $\Phi(\overline{X})$ using the unknown function $\Phi(\cdot)$. This transformation need not be performed explicitly, as it is performed implicitly within the optimization problem with the use of kernel similarity matrices (like the nonlinear PCA method of section 3.3.8). Note that $\Phi(\overline{X})$ is itself a high-dimensional vector in some transformed space, and the corresponding coefficients of $\Phi(\overline{X})$ is the vector $\overline{W}$, which has the same dimensionality as the transformed space. The corresponding decision boundary that separates the normal class from the outlier class is given by the following:

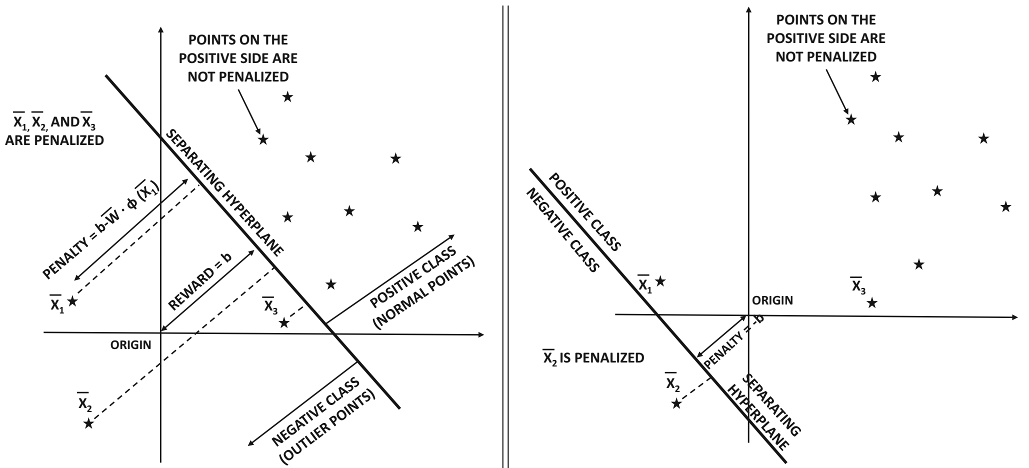
$$\overline{W} \cdot \Phi(\overline{X}) - b = 0 \tag{3.20}$$

Here, b is a variable that controls the bias. We should formulate the optimization problem so that the value of $\overline{W} \cdot \Phi(\overline{X}) - b$ is positive for as many of the N training examples as possible, because all training examples are assumed to belong to the normal (positive) class. Therefore, to account for any training example in which $\overline{W} \cdot \Phi(\overline{X}) - b$ is negative, we impose the *slack penalty* $\max\{b - \overline{W} \cdot \Phi(\overline{X}), 0\}$. On the other hand, the origin is rewarded for lying on the opposite side of this separator and therefore a negative value of $\overline{W} \cdot \Phi(\overline{X}) - b$ is desirable in the case when $\Phi(\overline{X}) = 0$. This is possible only when b is positive. This situation is illustrated in Figure 3.7(a). Therefore, in order to reward the origin for being as far away from the separator as possible on the opposite side of the normal points, we subtract b from the objective function formulation. This has the effect of pushing the separator as far away from the origin as possible towards the normal points. Furthermore, we add the margin regularizer term, which is $\frac{1}{2}||\overline{W}||^2$. Therefore, the overall objective function is as follows:

$$\text{Minimize } J = \underbrace{\frac{1}{2}||\overline{W}||^2}_{\text{Regularizer}} + \underbrace{\frac{C}{N}\sum_{i=1}^{N}\max\{b - \overline{W} \cdot \Phi(\overline{X_i}), 0\}}_{\text{Training Data Penalty}} - \underbrace{b}_{\text{Origin Reward}} \tag{3.21}$$

The constant $C > 1$ can be viewed as the differential weight of the normal points as compared to the outlier points. Specifically, one can view $\nu = 1/C$ as a prior probability that a data point in the training set is an outlier. In other words, the value of C regulates the trade-off between false positives and false negatives in this model. It is noteworthy that $\overline{W} = 0$ and $b = 0$ is a degenerate solution to this optimization problem with zero objective function value. Therefore, it is never possible for b to be strictly negative at optimality because negative values of b lead to a strictly positive value of the objective function J. The situation with a negative value of b is shown in Figure 3.7(b), which is always suboptimal with respect to the degenerate solution. It is noteworthy that if the wrong[8] type of kernel transformation is used, in which the origin is not linearly separable from the data points, and the value of C is very large, it can lead to the unfortunate situation[9] of the degenerate solution $\overline{W} = 0$ and $b = 0$.

[8]Mean-centering the kernel matrix could result in a degenerate solution. On the other hand, any nonnegative kernel matrix like an uncentered Gaussian kernel can always avoid the degenerate solution because all pairwise angles between points are less than 90^o in transformed space; therefore, the origin is always linearly


POINTS ON THE POSITIVE SIDE ARE NOT PENALIZED
$\overline{X}_1$, $\overline{X}_2$ AND $\overline{X}_3$ ARE PENALIZED
SEPARATING HYPERPLANE
PENALTY = b-$\overline{W}$ · Φ ($\overline{X}_1$)
REWARD = b
$\overline{X}_1$
$\overline{X}_2$
$\overline{X}_3$
ORIGIN
POSITIVE CLASS (NORMAL POINTS)
NEGATIVE CLASS (OUTLIER POINTS)
POINTS ON THE POSITIVE SIDE ARE NOT PENALIZED
POSITIVE CLASS
NEGATIVE CLASS
$\overline{X}_1$
$\overline{X}_3$
ORIGIN
$\overline{X}_2$ IS PENALIZED
$\overline{X}_2$
PENALTY = -b
SEPARATING HYPERPLANE

(a) Origin and normal points on opposite sides (b is positive)

(b) Origin and normal points on same side (b is negative)

Figure 3.7: The useful solutions correspond to the case shown in (a). The case shown in (b) is always suboptimal with respect to the degenerate solution $\overline{W} = 0$ and $b = 0$. Therefore, b can never be negative at optimality. Nonnegative kernels transform all points to a single orthant with a linearly separable origin and so a nontrivial solution always exists.

The positive side of the linear separator in the kernel representation corresponds to a "small" region of arbitrary shape in the *input* representation containing most of the points. Penalties are caused by points lying outside this region. An illustrative example of how the linear separator of Figure 3.7(a) might create an arbitrarily shaped region in the original input space is shown in Figure 3.8. Note that the three outliers $\overline{X_1}$, $\overline{X_2}$ and $\overline{X_3}$ lie outside this region because they violate the linear decision boundary in transformed space. Increasing the value of C increases the volume of the enclosed region containing inliers and also changes its shape.

Note that this optimization formulation is defined in an unknown transformed space $\Phi(\cdot)$. Furthermore, the transformation $\Phi(\cdot)$ is often defined *indirectly* in terms of pairwise (kernel) similarities among points. Rather than solving directly for $\overline{W}$, a more common approach is to predict the (optimized) value of $\overline{W} \cdot \Phi(\overline{X}) + b$ for test point $\overline{X}$ by using the *kernel trick* within a dual formulation. This is achieved by explicitly materializing the (non-negative) *slack variables* $\xi_1 \dots \xi_N$ for the N training points:

$$\xi_i \geq b - \overline{W} \cdot \Phi(\overline{X_i}) \tag{3.22}$$

One can then incorporate this constraint into the optimization formulation (together with non-negativity constraints on slack variables), and substitute $\max\{b - \overline{W} \cdot \Phi(\overline{X_i}), 0\}$ with ξ_i in the objective function J. This constrained formulation allows for an approach based on Lagrangian relaxation for which the dual formulation can be constructed. The dual

separable from the orthant in which all the data lies, and one rarely notices this problem in software executions. Nevertheless, the quality of the results from one-class SVMs tend to be highly unpredictable [184, 384].

[9]The original paper [479] incorrectly states that large training data penalties lead to negative values of b. The sensitivity of this approach to the kernel representation is, in fact, its major weakness. One possible solution to the degeneracy problem is to impose the constraint $||W|| = 1$ and get rid of the regularizer.

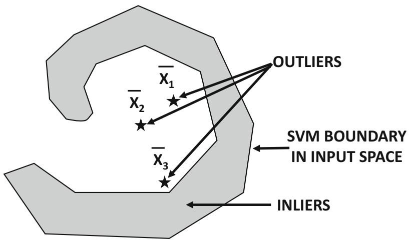


Figure 3.8: A hypothetical illustration of the linear SVM boundary of Figure 3.7(a) in the original input space. This is only an illustrative example and does not reflect an actual computation. The shape of the enclosed region is sensitive to the kernel and the parameter C.

formulation has N variables $\overline{\alpha} = [\alpha_1 \dots \alpha_N]^T$, each of which is a Lagrangian parameter corresponding to one of the constraints in Equation 3.22 for a training point. Interestingly, the dual formulation can be expressed in terms of the $N \times N$ kernel similarity matrix $S = [K(\overline{X_i}, \overline{X_j})] = [\Phi(\overline{X_i}) \cdot \Phi(\overline{X_j})]$ (rather than explicitly transformed points), which is the essence of the kernel trick. This similarity matrix is the same as the one that was used for nonlinear PCA in section 3.3.8. The dual formulation, described in [480], is as follows:

$$\begin{aligned}
\text{Minimize } & \frac{1}{2}\overline{\alpha}\, S\, \overline{\alpha}^T \\
& \text{subject to:} \\
& 0 \leq \alpha_i \leq \frac{C}{N} \quad \forall i \in \{1 \dots N\} \\
& \sum_{i=1}^{N} \alpha_i = 1
\end{aligned}$$

Although the coefficient vector $\overline{W}$ is not explicitly computed by the dual approach, it can still be determined which side of the decision boundary a data point $\overline{Y}$ lies on by using the following equivalence, which is shown in [128, 479]:

$$\overline{W} \cdot \Phi(\overline{Y}) - b = \sum_{i=1}^{N} \alpha_i \cdot K(\overline{Y}, \overline{X_i}) - b \tag{3.23}$$

The right-hand side can be computed as long as $\alpha_1 \dots \alpha_N$ and b are known. The values of $\alpha_1 \dots \alpha_N$ can be computed by solving the aforementioned dual optimization problem with gradient descent (cf. section 3.4.1). The value of b can be learned by computing $b = \overline{W} \cdot \Phi(\overline{Y}) = \sum_{i=1}^{N} \alpha_i \cdot K(\overline{Y}, \overline{X_i})$ for any training data point $\overline{Y}$ which lies on the separating hyperplane (i.e., the data point is a *free support vector*). It can be shown that any data point $\overline{X_j}$ for which $0 < \alpha_j < C/N$ is a free support vector. We can average the computed value of b over all such data points.

Once the model has been trained, any data point $\overline{X}$ (including out-of-sample points not included in the training data) can be scored using the following expression derived from the decision boundary of Equation 3.23:

$$\text{Score}(\overline{X}) = \sum_{i=1}^{N} \alpha_i \cdot K(\overline{X}, \overline{X_i}) - b \tag{3.24}$$

From the perspective of the modeling assumptions of the support-vector machine, a negative value of the score indicates that the data point is an outlier, whereas a positive value indicates that the data point is a non-outlier. However, one can also treat the computed value as a score, although it is sensitive to the value of C, which regulates the trade-off between outliers and inliers.

3.4.1 Solving the Dual Optimization Problem

The dual optimization problem can be solved using gradient-descent. However, during the descent, it is important to ensure that the constraints are not violated. The gradient of the dual objective function, which is $\frac{1}{2}\overline{\alpha}\, S\, \overline{\alpha}^T$, can be shown to be the N-dimensional vector $S\overline{\alpha}$. Therefore, the gradient-descent approach iteratively repeats the following steps after initializing each α_i to $1/N$:

1. $\overline{\alpha} \Leftarrow \overline{\alpha} - \eta \cdot S\overline{\alpha}$
2. Set any negative values in $\overline{\alpha}$ to 0 and any value of α_i larger than C/N to C/N.
3. Scale the vector $\overline{\alpha}$ so that $\sum_{i=1}^{N} \alpha_i = 1$.

Here, $\eta > 0$ is the learning rate. These steps are repeated to convergence. The last two steps are executed to force the current solution to (roughly) satisfy the optimization constraints although the constraints may not be fully satisfied in the early iterations because executing step 3 might again result in α_i exceeding C/N. At convergence, all the constraints will typically be satisfied. More effective steps for SVM training may be found in [128].

A number of off-the-shelf libraries such as *LIBSVM* [128] and *scikit-learn* [629] provide off-the-shelf implementations of one-class support vector machines. The *scikit-learn* code, which provides a different interface, calls the *LIBSVM* solver under the covers. It is noteworthy that *LIBSVM* uses a more sophisticated coordinate descent method rather than the simplified gradient-descent method described above. Like kernel PCA methods, one class support-vector machines have the desirable property that they can be used for arbitrary data types provided that a kernel similarity function can be defined between data objects.

3.4.2 Practical Issues

An insightful evaluation of some of these models for outlier detection in document data is provided in [384]. In particular, the main challenge associated with support-vector machines is that these methods can be sensitive to the choice of the kernels and the many hidden parameters associated with the method such as the value of C and the parameters associated with the kernel. The evaluation in [384] makes the following observations about the one-class support-vector machine:

> "However, it turns out to be surprisingly sensitive to specific choices of representation and kernel in ways which are not very transparent. For example, the method works best with binary representation as opposed to tf-idf or Hadamard representations which are known to be superior in other methods. In addition, the proper choice of a kernel is dependent on the number of features in the binary vector. Since the difference in performance is very dramatic based on these choices, this means that the method is not robust without a deeper understanding of these representation issues."

In another recent experimental evaluation of several detectors [184], the one-class SVM was the most poorly performing detector and frequently provided worse-than-random performance.

The one-class SVM assumes that the origin is a prior for the outlier class, which is not optimal even in kernel feature space. For example, a simple operation such as mean-centering the kernel matrix can have unfortunate effects. As a general rule, it is harder to use kernels in unsupervised problem settings (as compared to supervised settings) because of the natural difficulties in model selection and parameter tuning. For example, for the Gaussian kernel, the median of the pairwise distance between points provides a rough approximation to the bandwidth σ, although the value of σ is also sensitive to the data distribution and size. A second issue is that the size of the kernel matrix S, which also defines the number of terms in the dual objective function, is $O(N^2)$. For example, for a data set containing a hundred thousand points, this type of approach is not practical.

Both these problems can be partially solved using variable subsampling [32], which works particularly well in the context of unpredictable data size-sensitive parameters. The basic idea is to repeatedly sample a variable number of training points from the data, which vary between $n_{min} = 50$ and $n_{max} = 1000$. Therefore, in each training model the size of the kernel matrix is at most 1000×1000. Subsequently, all N points are scored with respect to this training model, and the scores are normalized to Z-values. This is possible in kernel support-vector machines, because out-of-sample points can be scored with the learned model. In fact, the out-of-sample scores can often be more robust because of less overfitting. In each training model constructed from the sample, a different kernel function and choice of parameters (within a reasonable range) may be used. This process can be repeated as many times as needed. The final outlier score can be reported as the average of the score across various detectors. This approach is an ensemble technique, which is discussed in detail in Chapter 6.

3.4.3 Connections to Support Vector Data Description and Other Kernel Models

The one-class SVM is intimately connected to many other kernel models. The support-vector data description (SVDD) [539] is a different kernel SVM method in which data is enclosed in a hypersphere of radius R in the transformed feature space (rather than a linear separator from the origin). The squared radius is minimized together with penalties for margin violation. The SVDD approach is closely related both to the one-class SVM of section 3.4 as well as to the kernel Mahalanobis method discussed in section 3.3.8.

The SVDD and linear SVM models are roughly[10] equivalent if the embedded points have an origin-centered spherical geometry [539]. The most common example is that of a Gaussian kernel, which embeds all points on the unit sphere. *The solutions are otherwise quite different.* For example, the degeneracy problem of one-class SVMs, which is discussed in section 3.4, is not encountered in SVDD even with mean-centered kernels. In fact, the SVDD predictions do not change by mean-centering the kernel matrix, because the origin is not assumed as a prior. On the other hand, the SVDD approach performs poorly with polynomial kernels because the data tends to be elongated in particular directions in the

[10]We use the term "roughly" because the SVDD optimization formulation (with the Gaussian kernel) can be transformed into a formulation like one-class SVM in which a normalization constraint $||\overline{W}|| = 1$ is imposed instead of including the regularizer $||\overline{W}||^2/2$ in the objective function [539]. The experimental results in [184] suggest that SVDD provides slightly better solutions than one-class SVMs with the Gaussian kernel.

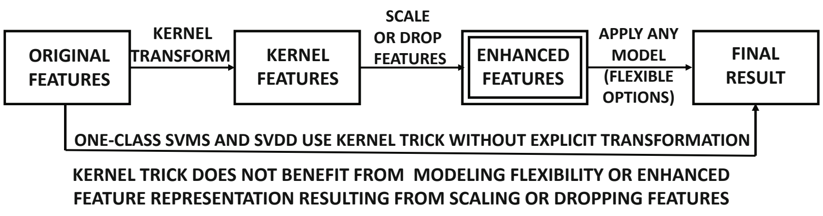

ORIGINAL FEATURES
KERNEL TRANSFORM
KERNEL FEATURES
SCALE OR DROP FEATURES
ENHANCED FEATURES
APPLY ANY MODEL (FLEXIBLE OPTIONS)
FINAL RESULT
ONE-CLASS SVMS AND SVDD USE KERNEL TRICK WITHOUT EXPLICIT TRANSFORMATION
KERNEL TRICK DOES NOT BENEFIT FROM MODELING FLEXIBILITY OR ENHANCED FEATURE REPRESENTATION RESULTING FROM SCALING OR DROPPING FEATURES

Figure 3.9: Relationship between explicit and implicit kernel transformations

transformed feature space, and a sphere fits poorly around it.

The kernel Mahalanobis method is closely related to SVDD. The former can be viewed as a way of creating a new representation of the data in which the distance to the center of the transformed data directly provides the outlier score, and there is no need to find a hard boundary of specific radius R with an optimization model. A key difference between the kernel Mahalanobis method and other one-class methods is the use of feature-space scaling of all transformed dimensions to unit variance in the former. This type of normalization is especially helpful because it prevents high-variance directions from masking the outliers. The normalization step can provide additional benefits that are not available in the original one-class SVM or SVDD solution based on the kernel trick within the dual. For example, one could very easily transform the data to unit-variance directions in feature space using the approach in section 3.3.8 and then apply a *linear* one-class SVM on the embedded representation to earn these additional benefits. In fact, the use of a circular separator (like SVDD) on the *normalized* kernel representation works reasonably well with a polynomial kernel, which is generally known to work poorly with SVDD. It is helpful to think of the nonlinear transformation in section 3.3.8 as a (normalized) *data-specific Mercer kernel map*, on top of which many trivially simple anomaly detection models (e.g., distance from centroid) become extremely effective. Furthermore, one gains the flexibility of using any other outlier detection method on the extracted representation. For example, one might use mixture modeling, distance-based methods [475], or clustering (cf. section 4.2.1 of Chapter 4) on the kernel representation. Explicit feature mapping with kernels is highly under-appreciated in spite of its obvious flexibility in enabling the use of other models, or in enhancing the underlying representation by feature scaling/selection. For example, dropping small eigenvectors leads to better accuracy in clustering and two-class SVMs [33, 481] because of de-noising effects. Similarly, dropping large eigenvectors sometimes leads to better accuracy of one-class SVMs because of outlier enhancement effects; this is simply a hard version of the softer feature scaling. Furthermore, the computational advantages of working with the dual are not guaranteed with respect to out-of-sample implementations of explicit feature mapping (see Equation 3.19); the only "disadvantage" of explicit transformation is that it renders the use of the endearing kernel trick as redundant within the SVM formulation once it has been used for creating the embedding. The relationship between explicit and implicit kernel transformation methods is illustrated in Figure 3.9.

The one-class support-vector machine may also be viewed as an approach that (roughly) finds a maximum margin separation of the data matrix D with its negative set $-D$ (instead of separating D from the origin) in kernel feature space. This intuitive view provides a clearer understanding of the relationship to two-class SVMs, and it also facilitates the adaptation of other two-class optimization models such as the Fisher's linear discriminant to the one-class setting. This view has been used to adapt the kernel Fisher discriminant method to anomaly detection [466]. The main difference between the kernel Fisher approach [466] and kernel SVMs is that the former identifies a direction that maximizes the ratio of inter-class separation to intra-class separation (between D and $-D$), whereas the latter finds the maximum margin separator between D and $-D$. However, both methods implicitly transform these data to the same kernel feature representation before optimization.

Among all these models, the kernel Mahalanobis method has the advantage of requiring the least number of free parameters (which only correspond to the kernel setting). The other methods attempt to find a *hard* boundary (cf. Figure 3.8) between outliers and inliers and therefore must use parameters like C to regulate the trade-off between outliers and inliers *early on* in the modeling. The Mahalanobis method is a soft approach that focuses on finding scores and (appropriately) leaves the hard labeling to the end, when more insight is available on the score distribution. Minimizing the number of user-driven parameters is always desirable in unsupervised problems like outlier detection especially because the underlying kernel feature representations are semantically opaque. An evaluation of the kernel Mahalanobis method in an ensemble-centric setting may be found in [35].

3.5 A Matrix Factorization View of Linear Models

PCA can be viewed as a type of matrix factorization. In order to understand this point, consider the rank-k representation of PCA from a mean-centered data matrix D.

$$D' = DP_k \tag{3.25}$$

Here, D' is the $N \times k$ reduced representation of matrix D, and P_k is a $d \times k$ matrix containing the largest k (orthonormal) eigenvectors of the covariance matrix $\Sigma = \frac{D^T D}{n}$ in its columns. As discussed earlier, D' also corresponds to $Q_k \Lambda_k$, where the rank-k diagonalization of the dot-product (i.e., similarity) matrix DD^T is given by $Q_k \Lambda_k^2 Q_k^T$. Here, Q_k is an $N \times k$ matrix containing the orthonormal eigenvectors of DD^T. Therefore, we have:

$$D' = DP_k \approx Q_k \Lambda_k \tag{3.26}$$

Interestingly, it can be shown that the matrices Q_k, Λ_k and P_k can be used to create a rank-k factorization of the original matrix D. By post-multiplying each expression in Equation 3.26 with the matrix P_k^T and setting $P_k P_k^T = I$, we obtain:

$$D \approx Q_k \Lambda_k P_k^T \tag{3.27}$$

This particular relationship is also referred to as the rank-k Singular Value Decomposition (SVD) of the data matrix D. By absorbing the diagonal matrix in one of the factors, one can express D as a factorization into two matrices, each of which have orthogonal columns. In particular, we define the $N \times k$ matrix U as $Q_k \Lambda_k$ and the $d \times k$ matrix V as P_k. Therefore, SVD can be expressed as a factorization into two low-rank matrices as follows:

$$D \approx UV^T \tag{3.28}$$

Interestingly, the matrices U and V, can be learned by solving the following optimization problem:

$$\begin{aligned} \text{Minimize } & ||D - UV^T||^2 \\ & \text{subject to:} \\ & \text{Columns of } U \text{ are mutually orthogonal} \\ & \text{Columns of } V \text{ are mutually orthonormal} \end{aligned}$$

Here, $|| \cdot ||^2$ represents the Frobenius norm of the error matrix $(D - UV^T)$. The Frobenius norm is defined as the sum of the squares of the entries. Note that this interpretation is consistent with mean-squared error optimization of PCA. It is also noteworthy that the absolute values of the entries of $D - UV^T$ provide *entry-wise* outlier scores, which are more detailed than the *row-wise* outlier scores discussed in earlier sections. These values represent the inability of the compressed representation UV^T to fully explain the values in the original matrix because of their deviations from the underlying correlation structure.

This view of SVD is particularly useful because it paves the way for using other types of matrix factorization methods. For example, the objective function can be changed in a variety of ways, regularization can be added, and the constraints can be modified to incorporate specific (desired) properties into the factor matrices U and V. The simplest generalization is one in which the orthogonality constraints on U and V are removed; this leads to *unconstrained* matrix factorization. Alternatively, for a non-negative data matrix D, we can impose non-negativity constraints on the factors to create non-negative representations, which are more interpretable. This type of factorization is very useful in text data and network data. In Chapter 12, we will provide a specific example of such a factorization. Finally, the most useful aspect of this type of matrix factorization is that it can be used for *anomaly detection in incomplete data sets*, and even provide insights about the specific *entries* of the matrix that lead to anomalies. This is not possible with the straightforward version of PCA, which is undefined for incomplete data matrices. In the next section, we will provide an example of unconstrained matrix factorization of incomplete data.

3.5.1 Outlier Detection in Incomplete Data

In this section, we will show how unconstrained matrix factorization can be used to discover anomalous rows or even anomalies *entries* of an incomplete data matrix. The latter can be useful in an application such as recommender systems. In a recommender system, we might have an $N \times d$ ratings matrix D with N users and d items. It may be desirable to discover anomalous ratings, such as the fake ratings created by “shills” in the recommender system. In such applications, each column (item) might typically contain less than 100 ratings over a population of millions of users. In such sparse settings, even the covariance matrix cannot be estimated accurately, as needed in PCA. In other applications, many data values might be missing because of weaknesses in data collection mechanism. For example, in a user survey, users might choose to leave many fields bank. In such settings, it might be desirable to discover anomalous *rows* of the data matrix. All these complex settings can be addressed with a straightforward generalization of the aforementioned PCA model.

For, the purpose of the following discussion, we will assume that the (i, j)th entry of D (when not missing) is denoted by x_{ij}. Consider a setting in which the set of entries in the matrix that are specified (i.e., not missing) is denoted by H. In other words, we have the following:

$$H = \{(i, j) : x_{ij} \text{ is observed (not missing)}\} \tag{3.29}$$

Then, we would like to factorize the data matrix D into the representation UV^T, so that $U = [u_{is}]$ is an $N \times k$ matrix, $V = [v_{js}]$ is a $d \times k$ matrix, and k is the rank of the factorization. The predicted value of an entry (i, j) is $\sum_{s=1}^{k} u_{is} v_{js}$, and we wish to minimize the aggregate error with respect to observed values. However, since we know only the subset of entries H in the data matrix, we must formulate the optimization problem *only over the specified entries.* Furthermore, it is important to use regularization to avoid overfitting because the number of specified entries might be small. This optimization problem can be written (together with a regularization term) as follows:

$$\text{Minimize } J = \underbrace{\frac{1}{2}\sum_{(i,j)\in H}(x_{ij} - \sum_{s=1}^{k} u_{is}v_{js})^2}_{\text{Error on observed entries}} + \underbrace{\frac{\alpha}{2}(||U||^2 + ||V||^2)}_{\text{Regularizer}}$$

subject to:

No constraints on U and V

Here, $\alpha > 0$ is the regularization parameter and the squared Frobenius norm on the factor matrices is included in the objective function. We have dropped the constraints on U and V to simplify the optimization process. For observed entries in D, the error e_{ij} of the (i, j)th entry is the difference between the observed value x_{ij} and the predicted values $\sum_{s=1}^{k} u_{is}v_{js}$:

$$e_{ij} = x_{ij} - \sum_{s=1}^{k} u_{is}v_{js} \tag{3.30}$$

Note that the error term is defined only for the observed entries in H. We can, therefore, rewrite the objective function J as follows:

$$J = \frac{1}{2}\sum_{(i,j)\in H} e_{ij}^2 + \frac{\alpha}{2}(||U||^2 + ||V||^2) \tag{3.31}$$

This error term is key; solving the optimization problem yields the (squared) error terms as the outlier scores of the individual *entries* in the data matrix.

The optimization problem can be solved using gradient descent. Therefore, we can compute the partial derivatives of the objective function with respect to the parameters u_{is} and v_{js}:

$$\frac{\partial J}{\partial u_{is}} = \sum_{j:(i,j)\in H} e_{ij}(-v_{js}) + \alpha \cdot u_{is}$$
$$\frac{\partial J}{\partial v_{js}} = \sum_{i:(i,j)\in H} e_{ij}(-u_{is}) + \alpha \cdot v_{js}$$

In gradient-descent, we create an $[(N + d) \cdot k]$-dimensional vector $\overline{W}$ of parameters corresponding to the entries in U and V and make the updates $\overline{W} \Leftarrow \overline{W} - \eta\nabla J$. Note that ∇J is defined by the entire vector of $(N + d) \cdot k$-dimensional vector of partial derivatives computed above. One can equivalently perform these updates with sparse matrix multiplication. Let E be an $N \times d$ sparse matrix in which only specified entries (i.e., entries in H) take on the value of e_{ij}. The missing entries take on zero values. Note that it makes sense to compute only the observed entries in E and use a sparse data structure to represent E. The aforementioned gradient-descent steps can be shown to be equivalent to the following matrix-wise updates:

$$U \Leftarrow U(1 - \alpha \cdot \eta) + \eta EV$$
$$V \Leftarrow V(1 - \alpha \cdot \eta) + \eta E^T U$$

These iterations can be executed to convergence. The value of $\eta > 0$ corresponds to the learning rate. Selecting the learning rate to be too large might result in overflows. On the other hand, a very small learning rate will lead to convergence which is too slow. A faster variant of this approach is stochastic gradient descent in which we cycle through the observed entries x_{ij} in H in random order and make the following updates:

$$\begin{aligned} u_{is} &\Leftarrow u_{is}(1 - \alpha \cdot \eta) + \eta\, e_{ij} v_{js} \quad \forall s \in \{1 \ldots k\} \\ v_{js} &\Leftarrow v_{js}(1 - \alpha \cdot \eta) + \eta\, e_{ij} u_{is} \quad \forall s \in \{1 \ldots k\} \end{aligned}$$

The resulting iterations are also executed to convergence. More details may be found in [34].

3.5.1.1 Computing the Outlier Scores

It remains to show how one can compute the entry-wise or row-wise outlier scores from the factors. Note that factorization approximated reconstructs the original data matrix (as in SVD). Deviations from the reconstructed values are presumed to be outliers. The error e_{ij} for the observed entry x_{ij} is defined according to Equation 3.30:

$$e_{ij} = x_{ij} - \sum_{s=1}^{k} u_{is} v_{js} \tag{3.32}$$

These errors are also referred to as *residuals*. Entries with large positive or negative residuals tend to be outliers because they do not conform to the normal model of low-rank factorization. Therefore, we use the squares of these entries as the outlier scores.

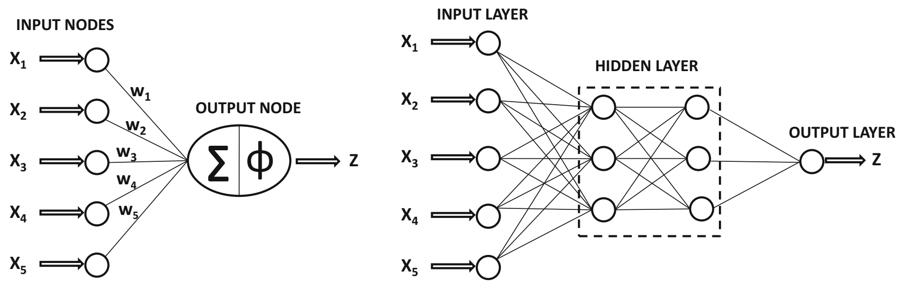
We can define the row-wise outlier score in a similar way. For any particular row of the data matrix, its outlier score is defined as the mean of the squared-residuals in its observed entries. Therefore, for the ith row $\overline{X_i}$ of D, we define its outlier score as follows:

$$\text{Score}(\overline{X_i}) = \frac{\sum_{j:(i,j) \in H} e_{ij}^2}{n_i} \tag{3.33}$$

Here, n_i denotes the number of observed entries in $\overline{X_i}$. Interestingly, it is possible to define column-wise outlier scores in an exactly analogous way. This can be useful in some applications. For example, in a collaborative filtering application, the merchant might be interested in items (columns) with unusual ratings patterns. Matrix factorization is one of the most general forms of linear modeling and it finds applications in network data as well. Some of these applications will be discussed in Chapter 12.

3.6 Neural Networks: From Linear Models to Deep Learning

Neural networks are computational learning models that simulate the human nervous system. In humans, learning is performed by changing the strength of synaptic connections between cells, which are referred to as *neurons*. The strengths of synaptic connections are changed in response to artificial stimuli. In the case of *artificial* neural networks, the individual nodes are referred to as *neurons*. The neurons receive inputs from other neurons using weighted connections (or from external training data), and might in turn transmit their outputs to other neurons after performing a computation on their inputs. Just as synaptic strengths are changed in biological neural networks for learning, artificial neural



(a) Perceptron (b) Multilayer

Figure 3.10: Single- and multilayer neural networks

networks change their weights in response to their inputs. The most basic form of input to a neuron is a feature of the training data, which can be considered the analog of the external stimulus used in biological neural networks.

The simplest form of a neural network is the *perceptron*. In its most basic form, the perceptron is virtually identical to a simple linear regression model or PCA/matrix factorization model. However, its neat conceptualization as a unit of computation allows us to put multiple perceptrons together in a multlayer network. This type of multilayer network allows the computation of any nonlinear function. For this reason, neural networks are referred to as *universal function approximators*. The perceptron contains two layers of nodes, which correspond to the input nodes and a single output node. The number of input nodes is exactly equal to the dimensionality d of the data. Let $\overline{X} = (x_1 \ldots x_d)$ be the d inputs, which correspond to the d feature values of the data records. The output of a perceptron is computed as a function of the underlying inputs with the associated weight vector $\overline{W} = (w_1 \ldots w_d)$:

$$z = \overline{W} \cdot \overline{X} = \sum_{i=1}^{d} w_i x_i \tag{3.34}$$

This function is also referred to as the *linear activation function*. It is noteworthy that the weights in the activation function are exactly analogous to those used in section 3.2 for linear regression and rank-1 PCA. In general settings, however, we can use an arbitrary activation function of this linear model and also incorporate a bias term:

$$z = \Phi(\overline{W} \cdot \overline{X} + b) \tag{3.35}$$

As we will discuss later, the activation function Φ is often a nonlinear function like the sigmoid or tanh functions. However, for now, we will work with the simpler form discussed in section 3.34 because of its similarity to the other models discussed in this chapter. A pictorial representation of the perceptron architecture is illustrated in Figure 3.10(a).

Neural networks can be used for either of the two settings discussed in section 3.2; recall that in one setting (cf. section 3.2.1) the dependent variable is treated as special, whereas in the other setting (cf. section 3.2.2) all attributes are treated in a homogeneous way and the mean-squared projection error on the hyperplane is minimized. The first type of setting is used to create *replicator neural networks* [160, 250] in which each attribute of the data is predicted using other attributes, and the errors of the prediction are used to quantify outlier scores. Such types of neural networks are also referred to as *autoencoders* and are discussed in section 3.6.2. A generalization of this approach with the use of *any* regression model (and not just neural networks) is discussed in section 7.7 of Chapter 7.

In this section, we will focus on two different types of neural networks. First, we will focus on a setting in which we are trying to create *one-class* neural networks in which the output of the network is always zero in spite of the fact that the weights are nonzero. This setting is somewhat less common and used rarely. However, we include it because it is directly related to the optimization approach discussed in section 3.2.2. In section 3.6.2, we will discuss the use of replicator neural networks and autoencoders for outlier detection. Both types of methods can be understood within the framework of dimensionality reduction methods, although the autoencoder framework is more general and potentially more powerful. This approach is used more commonly in outlier detection.

Since all training points are assumed to be normal points in the one-class setting, the prediction z of Equation 3.34 is expected to be 0. Note that this is *exactly identical* to the assumption in Equation 3.9 for linear modeling which is replicated here:

$$\sum_{i=1}^{d} w_i \cdot x_i \approx 0 \tag{3.36}$$

Therefore, any non-zero value of z predicted by the one-class neural network is assumed to be a result of outliers that do not conform to the model of normal data. Therefore, for a single instance $\overline{X_i}$, in which the neural network predicts z_i, the squared error for the ith point from our one-class assumption is as follows:

$$J_i = z_i^2 = (\overline{W} \cdot \overline{X_i})^2 \tag{3.37}$$

Therefore, one must update the weights of the neural network to account for this error. This is achieved with a gradient-descent update. This update may be written as follows:

$$\begin{aligned} \overline{W} &\Leftarrow \overline{W} - \eta \nabla J_i \\ &= \overline{W} - \eta z_i \overline{X_i} \end{aligned}$$

Here, $\eta > 0$ is the learning rate. Also, in order to avoid the trivial solution $\overline{W} = 0$, the updated vector $\overline{W}$ is scaled to unit norm. The training phase of the neural network feeds the d-dimensional records $\overline{X_1} \ldots \overline{X_N}$ to the perceptron one by one, and performs the aforementioned updates to the vector $\overline{W}$ until convergence is reached. This entire process is a version of stochastic-gradient descent, and the result would be almost identical to the solution we would obtain with PCA (cf. section 3.2.2) or with matrix factorization (cf. section 3.5) with the rank set to $(d-1)$.

In order to score a given data point $\overline{X_i}$, we use the learned model to compute its outlier score as follows:

$$\text{Score}(\overline{X_i}) = (\overline{W} \cdot \overline{X_i})^2 \tag{3.38}$$

Outliers will have larger scores. It is possible to score out-of-sample points using this model. In fact, as discussed later in section 3.6.3, out-of-sample points will have more robust outlier scores because they do not overfit the training data. The perceptron is equivalent to the use of matrix factorization or PCA with rank $(d-1)$, which is equivalent to scoring the data only along the smallest eigenvector after projecting it into the space of top-$(d-1)$ principal components. With this type of conservative approach to scoring, it is possible for all points to have outlier scores of 0 if the data set has rank strictly less than d. As a result, many true outliers will be missed. This is a classical manifestation of overfitting. Therefore, a more reasonable approach is to train the neural network using only half the points, and then score the remaining half as *out-of-sample* test points. The scores of the test points are standardized to zero mean and unit variance. The process is repeated multiple times over multiple random samples, and the scores of points are averaged. An alternative approach with the use of multiple output nodes will be discussed later.

3.6.1 Generalization to Nonlinear Models

So far, it does not seem as if one-class neural networks have achieved anything different from what we would be able to accomplish with a special case of matrix factorization or PCA. So, what is the point of this entire exercise? The major point is that the conceptualization of such models as neural network units helps us put them together in a *multilayer neural-network architecture*, which can model arbitrarily complex patterns in the underlying data. In other words, the conceptualization of a perceptron provides a black-box framework for "putting together" these simpler models into a more complex model. In fact, the generality of neural networks is even greater than the nonlinear PCA approach discussed in section 3.3.8 in terms of the types of decision boundaries that such a technique can model. Technically, given sufficient data, a multilayer neural network with a modest number of units can model virtually any one-class data distribution without making any assumptions about the shape of this distribution. As a result, neural networks are also sometimes referred to as "universal function approximators."

Multi-layer neural networks have an additional *hidden layer*, in addition to the input and output layers. The hidden layer might itself be connected in different types of topologies. A common type of topology is one in which the hidden layer has multiple layers, and the nodes in one layer feed forward into the nodes of the next layer. This is referred to as a *feed-forward network*. An example of a feed-forward network with two hidden layers is illustrated in Figure 3.10(b). Furthermore, one need not use linear functions in any of the layers. Various activation functions $\Phi(\cdot)$ (based on Equation 3.35) such as the *tanh* and *sigmoid* functions are used.

$$\Phi(z) = \frac{e^{2z}-1}{e^{2z}+1} \text{ (tanh function)}$$

$$\Phi(z) = \frac{1}{1+e^{-z}} \text{ (sigmoid function)}$$

$$\Phi(z) = \exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right) \text{ (Gaussian Radial Basis Function)}$$

In the single-layer neural network, the training process is simple because the expected output of the neuron is known to be 0. The problem in the case of the hidden layer is that we do not know what the outputs of neurons in these units should be. We only know that the final output at the output layer should be 0. In other words, some type of feedback is required from later layers to earlier layers about the expected output. This is achieved with the use of the *backpropagation* algorithm. The backpropagation algorithm has a forward phase and a backward phase, which is applied during the training of each instance. In the forward phase, the activation function is applied first in the input layer, then in the hidden layer, until the output propagates to the output layer. Then the error is computed, and the error estimates of various neurons also propagate backwards. These are then used to update the weights of various nodes. As in the case of the perceptron, it is important to scale the weights of the output node to unit norm throughout the update process. This is done in order to avoid the trivial solution where all weights in the output node are 0s. Furthermore, if W is the $h_1 \times h_2$ matrix of weights between any particular pair of hidden layers with h_1 and h_2 nodes (where $h_2 \leq h_1$), we need to impose the constraint $W^T W = I$ to avoid trivial transformations in which each node in a hidden layer corresponds to the same transformation or a zero output. These additional requirements necessitate the use of constrained gradient-descent methods, which are generally much harder.

As in the case of the perceptron, this process is applied for each training data point; furthermore, we cycle through the various training data points until convergence is reached. A detailed discussion of the backpropagation algorithm is provided in [84]. By increasing the number of layers in the network and using different types of activation functions, we can model arbitrarily complex non-linear patterns. The process of learning the parameters of a neural network with a large number of layers requires a number of specialized tricks; this class of methods is referred to as *deep learning* [223].

Reducing Representation Rank with Multiple Outputs: The aforementioned approach uses only one output node. In the case of the perceptron, this corresponds to a linear reduction of rank $(d-1)$. This is equivalent to using only the scores along the smallest eigenvector in PCA. In practice, this is too large a representation rank to obtain a meaningful reduction. One way of reducing the representation rank is to use multiple (say r) output nodes so that the output of each node is expected to be zero. This case is analogous to scoring along the r smallest eigenvectors in PCA (i.e., using a representation rank of $(d-r)$). Therefore, the error is equal to the sum of the squares of the outputs of the various nodes. In addition, we need to incorporate some constraints on the weights in the output layer. Let W be the $h \times r$ matrix of weights in the output layer, where $h \geq r$ is the number of nodes in the last hidden layer. In order to ensure mutual orthogonality and normalization of the weight vector, we need to impose the additional constraint $W^T W = I$. Such an approach will require constrained gradient descent for the weights in the output layer. This makes the optimization problem more challenging. A more satisfactory and interpretable way of achieving these goals is the use of replicator neural networks, which will be discussed in the next section.

Outlier Scoring: For any given point $\overline{X_i}$, if the multilayer network with a single output node outputs z_i, the outlier score is z_i^2. If the neural network has r outputs $z_i(1) \ldots z_i(r)$, the outlier score is $\sum_{j=1}^{r} z_i(j)^2$. Note that these scores are constructed in a very similar way to the approach used in matrix factorization and PCA.

3.6.2 Replicator Neural Networks and Deep Autoencoders

Although the approach in the previous section is intuitively appealing because of its natural relationship to supervised learning in traditional neural networks, it is rarely used. This is because of the constraints in the underlying optimization problem and the fact that it cannot be used to derive a compressed representation of the data. A more common methodology is to use *autoencoders*. An example of an autoencoder with three hidden layers is shown in Figure 3.11. Note that the number of outputs is the same as the number of inputs, and each input x_i is reconstructed to x_i' for the ith dimension. The aggregate error of reconstruction $\sum_{i=1}^{d}(x_i - x_i')^2$ over all d dimensions is summed up over all data points, and is minimized by neural network training. The point-specific error of reconstruction, which is $\sum_{i=1}^{d}(x_i - x_i')^2$, provides the outlier score of that point. The use of three hidden layers in replicator neural networks is common, and was also used in the approach discussed in [250].

Autoencoders are a natural choice for outlier detection because they are commonly used for dimensionality reduction of multidimensional data sets as an alternative to PCA or matrix factorization. Note that the number of nodes in the middle hidden layer of Figure 3.11 is much less than the number of nodes in the input layer (which is very typical), and the

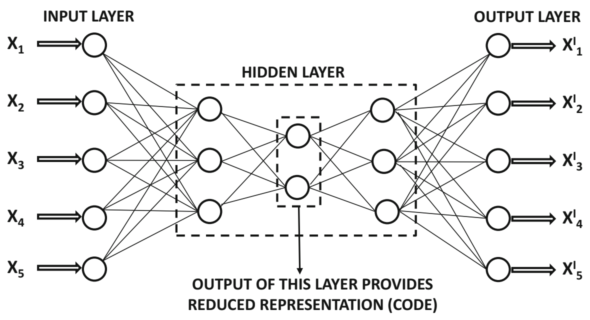
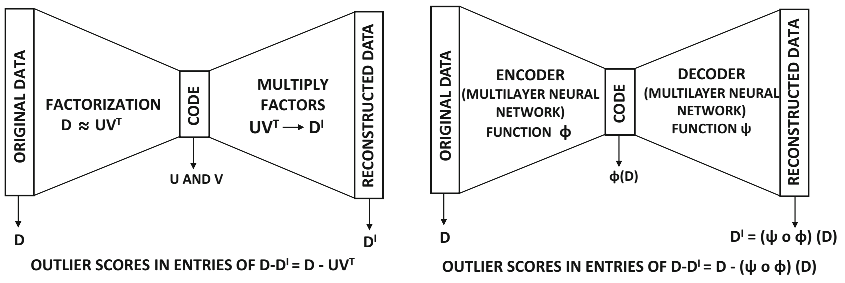


Figure 3.11: The autoencoder architecture: It is instructive to compare the autoencoder architecture with that in Figure 3.10(b).

output of the nodes in the middle hidden layer can be viewed as a reduced representation of the data. The use of neural networks for dimensionality reduction is discussed in [259, 264]. In this context, it has been shown [68] that certain simplified architectures of autoencoders yield dimensionality reductions that are closely related to those obtained using principal component analysis. Note that the architecture of the neural network on both sides of the middle hidden layer is often (but not necessarily) symmetric. In fact, one can divide the autoencoder on two sides of the middle hidden layer into two portions corresponding to an encoder and a decoder, which provides a point of view very similar to that in matrix factorization and PCA.

In order to understand why autoencoders are effective for outlier detection, we will first understand traditional matrix factorization as a method for encoding and decoding data. The factorization $D \approx UV^T$ can be viewed as a kind of encoder that compresses the data matrix D into low-rank factors U and V. Furthermore, the product UV^T provides a reconstructed matrix $D' = UV^T$. Therefore, the multiplication of U and V^T can be viewed as a type of decoder. Note that D' is not exactly the same as D and the absolute values of the entries in $(D - D')$ provide the entry-wise outlier scores. The entire process of matrix factorization (in the context of an encoder-decoder architecture) is illustrated in Figure 3.12(a).

When using PCA or matrix factorization, one makes the implicit assumption of linear compression. However, the multilayer neural network architecture provides a more general type of dimensionality reduction in which any type of nonlinear reduction is possible with the neural network model. In fact, by splitting up the replicator neural network of Figure 3.11 on two sides of the middle hidden layer, we obtain a multilayer neural network on each side of the middle hidden layer corresponding to the encoder and the decoder portions. This situation is illustrated in Figure 3.12(b). The first part of the network learns the encoding function ϕ and the second part of the neural network learns the decoding function ψ. Therefore, $\phi(D)$ represents a compressed representation (i.e, dimensionality reduction) of the data set D, and applying the decoder function ψ to $\phi(D)$ yields the reconstructed data $D' = (\psi \, o \, \phi)(D)$, which may not be exactly the same as D. Outlier entries are resistant to compression and will show the greatest variation between D and D'. The absolute values of the entries in the residual matrix $(D - D')$ provide the entry-wise outlier scores. As in the case of matrix factorization, one can convert these entry-wise scores to either row-wise scores or column-wise scores. The main difference from matrix factorization is the greater power of autoencoders in representing arbitrary data distributions. With a larger number


ORIGINAL DATA
FACTORIZATION
D ≈ UV^T
CODE
U AND V
MULTIPLY FACTORS
UV^T → D'
RECONSTRUCTED DATA
D
D'
OUTLIER SCORES IN ENTRIES OF D-D' = D - UV^T
ORIGINAL DATA
ENCODER (MULTILAYER NEURAL NETWORK) FUNCTION φ
CODE
φ(D)
DECODER (MULTILAYER NEURAL NETWORK) FUNCTION ψ
RECONSTRUCTED DATA
D
D' = (ψ o φ) (D)
OUTLIER SCORES IN ENTRIES OF D-D' = D - (ψ o φ) (D)

(a) Matrix factorization (b) Autoencoder

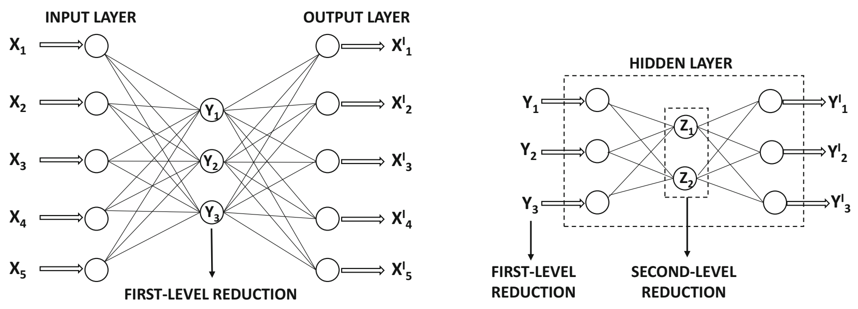
Figure 3.12: Similarities between autoencoders and dimensionality reduction/matrix factorization methods

of hidden units, one can obtain a low-dimensional representation of any data distribution whether it is linear or not. In fact, one can often model more complex distributions with autoencoders than with kernel PCA.

Although the splitting of the replicator neural network into an encoder and a decoder is useful in understanding the relationship to dimensionality reduction, it is not necessary in practice. The splitting of a replicator neural network into an encoder-decoder pair is a particular type of architecture (referred to as n/p/n-architecture) that allows the creation of a compressed representation. One can work with even more general architectures if one does not explicitly need the reduced representation of the data. In the original work on replicator neural networks [250], three hidden layers were used. The tanh activation function was used in the hidden layers, and the linear or sigmoid function is used in the output layer. The middle hidden layer used a stepwise variation on the tanh function. The error in the output layer is given by the reconstruction error and backpropagation of these errors is used to train the neural network. The trade-off between the generality of the approach and the tendency to overfit is regulated by the number of hidden layers, and the number of units in each hidden layer. The number of hidden layers and the number of hidden nodes in each layer can be determined empirically with the use of a validation set [160]. The validation set is also useful for determining the termination criterion for the training process. The training phase is terminated when the error on the validation set begins to rise.

The success of the approach follows from its ability to model complex nonlinear distributions, although one must always guard against excessively increasing the number of hidden layers and causing overfitting. As shown in [264], careful design choices can provide better reductions than PCA with neural networks. In particular, when working with very deep networks, an unsupervised method, referred to as *pretraining* [80, 459] is essential in achieving a meaningful dimensionality reduction. In pretraining, a greedy approach is used to train the network one layer at a time by learning the weights of the outer hidden layers first and then learning the weights of the inner hidden layers. The resulting weights are used as starting points for a final phase of traditional neural network backpropagation in order to fine tune them.

An example of pretraining for the network of Figure 3.11 is shown in Figure 3.13. The basic idea is to assume that the two (symmetric) outer hidden layers contain a first-level reduced representation of larger dimensionality, and the inner hidden layer contains a second level reduced representation of smaller dimensionality. Therefore, the first step is to learn the first-level reduced representation and the corresponding weights associated with the outer hidden layers using the simplified network of Figure 3.13(a). In this network, the



(a) Pretraining first-level reduction and outer weights (b) Pretraining second-level reduction and inner weights

Figure 3.13: Pretraining the neural network of Figure 3.11.

middle hidden layer is missing and the two symmetric outer hidden layers are collapsed into a single hidden layer. The assumption is that the two outer hidden layers are related to one another in a symmetric way like a smaller replicator neural network. In the second step, the reduced representation in the first step is used to learn the second-level reduced representation (and weights) of the inner hidden layers. Therefore, the inner portion of the neural network is treated as a smaller replicator neural network in its own right. Since each of these pretrained subnetworks is much smaller, the weights can be learned without much overfitting. This initial set of weights is then used to train the entire neural network of Figure 3.11 with backpropagation. Note that this process can be performed in layerwise fashion for a deep neural network containing any number of hidden layers. It is noteworthy that the symmetric autoencoder naturally constructs hierarchically related dimensionality reductions of different levels of compression.

Without pretraining, it is possible for the approach to obtain a trivial reduction, so that the reconstruction always returns the average of the training data [264]. This occurs because of overfitting in the large parameter space of a deep network. From the point of view of optimization of parameters, one can view overfitting as a process of getting stuck in local minima. Pretraining helps in conditioning the parameters towards a basin of more attractive local minima by selecting a good initialization point. These techniques are recent results in deep learning that can only be described as breakthrough advancements [186]. For example, it has been shown in [264] that one can convert a 784 pixel image into just 6 real numbers with deep autoencoders. This is not possible with PCA. Just as deep autoencoders provide better reconstruction than matrix-factorization and PCA [264], such methods are also likely to be more accurate for anomaly detection. A number of different implementations of neural networks in different domains are provided in [147, 160, 249, 552].

3.6.3 Practical Issues

There are two problematic issues with the use of neural networks. The first issue is that neural networks are slow to train. There is little that one can do to address this issue because computational complexity is an inherent problem with neural networks. Nevertheless, recent hardware and algorithmic advancements have made neural networks (in general) and deep learning (in particular) more feasible.

The second problem with neural networks is that they are sensitive to noise. In fact, since outliers are treated as normal points during the training phase, there will inevitably be errors in the model. The problem of treating outliers as normal points will be manifested as overfitting. This problem is particularly significant in the case of multilayer networks. For a sufficiently complex multilayer network, every point in the training data will be able to obtain an outlier score of 0 even when pretraining is used to avoid overfitting. However, out-of-sample points would continue to have more realistic scores because they were not included in the training process. Therefore, it is advisable to use only a random subset of the data to train the model and score the remaining points with the resulting model. The size of this random subset depends on the complexity of the multilayer network used. For more complex networks, larger training data sets are required. Therefore, one should select the complexity of the neural network depending on the amount of available training data. A natural approach is to repeatedly sample points from the training data, create the model, and score the remaining points with the resulting model. The scores of each point over various random samples (in which they are scored) are then averaged in order to provide the final result. One challenge with the use of this approach is that neural networks are slow to train and therefore it is sometimes computationally infeasible to train multiple models. Nevertheless, some recent methods like *dropout training* [510] can be leveraged to simulate ensemble performance in an efficient way without explicitly creating multiple network models.

3.6.4 The Broad Potential of Neural Networks

The aforementioned sections showed two different ways of modeling outliers in the neural network setting by using outputs with different interpretations. In practice, there are almost an unlimited number of ways in which one could use neural networks to model outliers. For example, neural networks can be used to capture various unsupervised models such as self-organizing maps [593], mixture modeling and clustering methods by appropriately defining the output nodes and corresponding optimization problem. Since methods like clustering and mixture modeling can be used for outlier detection (cf. Chapters 2 and 4), one can also adapt these neural network methods for scoring data points as outliers. Given the recent advances in deep learning, these directions represent an untapped potential of neural networks. However, one always needs to be careful to design the model in such a way so as to avoid the pervasive problem of overfitting with neural networks.

3.7 Limitations of Linear Modeling

Regression analysis has a few limitations as a tool for outlier detection. The most significant of these shortcomings was discussed at the very beginning of this chapter, in which the data-specific nature of regression analysis was explored. In particular, the data needs to be highly correlated, and aligned along lower-dimensional subspaces for regression analysis techniques to be effective. When the data is uncorrelated, but highly clustered in certain regions, such methods may not work effectively. In such cases, nonlinear models and kernel methods provide a useful choice. However, such methods are computationally intensive and may often result in overfitting.

Another related issue is that the correlations in the data may not be global in nature. A number of recent analytical observations [7] have suggested that the subspace correlations are specific to particular localities of the data. In such cases, the global subspaces found by PCA are suboptimal for outlier analysis. Therefore, it can sometimes be useful to combine linear models with proximity-models (discussed in the next chapter), in order to create more general local subspace models. This will be the topic of high-dimensional and subspace outlier detection, which is discussed in detail in Chapter 5.

As with any model-based approach, overfitting continues to be an issue, when used with a small set of data records. In this context, the relationship of the number of records to the data dimensionality is important. For example, if the number of data points are less than the dimensionality, it is possible to find one or more directions along which the variance is zero. Even for cases, where the data size is of greater (but similar) magnitude as the data dimensionality, considerable skew in the variances may be observed. This is evident from the results of Figure 3.5(c) and (d), where there is considerable skew in the eigenvalues for a small set of uniformly distributed data. This skew reduces as the data size is increased. This is a classic case of overfitting, and it is important to interpret the results carefully, when the data set sizes are small.

The interpretability of regression-based methods is rather low. These methods project the data into much lower-dimensional subspaces, which are expressed as a linear (positive or negative) combination of the original feature space. This cannot be easily interpreted in terms of physical significance in many real application. This also has the detrimental effect of reducing the intensional knowledge of the user for a particular application. This is undesirable, because it is usually interesting to be able to explain *why* a data point is an outlier in terms of the features of the original data space.

Finally, the computational complexity of the approach may be an issue when the dimensionality of the data is large. When the data has dimensionality of d, this results in an $d \times d$ covariance matrix, which may be rather large. Furthermore, the diagonalization of this matrix will slow down at least quadratically with increasing dimensionality. A number of techniques have recently been proposed, which can perform PCA in faster time than quadratic dimensionality [230]. The computational issue is particularly challenging in the case of kernel generalizations of linear methods. These issues can also be ameliorated with ensemble methods [32]. With advances in methods for matrix computation and the increasing power of computer hardware, this issue has ceased to be as much of a problem in recent years. Such dimensionality reduction techniques are now easily applied to large text collections with a dimensionality of several hundreds of thousands of words. In fact, methods like neural networks and deep learning have become increasingly feasible in recent years. If one can overcome the computational challenges associated with these methods, they can often be used to provide robust results.

3.8 Conclusions and Summary

This chapter presents linear models for outlier detection. Many data sets show significant correlations among the different attributes. In such cases, linear modeling may provide an effective tool for removing the outliers from the underlying data. In most cases, principal component analysis provides the most effective methods for outlier removal, because it is more robust to the presence of a few outliers in the data. Such methods can also be extended to nonlinear models, although the approach is computationally complex and can sometimes overfit the data. Many other mathematical models such as SVMs, matrix factorization and neural networks uses different variations of these concepts. Multilayer neural networks can model complex and nonlinear patterns, especially with the use of deep-learning methods. Other than neural networks, most of these models are global models, which do not recognize the varying subspace and correlation patterns in different data localities. However, it provides a general framework, which can be used for generalized local linear models, which are discussed in Chapters 4 and 5.

3.9 Bibliographic Survey

The relationships between the problems of regression and outlier detection has been explored extensively in the literature [467]. Outlier analysis is generally seen as an enormous challenge to robust regression in terms of the *noise* effects, and this has motivated an entire book on the subject. In many cases, the presence of outliers may lead to unstable behavior of regression analysis methods. An example of this was illustrated in Figure 3.3(b) of this chapter, where a single outlier completely changed the regression slope to one that does not reflect the true behavior of the data. It can be shown that under specific conditions, outliers can have an arbitrarily large effect on the estimation of the regression coefficients. This is also referred to as the *breakdown point* [245, 269] of regression analysis. Such circumstances are very undesirable in outlier analysis, because of the likelihood of very misleading results. Subsequently, numerous estimators have been proposed with higher breakdown points [467]. In such cases, a higher level of contamination would need to be present in the data for breakdown to occur.

The method of *principal component analysis* is also used frequently in the classical literature [296] for regression analysis and dimensionality reduction. Its application for noise correction in the text domain was first observed in [425], and then modeled theoretically in [21]. It was shown that the projection of the data points onto the hyperplanes with the greatest variance provides a data representation, with higher quality of similarity computations because of the removal of noise from the data. Latent Semantic Indexing [162, 425], a variant of PCA, was initially proposed in the context of text data for the purpose of reducing dimensionality for retrieval, rather than for noise reduction. However, many years of experience with LSI have revealed that the quality of retrieval actually improved, as was explicitly noted in [425]. Later, this was theoretically modeled for relational data [21]. PCA and LSI are dimensionality reduction techniques that summarize the data by finding linear correlations among the dimensions.

PCA-based techniques have been used in order to detect outliers in a wide variety of domains such as statistics [116], astronomy [177], ecological data [282], network intrusion detection [334, 493, 544], and many kinds of time-series data. Some of the aforementioned applications are temporal, whereas others are not. Because of the relationship between PCA and time series correlation analysis, much of the application of such regression methods has been to the temporal domain. Regression-based methods will be re-visited in Chapter 9, where a number of methods for temporal outlier analysis will be discussed. In the context of temporal data, the outlier analysis problem is closely related to the problem of *time series forecasting*, where deviations from forecasted values in a time series are flagged as outliers. A variety of regression-based methods for noise reduction and anomaly detection in time-series sensor data streams are also discussed in [22]. In addition, a number of methods that resemble structural and temporal versions of PCA have been used for anomaly detection in graphs [280, 519]. In such methods, an augmented form of the adjacency matrix, or the similarity matrix, may be used for eigenvector analysis. Such methods are commonly referred to as *spectral methods*, and are discussed in Chapter 12. Nonlinear dimensionality reduction methods are discussed in [481], and applications to novelty detection is discussed in [270, 541]. The work in [270], however, uses reconstruction error (hard kernel PCA) as the anomaly score, rather than the soft approach [35] discussed in this book. The kernel Mahalanobis method is formally proposed and tested as a distinct method in [35]. The application of such nonlinear dimensionality reductions for outlier detection is also discussed in [475]. However, the approach in [475] uses spectral methods instead of global dimensionality reduction to enhance the local nature of the reduction process. This type of approach will be more effective if different portions of the data show different manifold structures.

Another general model beyond global PCA is one in which the data is modeled as a probabilistic mixture of PCAs [549]. This is referred to as *Probabilistic PCA (PPCA).* Such methods are quite prone to noise in the underlying data during the process of mixture modeling. A method proposed in [161] increases the robustness of PCA by modeling the underlying noise in the form of a Student's t-distribution. The effects of outliers on PCA-based clustering algorithms are significant. The work in [7] provides a methods for providing the outliers as a side product of the output of the clustering algorithm. Furthermore, methods for using local PCA in outlier analysis will be discussed in detail in Chapter 5 on outlier analysis in high-dimensional data. A recent technique for using dimensionality reduction methods in high-dimensional data is provided in [591]. A complementary approach to dimensionality reduction is the RODS framework in which *sparse coding* is used [178]. Sparse coding methods transform the data to a high-dimensional and sparse representation. Outliers are defined as data points containing dictionary atoms that do not normally occur in other data points are therefore specific to the anomalies. A framework that discovers sparse codings and outliers jointly is discussed in [3].

Kernel support vector machines have been used frequently for novelty detection with the use of a one-class version of the model [480, 384]. A general discussion of support vector machines may be found in [33]. The work in [384] is particularly noteworthy because it provides an interesting evaluation of such models with respect to their performance in different data sets. One-class support-vector machines can be sensitive to the data domain, feature representations, and the choice of the kernel function. The variations and sensitivity in the performance of support-vector machines are discussed in [384, 460]. Other one-class methods for support-vector machine classification are discussed in [52, 113, 303, 384, 460, 538, 539].

In principle, any matrix factorization technique can be used for outlier analysis. An example of an outlier analysis method that uses matrix-factorization is discussed in [576]. The core principle is that dimensionality reduction methods provide an approximate representation of the data along with a corresponding set of residuals. These residuals can be used as the outlier scores. The matrix factorization methodology discussed in this chapter is commonly used in recommender systems [34].

Neural networks are discussed in detail in [84]; the problem of deep learning is discussed in [223]. The application of one-class neural networks to outlier detection is discussed in [268, 388, 389, 529]. A specific application of one-class neural networks to document classification is provided in [385]. Another class of neural networks that is commonly used is the *replicator neural network* [53, 147, 250, 567, 552], which is used to score data points as outliers. A recent implementation of replicator neural networks may be found in [160]. The use of deep-learning autoencoders for anomaly detection is explored in [53].

3.10 Exercises

1. Consider the data set of the following observations: { (1, 1), (2, 0.99), (3, 2), (4, 0.98), (5, 0.97) }. Perform a regression with Y as the dependent variable. Then perform a regression with X as the dependent variable. Why are the regression lines so different? Which point should be removed to make the regression lines more similar?

2. Perform Principal Component Analysis on the data set of Exercise 1. Determine the optimal 1-dimensional hyperplane to represent the data. Which data point is furthest from this 1-dimensional plane?

3. Remove the outlier point found in Exercise 2, and perform PCA on the remaining four points. Now project the outlier point onto the optimal regression plane. What is the value of the corrected point

4. Suppose that you have survey containing numerical data. You know that participants have occasionally made mistakes in exactly one of the fields, because it is so difficult to fill correctly. Discuss how you would detect such outliers.

5. How would your answer to the previous question change if the participants could have made mistakes in any of the fields and not just a specific field.

6. Download the *KDD CUP 1999 data set* from the UCI Machine Learning Repository [203], and perform PCA on the quantitative attributes. What is the dimensionality of the subspace required to represent (i) 80% of the variance, (ii) 95% of the variance, and (iii) 99% of the variance.

7. Repeat Exercise 6 with the use of the *Arrythmia* data set from the *UCI Machine Learning Repository* [203].

8. Generate 1000 data points randomly in 100-dimensional space, where each dimension is generated from the uniform distribution in $(0, 1)$. Repeat Exercise 6 with this data set. What happens, when you use 1,000,000 data points instead of 1000?

9. Consider a 2-dimensional data set with variables X and Y. Suppose that $Var(X) \ll Var(Y)$. How does this impact the slope of the X-on-Y regression line, as compared to the slope of the Y-on-X regression lines. Does this provide you with any insights about why one of the regression lines in Figure 3.3(b) shifts significantly compared to that in Figure 3.3(a), because of the addition of an outlier?

10. Scale each dimension of the *Arrythmia* data set, such that the variance of each dimension is 1. Repeat Exercise 7 with the scaled data set. Does the scaling process increase the number of required dimensions, or reduce them? Why? Is there any general inference that you can make about an arbitrary data set from this?

11. Let Σ be the covariance matrix of a data set. Let the Σ be diagonalized as follows:

$$\Sigma = PDP^T$$

Here, D is a diagonal matrix containing the eigenvalues λ_i, and D^{-1} is also a diagonal matrix containing the inverse of the eigenvalues (i.e. $1/\lambda_i$)

- Show that $\Sigma^{-1} = PD^{-1}P^T$
- For a given data point $\overline{X}$ from a data set with mean $\overline{\mu}$, show that the value of the Mahalanobis distance $(\overline{X} - \overline{\mu})\Sigma^{-1}(\overline{X} - \overline{\mu})^T$ between $\overline{X}$ and the mean $\overline{\mu}$ reduces to the same expression as the score in Equation 3.17.