Chapter 6
Machine Learning Systems

Devika Subramanian and Trevor A. Cohen

After reading this chapter, you should be able to answer the following questions:

- What types of problems are suitable for machine learning?
- What are the steps in the design of a machine learning workflow for a clinical prediction problem?
- What are key techniques for transforming multi-modal clinical data into a form suitable for use in machine learning?
- What limitations in model-building arise from just using observational data? How does the use of prospective data mitigate some of these limitations?
- What are some examples of biases in observational data?
- What is feature engineering and when is it required?
- When is it appropriate to use an ensemble model instead of a single global model?
- What are the challenges in deploying machine learned models in clinical decision-making settings?

D. Subramanian (✉)
Rice University, Houston, TX, USA
e-mail: devika@rice.edu

T. A. Cohen
University of Washington, Seattle, WA, USA
e-mail: cohenta@uw.edu

T. A. Cohen et al. (eds.), *Intelligent Systems in Medicine and Health*, Cognitive Informatics in Biomedicine and Healthcare,
https://doi.org/10.1007/978-3-031-09108-7_6

Identifying Problems Suited to Machine Learning

The exponential growth of a diverse set of digital health data sources (as described in Chap. 3) has opened new opportunities for data-driven modeling and decision-making in clinical medicine. These data sources including electronic health records (EHRs); large de-identified health datasets such as Medical Information Mart for Intensive Care (MIMIC) [1], which contains lab tests and time series of vital signs from Intensive Care Units (ICUs); the Cancer Genome Atlas (TCGA [2]), which includes imaging as well as proteomic and genomics data; and longitudinal, nationwide EHR datasets aggregated by companies such as Cerner. Machine learning is a critical enabling technology for analyzing and interpreting these massive data sources to support and improve clinical decision-making. Recent successes in machine learning have mainly focused on image-based predictive diagnostics: diabetic retinopathy [2], classification of skin cancer lesions [3], and identification of lymph node metastases in breast cancer [4] (see Chap. 12). Contemporary machine learning methods provide the means to go beyond these successes by exploiting the full range of data sources available today, including genotype sequencing, proteomics and other -omics data, data from wearable devices such as continuous glucose monitors, from health apps on smartphones, and from patients' social media interactions (including text data). By integrating them with other classical data sources, physicians can leverage rich, time-indexed, multimodal representations of patients. Standardized frameworks such as the **Observational Medical Outcomes Partnership (OMOP** [5])[1] common data model for encoding disparate data types, make it possible to incorporate diverse data sources into a machine learning workflow.

There are two broad classes of problems that can be solved using machine learning: (1) *prediction problems* involving probabilistic estimation of a diagnosis, outcome of a therapy modality, risk of developing a disease, or disease progression from observational patient data; and (2) *probabilistic modeling* involving estimation of joint distributions of clinical variables from observational and interventional data, which can then be used to make "what-if" inferences to answer questions such as "will adding a specific therapeutic intervention reduce risk of hospital readmission?". **Supervised machine learning** models solve prediction problems. They learn mappings between predictor variables and outcome variables from paired associational training data that take the form (predictors, outcomes). A typical example might involve assigning a diagnostic label to a radiological image (predictors: pixels; outcomes: diagnoses). Training such models requires sets of predictors labeled with the outcomes of interest. **Unsupervised machine learning** models find patterns between a collection of clinically relevant variables, without the need for explicitly labeled data. Examples of patterns include finding phenotypic clusters and dimensionality reduction by inferring latent factors of variation among a large collection of variables.

The focus of the current chapter is on supervised machine learning models. This class of machine learning models currently predominates in AIM applications for tasks such as diagnosis assignment and outcome prediction. The discussion makes

[1] https://www.ohdsi.org/data-standardization/the-common-data-model/. (accessed August 19, 2022)

assumptions about the reader's knowledge of pertinent mathematics and notational conventions. Those who lack the requisite background may wish to focus on the conceptual aspects of the discussion rather than the mathematical details.[2]

There are unique technical challenges and opportunities that arise in defining solvable problems in the healthcare context. In setting up a prediction model, one needs to consider carefully the following: (1) *The prediction target*: what is to be predicted, (2) *Technical feasibility of prediction*: is it predictable at all, (3) *Economic feasibility of prediction*: is it worth predicting in the context of the clinical workflow, and (4) *Information source selection*: from what information sources can the prediction be made.

For example, consider the problem of determining the optimal time to insert a left ventricular assist device (LVAD—a surgically implanted artificial pump that assists the heart in circulating blood throughout the body) in a patient with chronic heart failure. The standard of care for LVAD insertion defines the optimal insertion time to be when the "pumping life" of the heart is estimated to be approximately 1 year. The prediction problem is then reduced to a mortality estimation problem—what is the probability of a patient's survival for 1 year given various clinical assessments of the heart's mechanical and electrical efficacy, coupled with a broad range of laboratory assessments on the condition of other vital organs. This prediction problem is solved routinely by expert cardiologists, so there is evidence that it is a solvable problem. Unfortunately, not all LVAD insertion decisions made by experts lead to optimal patient outcomes, which opens the possibility of machine learning analysis of this decision problem. The informational basis for prediction can initially be set to all the records reviewed by the expert cardiologist in making the LVAD insertion decision. Training data can be assembled from a retrospective study of EHR records containing all the relevant information (such as arterial blood pressure, EKG findings and ultrasound studies of cardiac function) together with the correct final go/no-go insertion decision. Note that the variable to predict is the *correct* decision, not necessarily the decision made by any individual doctor. The correct decision needs to be validated with information on the patient *after* the LVAD procedure, or by an expert committee. A predictive supervised machine learning model can extract probabilistic patterns to predict appropriate times for LVAD insertion from the curated dataset, in effect summarizing the experiences of the most successful expert cardiologists.

Defining outcome and predictor variables for a prediction problem can be tricky. One problem is that information about the outcome variable can be leaked through the predictors. Consider predicting diabetic ketoacidosis (DKA)[3] in a pediatric Type 1 diabetes[4] patient based on data gathered from the EHR, including demographic

[2] Introductory material on the pertinent mathematical details (e.g., probability theory, linear algebra, calculus) is provided in the suggested readings at the end of this chapter.

[3] DKA is a serious complication of diabetes resulting from the buildup of fatty acid byproducts called ketones in the bloodstream, with dangerous increases in blood acidity if untreated.

[4] Type 1 diabetes tends to arise first in children and requires treatment with insulin. The type of disease that arises in adults, who are often overweight, is Type 2 diabetes and can often be treated with medications rather than insulin.

information, lab tests, antibody titers, insulin dosing and modality. A supervised machine learning model built from retrospective patient data predicts that high values of beta-hydroxy-butyrate are predictive of DKA. Unfortunately, beta-hydroxy-butyrate is measured only for patients with DKA and is used in monitoring and management of that condition. By using beta-hydroxy-butyrate as a predictor, the answer has inadvertently been revealed to the algorithm.

Missing values in predictor variables pose a fresh set of challenges. Not all supervised learning methods can handle missing values. Those that cannot then drop data points, leading to models built on fewer samples, which may not be statistically robust. A further complication is the nature of missing data—was a measurement randomly omitted, or omitted for a specific cohort as in the beta-hydroxy-butyrate example above? Similarly, is there limited representation of specific demographic groups in the database, reflecting human biases in data collection that could yield large errors in prediction for that subgroup? Compensating for these biases in the construction of training data sets is essential for a successful machine learning project, and a detailed account of the origin of missing values in clinical data and methods to manage them is provided in Chap. 11.

The Machine Learning Workflow: Components of a Machine Learning Solution

This chapter seeks to introduce principles and mechanics of building data-driven predictive machine learning models for healthcare applications. At the heart of the process is the clinical question that needs answering, for it drives the selection of both the data and the machine learning model. Formulating clinical questions appropriate for data-driven machine learning analysis is still an art. One typically cycles through the steps shown in Box 6.1: specification of the clinical question, data source selection, data extraction and transformation, model specification and construction, model validation, and incorporation of the model into a clinical workflow.

Box 6.1 Steps in a Typical Workflow for Data-Driven Predictive Modeling in Healthcare

- **Step 0:** Specify the clinical question that needs to be answered
- **Step 1:** Identify data sources relevant to answering that question
- **Step 2:** Extract and transform raw data from the original sources into a form needed for analysis by specific machine learning (ML) algorithms
- **Step 3:** Select a suitable algorithm and build a model, ensuring appropriate choice of algorithm hyper-parameters
- **Step 4:** Validate model predictions; checking for robustness and going back to the earlier steps in the workflow, if warranted
- **Step 5:** Incorporate model into clinical workflow with a human-centered approach, and construct system-level impact assessments

Here is a specific example of a predictive modeling workflow.

- **The clinical question**: Is the risk of developing DKA in the next 3 months for a teenage, white, pediatric Type 1 diabetes patient with an HbA1c[5] increase of 1 point (from 7 to 8) over the last 3 months high enough to warrant an intervention? Which of the two following interventions is likely to be more successful—a social worker visit to ensure insulin compliance or an increase in insulin dosing?
- **The data sources**: Taking a data-driven perspective on this question would involve retrieving, from the EHR database, pertinent information on all Type 1 diabetes patients with similar demographic and clinical profiles, and examining the percentage of times interventions were applied, and their relative success, to recommend an evidence-based solution. While modern-day database tools allow easy retrieval of records from the EHR, the definitions of similar demographic and clinical profiles must be chosen carefully with an expert endocrinologist's help. For example, should gender be included in the demographic profile? Which lab tests are the most relevant to determine similarity in clinical profiles? Are there genetic markers available to risk-stratify patients? Predictive modeling is a collaboration between expert human beings and the machine, with the human expert providing nuanced decision criteria for cohort selection and the machine performing detailed analysis on expert-defined patient cohorts.
- **Data extraction and transformation**: Transforming raw clinical data into analysis-ready data sets can be challenging. This is primarily because the goal of data collection in hospitals is to support care and to manage costs and payments, and not necessarily to enable retrospective or predictive analyses. For structured fields, values may be missing or entered incorrectly. Further, approaches to handle time-series data sampled at different time scales are needed. There can also be wide variation in formats of unstructured data such as free text clinical notes. Since most machine learning algorithms take tabular data (two-dimensional arrays) or multi-dimensional arrays as input, data must be represented in these forms to be processed by these algorithms.
- **Model specification and construction**: In this example, the goal is to quantify the statistical association (if any) between risk of DKA in the next 3 months and the demographic and clinical profile of Type 1 diabetes patients. If there are enough patients in our analysis cohort, a supervised ML algorithm can be used to identify the probabilistic patterns that relate available predictors to DKA risk. In addition, it may be of value to know if there is a relationship between patient profiles and effectiveness of a specific type of intervention. Such associational queries can be easily answered using a range of supervised learning algorithms, which are covered in this chapter. Demographic and clinical data for pediatric

[5] HbA1c stands for Hemoglobin A1c, which reflects average blood sugar levels over the past 2–3 months.

Type 1 diabetes patients can be assembled from the EHR. These can be combined with patient data on interventions and their outcomes. From such a dataset, it is possible to build a simple logistic regression model to predict the probability of success of a specific intervention given a patient's profile. More sophisticated models to capture nonlinear interactions between predictors and outcomes, such as **gradient boosted decision trees**, can also be built if there are sufficient data to build them. Ultimately, an end-to-end machine learning pipeline (whose structure is dictated by the nature of the problem/data) is built and compared against a simple baseline model (such as a logistic regression). The pipeline is refined by evaluating whether it *overfits* or *underfits* the data (see section on "Bias and Variance"), and either reduce the number of parameters in the model or add more data (data augmentation) to support robust learning of model parameters.

- **Model validation**: Once a predictive model is constructed, there is a need to assess its performance on new (as yet unseen) data. In a retrospective study, a randomly selected portion of the available data (typically 20%), called the **test set**, is set aside and the remaining 80% is used to train the model. For classification problems, it is possible to use several performance measures. A detailed presentation of some widely-used performance metrics is provided in the next section on "Evaluating machine learning models: validation metrics". These metrics can be calculated over the set-aside test data to get an unbiased estimate of the performance of the trained model. With a prospective study, a new test set can be constructed by retrieving fresh data from the EHR to evaluate the performance of the model. A very important principle in model validation is to ensure that the *training and test sets are kept separate*—that is, the test set is not inadvertently used in the training process (for example, to select algorithm parameters). Model configuration may be accomplished with a held-out subset of the **training set** that is often referred to as a **validation set**. The test set estimates performance of the model on "unseen" data. The training set can be viewed as the analog of the "homework problem set" in human learning, and the test set serves as the "exam". Clearly, using problems identical to the homework in the exam provides an overoptimistic estimate of the model's (the student's) predictive performance. The choice of evaluation metric is also key and reflects priorities in the clinical use context—e.g., is it more important to avoid false negatives (failing to predict future DKA, leading to a missed opportunity of diagnosis) or to minimize false positives (incorrectly predicting future DKA, potentially leading to overtreatment).
- **Incorporation into clinical workflow and system-level assessment**: While having a predictive model with strong performance is a necessary component, it is unfortunately not all that is needed for clinical impact. One needs to determine where to inject the model's predictions in the workflow of a pediatric endocrinologist for maximal impact on patient outcomes. Human factor considerations

play a critical role in the design of the user interface through which the model's decisions as well as its explanations are presented to the doctor (see Chap. 17). The final impact of the model can be estimated at the health system level with measures such as reduction in DKA admissions over a given time frame. While the discussion of validation here is focused on measures of accuracy, the reader is referred to Chap. 17 for a holistic account of evaluation that considers other important aspects such as system integration, usability, and eventual clinical outcomes.

Evaluating Machine Learning Models: Validation Metrics

A broad range of evaluation metrics are widely used to measure the performance of machine learning algorithms. While an exhaustive account of these metrics is not provided in this chapter, some of the most commonly applied ones are introduced, as well as some principles to consider when interpreting them. These metrics are introduced using the schematic representation of the results of a two-class classification system shown in Fig. 6.1.

Recall (Sensitivity) Recall measures the proportion of testing examples in the positive class that have been correctly identified by the model. This corresponds to the estimation of the *sensitivity* of a test in medicine (e.g., what proportion of cases of a disease in a population are detected by a laboratory test), and this term may be more familiar to a clinical audience. As is the case with some of the other metrics here, recall can be derived from the cells of a 2 × 2 table with cells corresponding to counts of correctly (true positive and true negative) and incorrectly (false positive

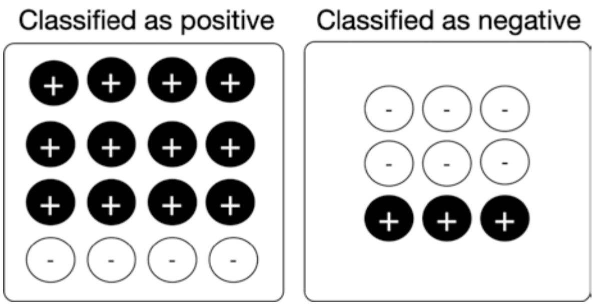


Fig. 6.1 Illustration of a possible output from a two-class classifier with an evaluation set of 25 examples, 15 of which are in the positive class (+), and 10 of which are in the negative class (−). Note that the classifier is imperfect, in that 4 members of the negative class have been classified as positive and 3 members of the positive class have been classified as negative

and false negative) classified examples. However, a straightforward way to think of recall is as the proportion of all the positive cases that were correctly identified by the model:

$$\frac{number\ of\ correctly\ classified\ positive\ examples}{total\ number\ of\ \ positive\ examples}$$

In the example in Fig. 6.1, recall would be estimated as 12/15 = 0.8.

Precision (Positive Predictive Value) It follows from the definition of recall that it is not affected by the number of negative examples that are incorrectly classified as positive cases (so-called false positives—the bottom row of the left panel in Fig. 6.1). However, it is also important to know what proportion of the ostensibly positive cases identified by a machine learning classifier were correctly identified. For example, a diagnostic system that falsely identifies many cases of a disease that then require extensive follow-up may do more harm than good, both at the individual and the societal level. Precision measures the proportion of a model's positive predictions that were true, and can be defined as:

$$\frac{number\ of\ correctly\ classified\ positive\ examples}{total\ number\ of\ \ classified\ positive\ examples}$$

Precision can be estimated from the leftmost panel of Fig. 6.1 as 12/16 = 0.75.

Precision corresponds to the *positive predictive value* of a clinical test—what proportion of people with a positive test truly have the condition it is intended to detect.

It is worth considering the trade-off between recall and precision. Given that machine learning classifiers often use a threshold (e.g., probability > 0.5) to assign discrete classes based on a probabilistic estimate, one might imagine simply setting this threshold to optimize for recall (by setting a low threshold such that most everything is classified as positive), or for precision (by setting a high threshold, such that the model only classifies examples it is very confident about as positive cases). In some circumstances, such as when the risk of missing a diagnosis is tolerable whereas the next step after automated detection is an invasive and expensive examination, precision may be more important than recall. In a screening scenario where it is desirable to detect most instances of a disease in the population, and a sequence of follow-up tests with greater precision is readily available, recall may be key. As such, the optimal performance characteristics of a machine learning model may vary depending upon how the predictions it makes will be used. Also, this trade-off suggests it may not be particularly meaningful to consider precision or recall in isolation.

The F-measure The F-measure evaluates performance by balancing precision against recall. With the most widely used variant of this measure, known as the F1

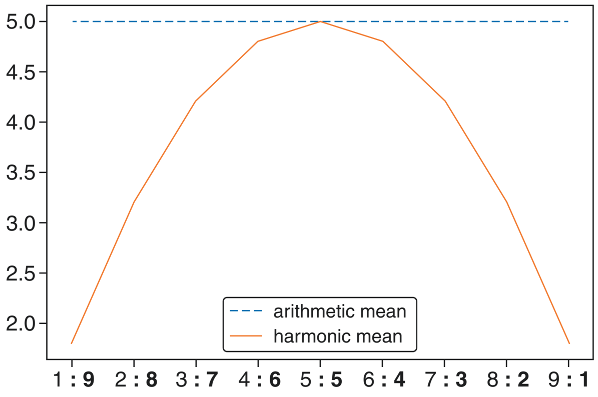


Fig. 6.2 Comparison between harmonic and arithmetic means at different ratios**.** Note that the harmonic mean is highest when the two values are equally balanced

measure, there is an equal balance between these two components. The F1 measure is the *harmonic mean*[6] between precision (p) and recall (r), which can be defined as follows:

$$F1(p,r) = \frac{2}{1/p + 1/r} = \frac{2}{(p+r)/pr} = \frac{2pr}{p+r.}$$

The implications for combining two metrics are that the harmonic mean rewards balanced combinations. The harmonic mean will be highest when p and r are relatively close together, and lowest when they are relatively far apart. Figure 6.2 illustrates that unlike the typically used arithmetic mean, the optimal harmonic mean of two values that sum to ten occurs when they are perfectly balanced. Likewise, the optimal F score will reflect a balance between precision and recall.

However, there may be tasks in which a perfect balance between precision and recall is not the optimal configuration. The F1 measure is the balanced form of the F-measure, which can be more generally formulated as follows:

$$F(p,r) = \frac{(1+\beta^2)pr}{\beta^2 p + r}$$

[6] The name ‘harmonic’ concerns the musical relationships that emerge from applying this mean to tonal frequencies. For example, the frequency of “middle C” is ~261.6 Hz, and that of the C above this ~523.3 Hz (about double). The harmonic mean of these frequencies is ~348.8 Hz, which is represented by “F” (albeit only approximately on evenly tempered keyboard instruments), the next point of harmonic progression when traversing the cycle of fourths.

The β parameter permits modification of the measure. For example, the F2 measure with $\beta = 2$ penalizes precision more heavily than recall, because the effects of precision on the denominator are multiplied by four, decreasing the overall score.

While the F measure effectively balances precision against recall (with the capability to shift emphasis), predictive models for classification generally produce a measure of confidence in their prediction—a *probability* that a test case falls into a particular category. Categories are assigned when this probability exceeds some threshold. Simply evaluating based on assigned categories discards information about the probabilistic estimates concerned. With a threshold of 0.5, a test case assigned a category-related probability of 0.6 and another assigned a probability of 0.9 would be treated identically when evaluating the model. However, it is often desirable to compare model performance across the full range of possible thresholds, without discarding these differences in a model's confidence in its predictions.

Measures Derived from Performance Curves This comparison can be accomplished by comparing the area under curves that measure performance characteristics of interest. Two widely used metrics of this nature are the area under the receiver operating characteristic curve (AUROC), and the area under the precision recall curve (AUPRC). Both of these metrics are estimated across a range of possible threshold values, effectively assessing model performance irrespective of the threshold chosen for category assignment.

The AUROC measures the area under a curve that is typically plotted as the sensitivity (recall) (y-axis) against the false positive rate—the proportion of all negative examples that have been misclassified as positive (x-axis).

In contrast the AUPRC measures the area under a curve that is typically plotted as the precision (y-axis) against the recall of a model (x-axis).

To illustrate these measures, consider a classification task with 10 positive test cases amongst 1000 in total.[7] Based on model output, the positive cases have been ranked among the 1000 cases, and the recall, precision, and false positive rate at the rank of each example is shown in Table 6.1.

Note in particular the *denominator* when calculating precision (number of predicted positives) and the false positive rate (number of negative examples). With precision, the denominator increases with recall. Moving down the ranked list of model predictions, each correctly classified positive example comes at the cost of many false positive results. On account of the class imbalance in the dataset these increases in the denominator result in substantial drops in precision with each positive example that is correctly classified. In contrast, the denominator of the false positive rate is constant, at 990—the number of negative examples. The false positive rate therefore increases gradually while working down the long list of negative examples.

[7] This presentation is inspired by Hersh's account of ranked retrieval evaluations [6].

Table 6.1 Performance characteristics of a hypothetical classifier

Rank	Recall	Precision (of predicted positives)	False positive rate (of 990 negative examples)
1	0.1 = 1/10	1.0 = 1/1 (all true positives)	0 (all true positives)
5	0.2 = 2/10	0.4 = 2/5	0.003 ≅ 3/990
15	0.3 = 3/10	0.2 = 3/15	0.012 ≅ 12/990
17	0.4 = 4/10	0.24 ≅ 4/17	0.013 ≅ 13/990
24	0.5 = 5/10	0.21 ≅ 5/24	0.019 ≅ 19/990
100	0.6 = 6/10	0.06 = 6/100	0.095 ≅ 94/990
191	0.7 = 7/10	0.04 ≅ 7/191	0.186 ≅ 184/990
300	0.8 = 8/10	0.03 ≅ 8/300	0.295 ≅ 292/990
488	0.9 = 9/10	0.02 ≅ 9/488	0.483 ≅ 479/990
1000	1.0 = 10/10	0.01 = 10/1000	1.0 = 990/990 (all false positives)

This disparity is illustrated in Fig. 6.3, in which the AUPRC and AUROC for these results are compared. In both cases the area under the curve is invariant to which value is assigned to the x-axis, and also the two graphs have a value in common—the model recall. Therefore, for illustrative purposes the assignment of axes for the AUPRC is reversed, such that recall occupies the x-axis in both graphs. To map between the graph and Table 6.1 move up the y axis (recall) while moving from top to bottom of the table. As the proportion of positive examples that are correctly classified increases, the corresponding part of the AUPRC (the area under the orange PR curve) drops precipitously as the PR curve moves rapidly leftward while the corresponding AUROC (the area under the blue ROC curve) rises gradually as the corresponding ROC curve moves slowly to the right.

Supervised Machine Learning

This section describes some of the key approaches and algorithms used in supervised machine learning. It is not intended to be an exhaustive account of these methods. More information can be found in one of the detailed and widely used textbooks of machine learning already available as resources, and several are suggested for further reading at the conclusion of this chapter. Rather, the goals in introducing these methods are first to familiarize the reader with standard nomenclature and notation used in machine learning, thereby eliminating a potential barrier to further exploration of related literature; and second to explain through illustration some fundamental concepts that relate to machine learning in general and must be understood before these methods can be applied in a principled manner. The key concepts that are developed during the course of the illustration of selected methods include the notion of a machine learning model with pliable parameters that can be fit to training data in order to make predictions; how training objectives can be configured to emphasize data points of greater predictive utility; and—of particular importance

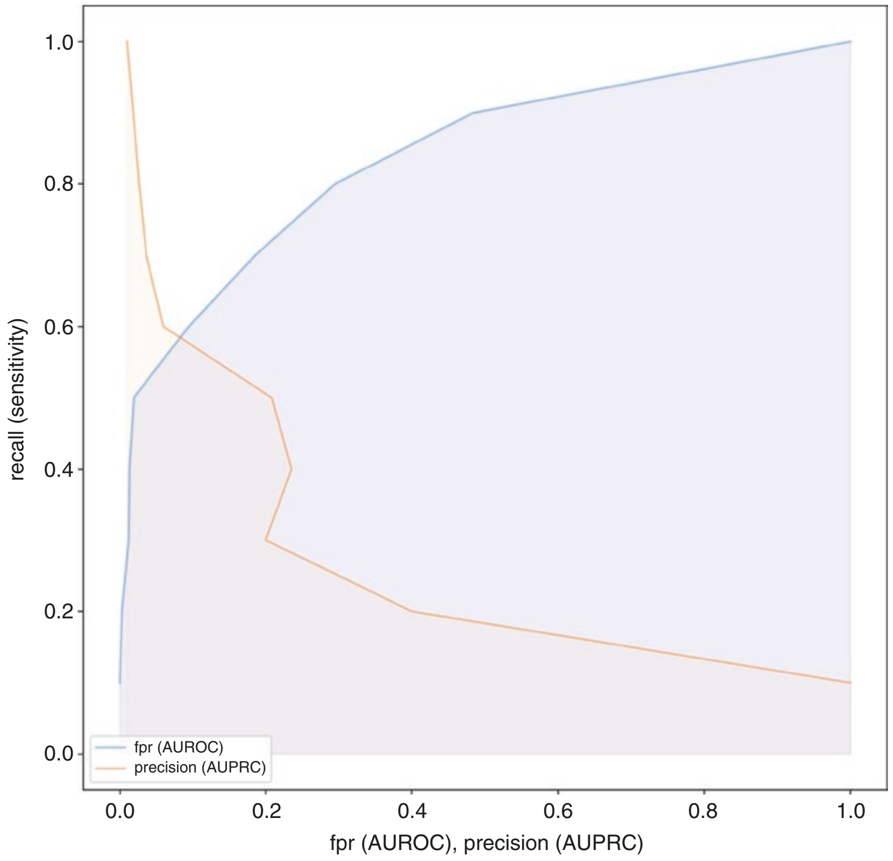


Fig. 6.3 Comparison between the AUROC and rotated AUPRC. The area under the blue curve shows the AUROC, and the area under the orange curve the AUPRC. The main contribution to the area under this orange curve occurs at low recall values (the bottom 20% of the y-axis)—precision drops precipitously as new positive examples are identified, because each of these carries the baggage of many negative examples in an imbalanced set. In contrast, the area under the blue AUROC curve continues to increase gradually throughout the full range of recall values, with larger changes to the false positive rate (fpr) delayed until around half of the positive examples have been identified

to supervised learning—the inherent trade-off between the ability of a model to conform to its training data and its ability to generalize to data outside of this training set. The general discussion offered is pertinent to applications in healthcare and biomedicine, as well as to other domains.

Machine learning is essentially an automated search for meaningful patterns in data. Traditional computational systems map inputs to outputs according to manually specified and programmed decision rules. In the clinical context, such systems are rigid and require frequent updates to accommodate changes in standard of care and/or evolution of the understanding of disease. The promise of machine learning is the automatic inference of general decision rules from lots of specific examples of decision-making in a variety of contexts. It mirrors the residency phase of training of clinicians where clinically useful rules and patterns are learned directly by observation and integrated with those learned during didactic classroom learning. With retraining on suitable new data as they emerge, a machine learning system can adapt to new experiences and new examples without the need for explicit reprogramming.

The Structure of a Supervised Machine Learning Algorithm

In order to approach the topic of machine learning, it is necessary to become familiar with some notational conventions that are standard in the field (Box 6.2). This section illustrates the use of these conventions to describe a simple machine learning algorithm. It also introduces the fundamental concept of a **loss function**, a function that measures the extent to which a machine learning model conforms to its training data, and the **gradient descent** algorithm to achieve this end.

Box 6.2 Notational Conventions

- x: a feature of a data point, such as a patient's HbA1C level
- $\mathbf{x}$: (**boldface**) a vector made up of individual features for a data point
- X: the entire set of data points, such as a set of patients
- y: a label attached to a data point, e.g., 1 indicating "developed DKA"
- Y: the set of labels for the entire set of data points
- θ: the parameters of a model, e.g., the coefficients of a regression model
- *argmin*: the arguments that minimize some function, e.g., the parameters that minimize the difference between predicted and actual values for a data set
- $||\boldsymbol{x}||$: the vertical lines indicate the length (or norm) of the vector $\boldsymbol{x}$
- $\boldsymbol{x}^{\mathrm{T}}$: the superscript "T" indicates the transpose of the vector $\boldsymbol{x}$—for example, the transpose of a row vector becomes a column vector
- $\boldsymbol{x}_1^{\mathrm{T}}\boldsymbol{x}_2$: shorthand for the scalar (or "dot") product between two vectors, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. This is calculated by multiplying the values in corresponding coordinates, and summing up the total

The simplest example of a machine learning algorithm is finding the least-squares line that fits given (x,y) points on the plane. The training data for the algorithm consists of pairs of real numbers (x,y), and the pattern or model to be found is a line represented by the equation $y = \theta_1 x + \theta_0$, where the parameters θ_1 and θ_0 stand for the slope and intercept. The parameters θ_1 and θ_0 are obtained by minimizing the mean squared prediction error of the model over the given data points. As such, this simple example serves to introduce some of the standard nomenclature used to describe machine learning approaches: a dataset composed of pairs ($\mathbf{x}$,y) where $\mathbf{x}$ is an input data point (part of a larger set X), and a corresponding output y (part of a larger set Y), and a function f(θ) characterized by some parameters θ that will be learned through application of some algorithm, so as minimize the difference between the predicted and actual outputs. This difference provides a measure of how “wrong” the model is about a data point, which can be averaged across all the points in a data set to assess overall model fit. More formally, for a dataset D = $\{(x^{(i)},y^{(i)}) \mid x^{(i)} \in \mathrm{R}$ and $y^{(i)} \in \mathrm{R}; 1 \leq i \leq m\}$ containing m pairs of real values, and the parameter pair (θ_0, θ_1) defining a model of a line, the error made by the model is specified by a loss function, which is derived from the difference between y as predicted by this equation for each observed x value ($x^{(i)}$) given model parameters θ, and the actual value of y for the data points concerned. One widely used loss function is the *mean squared error loss function* (also known as the quadratic loss function), which minimizes the average of the square of this difference across all data points in the set:

$$Loss\left(\left(\theta_0,\theta_1\right)\right) = \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - \left(\theta_0 + \theta_1 x^{(i)}\right)\right)^2$$

This loss function perspective on supervised machine learning unifies all the algorithms into a common framework. The entire family of supervised machine learning algorithms can be characterized by a loss function, a function family (in this case—the family of linear functions) for capturing the relationship between the predictors in vector **x** and the outcome y, and an optimization algorithm to find the parameters of that function—typically gradient descent to minimize the loss. Linear regression and the other algorithms introduced in this chapter all share the same structure. The differences may be in the loss function, the optimization algorithm, or the expressive power of the function family. Considering machine learning from this perspective not only provides a unified framework to support learning, but also permits communicating with machine learning researchers and practitioners with a shared terminology, a prerequisite to effective team science.

Values of θ_0 and θ_1 can be found that globally minimize this quadratic squared empirical loss function—these are the parameters that define the best fit line for the dataset D. By definition, these values will be the ones that minimize the average error in prediction across all the points in the set. These values can be found using an approach called **gradient descent**. The underlying idea is to start with a random guess, and gradually move toward a correct solution by adjusting the parameters to decrease the loss.

For a linear model with two parameters θ_0 and θ_1, a 3D visualization and contour plot of the loss function is shown in Fig. 6.4. The figures show how the loss (vertical z axis in Fig. 6.4 left and labeled blue ellipses in Fig. 6.4 right) changes as these parameters are adjusted. The loss is minimized with θ_0 and θ_1 at approximately 35 and −1 respectively. The figures plot this loss function across a broad range of parameter values. However, it would be preferable not to explore this space exhaustively in order to find a solution.

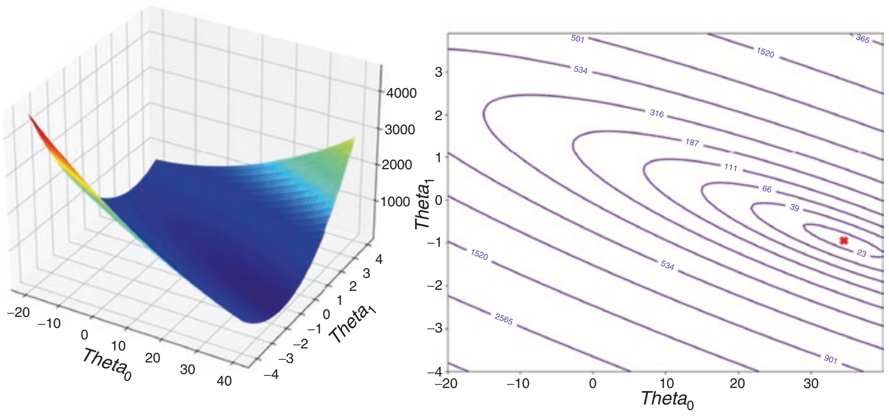


Fig. 6.4 Two views of gradient descent. These figures illustrate how the loss (vertical z axis in the left panel, labeled blue ellipses in the right panel) changes at different values of the parameters θ_0 and θ_1. The loss in this case is minimized with θ_0 and θ_1 at approximately 35 and −1, respectively, which corresponds to the lowest point on the z axis (left panel), and the point marked by the red **X** (right panel)

To find the lowest loss value (marked with a red x in the contour plot in Fig. 6.4 right), a gradient descent algorithm starts with an initial random guess for (θ_0, θ_1) and follows the direction of steepest descent of the loss function $Loss((\theta_0, \theta_1))$ in the parameter space, updating its values using

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial Loss(\theta)}{\partial \theta_j}$$

until the values stabilize, and the gradient descent process converges. Conceptually, each parameter θ_j is updated *in accordance with its influence* upon the loss function. This is estimated as the partial derivative of the loss function with respect to each parameter, because the partial derivative estimates the extent to which changing a parameter of a function will change its output. In this way, each iterative step of the algorithm serves to move the parameters closer to values that reduce the error in predicted y values for the training data. Gradient descent provably converges to the optimal solution because the loss function is a **convex function.**[8] The step size of the gradient descent algorithm is denoted α, the learning rate. It is chosen to be small enough so that the algorithm does not oscillate around the true minimum value of the loss function Loss (θ).

[8]A convex function is a function in which a line drawn between the results of evaluating the function at any two points (e.g., f(x) with x = 0.25 and x = 0.5) will lie above the result of evaluating it at any value in between (e.g., f(x) with x = 0.35). Effectively this means that the function has a single (global) minimum, and that this can be reached by following the slope of descent.

Once the parameters that lead to minimal loss have been found, the learned model can predict y for new values of x using $y = \theta_0 + \theta_1 x$. The quality of the model is assessed by measuring squared error on new x values.

Note that the specification of the linear regression problem requires more than just the training data, i.e., the (x,y) pairs. Several assumptions about the nature of these data, and a model that will fit them are made. It is assumed that the model that predicts y is linear in x and described completely by a slope and intercept parameter. The goodness of fit of the linear model is assessed by measuring averaged squared prediction error over the training data itself. That is, the strong assumption is made that the training data are a good proxy for new data that the model will predict on. Formally, it is said that the data set D is a **representative sample** of the fixed, but unknown distribution P of (x,y) example pairs—both observed and unobserved. Such an assumption is common in human learning in a classroom setting—problem sets solved by students are assumed to be a representative sample from the fixed distribution from which exam questions will be drawn. It is further assumed that each (x,y) is drawn independently, so there is no temporal dependence between the samples. This assumption is violated with time series data (see Chap. 11).

Supervised Learning: A Mathematical Formulation

The training of the model described in the previous chapter is an example of **supervised machine learning**, because the model parameters were fit to a set of data points (the x values) with labels (the y values) it learns to predict. More broadly, supervised machine learning can be conceptualized as shown in Fig. 6.5, as a search for the "best" function/pattern in the space of functions **H,** guided by a representative training sample. This view casts supervised machine learning as an **optimization problem** with a sample of (x,y) pairs drawn from an unknown but fixed distribution of examples, and a pre-defined space of functions characterizing the class of pattern relating the x's to the y's. The training data are used to navigate the space **H** of functions mapping x's to the y's to find one that "explains" the labeled training data the best. To guide the search for a suitable model in **H,** a loss function which quantifies how well the model fits the data is needed. Also needed are smoothness properties of the function space **H** to make search tractable. That is, **H** must be defined such that neighboring points in function space have similar losses with respect to the training data, so that the space can be explored systematically. The gradient descent algorithm works only when the function space **H** is smooth in this sense and supports computation of the derivative of the loss function with respect to the parameters of **H**. However, supervised machine learning problems can often be approached in this way, and gradient descent is a keystone of many contemporary machine learning approaches, including deep neural networks.

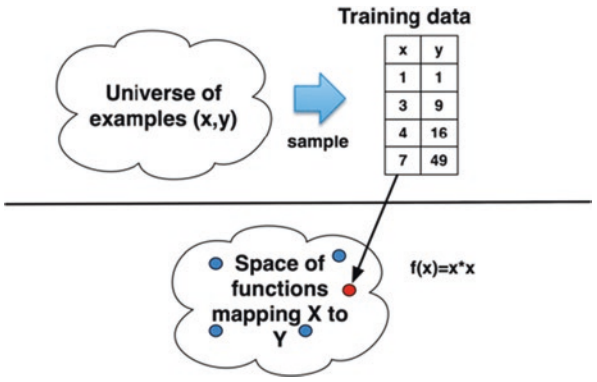


Fig. 6.5 Schematic depiction of supervised machine learning. Training data are a subset of the universe of possible labeled exampled pairs. The figure shows only a single x value per example, but in practice this is likely to consist of a vector of features, $\boldsymbol{x}$ (such as the intensities of pixel values in a radiology image). The task in supervised machine learning is to identify a function, f(x) (referred to as h^* in Box 6.3) that correctly maps the x values of the data to the y values of the labels

Box 6.3 Mathematical Components of a Supervised Machine Learning Problem

Given

- a finite data set of pairs ($\boldsymbol{x}$,y), where $\boldsymbol{x}$ is a vector of predictor variables, and y, the associated real-valued prediction
- a class of functions $\boldsymbol{H}$: $\boldsymbol{X} \rightarrow Y$ which map vectors $\boldsymbol{x}$ in $\boldsymbol{X}$ to real numbers y in Y
- a loss function L: $Y \times Y \rightarrow R$ which maps a real-valued prediction and the true value to a real number denoting the distance between them

Find

- a function h^* in $\boldsymbol{H}$ which minimizes empirical loss (hence, the argument of the minimum, or *argmin*)

$$h^* = argmin_{h \in H} \frac{1}{m} \sum_{i=1}^{m} L\left(y^{(i)}, h\left(x^{(i)}\right)\right)$$

Box 6.3 provides a compact formulation of supervised learning which reveals the ingredients needed for building a good predictive model: (1) a representative, labeled training set composed of pairs ($\boldsymbol{x}$,y), (2) a mathematical family of patterns $\boldsymbol{H}$ that potentially captures the association between $\boldsymbol{x}$ and y, and (3) a loss function L to evaluate the quality of fit between the model and the paired data. All of these must be constructed in a problem-specific way involving close collaboration between clinicians and machine learning scientists/engineers.

The class of patterns or functions $\boldsymbol{H}$ mapping the predictor variables $\boldsymbol{x}$ to an outcome variable y can be **parametric**, or **non-parametric**. Parametric models have a fixed number of parameters. The simple linear regression model on one variable has two parameters, θ_0 and θ_1, which are learned from (x,y) data by gradient descent optimization of the squared error loss function. Parametric methods make strong assumptions about the underlying function to be learned. They are computationally efficient at prediction time because they require the evaluation of a fixed parametric function (in the case of a linear regression model, each incoming feature would simply be multiplied by its respective parameter, followed by adding up of the results, and addition of the bias term θ_0).

Non-parametric methods make fewer assumptions about the nature of the underlying pattern relating the x's to the y's. A classic example of a non-parametric learning method is the k-nearest neighbors algorithm, illustrated in Fig. 6.6. To classify a new point, denoted by x in the figure, the algorithm computes the k closest points to x in the training data set, and outputs the majority vote (+ or −) among them. The method is very sensitive to the choice of distance metric as well as to the number k of neighbors chosen, with the latter illustrated in Fig. 6.6.

Augmenting Feature Representations: Basis Function Expansion

Returning to parametric models and fitting (x,y) points with a mean squared error loss function, it is possible to expand the model class to include higher order terms in x, while still retaining linearity in the parameter space. In the linear model introduced in the section on "The Structure of a Supervised Machine Learning

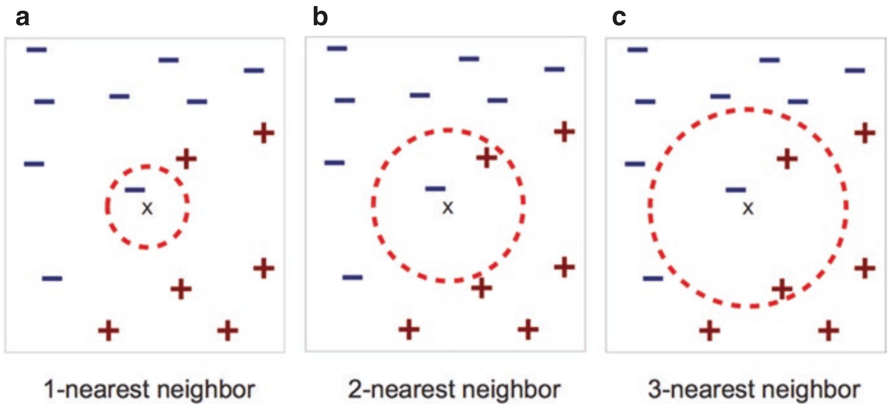


Fig. 6.6 Effect of the choice of k on the class assigned to the unseen data point x by a k-nearest neighbors classifier. With (**a**) k = 1, the negative class (−) is assigned. However, with (**b**) k = 2, the classes are tied, and with (**c**) k = 3, the positive class (+) is assigned

Algorithm", a change in the x value of a data point will always result in the same change in the y value that the model outputs. For example, if the parameter $\theta_1 = 2$ and $\theta_0 = 0$, doubling x will quadruple y. However, there are many examples in medicine (and beyond) in which the relationships between variables of interest are non-linear. Changes to an outcome of interest may be more severe once a value reaches its extremes. For example, risk of stroke increases gradually up to the age of around 65, and much more rapidly after this point [7]. The simple linear model introduced previously would underestimate the risk of stroke in older patients, because it cannot address changes in the relationship between age and risk. There are two fundamental approaches to increasing model expressivity to accommodate such relationships. One involves modifying the features that are provided to the model, and the other involves modifying the model itself. More expressive models are introduced in subsequent sections of this chapter. The discussion that follows shows how transforming the features of a data set into a more expressive *feature set* can allow a linear model to capture non-linear relationships between y (e.g., stroke risk) and x (e.g. age).

One expression of this idea is called **basis function expansion** and involves extending the feature set of a model to include higher order terms. The feature x can be expanded to the feature set x, x^2, x^3 and so forth. In the case of stroke risk, a model might be $risk\_stroke = \theta_0 + \theta_1 age + \theta_2 age^2$. The relationship between stroke risk and age would then be modeled as a weighted sum of a linear (*age*) and an exponential (age^2) function, with the parameters θ_1 and θ_2 indicating how much each of these should influence the model. While these parameters are constant once trained, the influence of the exponential component of the model will be stronger as age increases, as its contribution to the sum grows proportionately larger. With appropriately trained parameters, this model will be able to predict a sharper rise in stroke risk with increasing age accurately.

However, when applied injudiciously, basis function expansion can reduce the accuracy of model predictions on unseen examples. Consider, for example, fitting a cubic or higher-order function on given (x,y) points as shown in Fig. 6.7. These data points correspond to the pattern produced by a pneumotachogram (also known as a pneumotachograph), which measures the rate of air flow during inspiration (left part of the curve) and expiration, and is used to study lung function [8].

The model associated with the degree 3 polynomial shown at the bottom left panel is $y = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x + \theta_0$ and a gradient descent algorithm finds values for all the θ coefficients to minimize the average squared error. It is possible to enrich the class of patterns even further and select a ninth order polynomial as shown in the bottom right panel. The fitted curve passes through all the training points, and zero training error is achieved with respect to the loss function. However, the learned polynomial performs poorly outside of the training set. A small vertical jitter (shifting each data point in the y-axis) applied to the training points will result in a cubic polynomial that is not very different from the one shown in the lower left of Fig. 6.7, but the shape of the degree 9 polynomial at the bottom right will undergo radical changes.

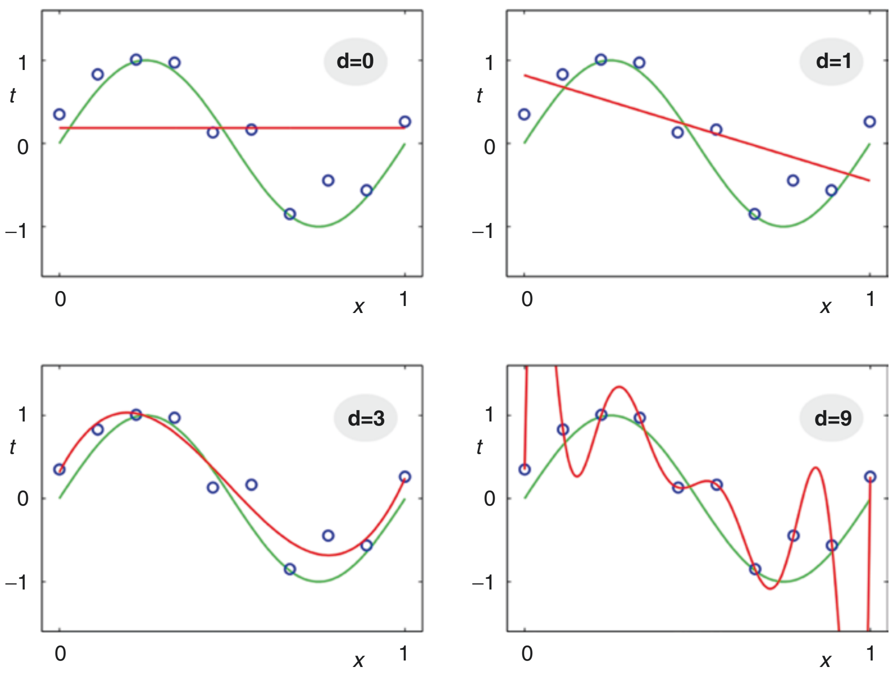


Fig. 6.7 Using basis function expansion to fit—and to overfit—a complex function with a model that is non-linear in x but linear in the expanded parameter vector θ. (Adapted from "The Elements of Statistical Learning" [9])

This figure illustrates the concept of **overfitting**—where a model has sufficient degrees of freedom to fit so tightly to the individual data points that it fails to capture the pattern of relatedness between them and is exceedingly sensitive to noise introduced to the data. Overfitting is a fundamental issue in machine learning, and one that will be returned to throughout the chapter. It is especially important in the medical domain, where datasets may be relatively small because they are limited to data from one institution or concern a rare condition. In the context of such limited data, a model with many degrees of freedom may conform closely to anomalous data points within the set that are not truly representative of the phenomenon that is being modeled and perform poorly at the point of deployment when applied to unseen data. Overfitting is also related to the tradeoff between *bias* and *variance*.

Bias and Variance

Figure 6.7 illustrates the tradeoff in machine learning between *complexity* of the model class—e.g., third vs. ninth degree polynomial (bias—which indicates the capacity of the functions in H and relates to how closely they can fit to individual data points) and *stability* of the parameter estimates (variance—which concerns how robust the estimated function is to variations in the training data). Models with high bias will have low variance because their strong assumptions render them insensitive to small changes in the data, so it is important to consider the tradeoff between bias and variance, and how they relate to sources of error. It is possible to decompose the error made by a model into two components: **structural error** (caused by limiting the class of functions considered), and **approximation error** (caused by limits on the amount of training data made available).

At the top right of Fig. 6.7, there is a model with a very strong bias (it assumes the training data can be explained by a line). The error made by this model is all accounted for by the limiting structural assumption. Providing the model with more training data will not reduce its prediction error. Such a model is called an **underfitted model**. Underfitted models have high errors on both the training and test data. At the bottom right, there is a model with many more degrees of freedom; thus, it has much lower bias. Its errors are approximation errors, caused by the limited amount of training data—there are just ten points to estimate the ten parameters of the polynomial. Such a model is an *overfitted* model. Overfitted models have low training error (because they have the freedom to fit tightly to the individual training data points) and high test error (because the tightness of the fit to a small number of potentially noisy training examples obscures the general pattern that would apply to other examples beyond the confines of the training set). The consequence is that overfitted models generalize poorly beyond their training data.

For the given collection of ten training points, the degree 3 polynomial (bottom left panel) offers a good tradeoff between structural and approximation error. The model class is powerful enough to capture the patterns in the data, and there are enough training samples to fit the model with low variance estimates. To build a successful machine learning model, one needs to find the right function class (bias) and provide a large enough training set to estimate the parameters of the learned function (variance) robustly. A family of techniques called regularization, to trade off bias and variance automatically, are introduced below. Regularization remains an important concern in machine learning, including in deep learning models, where techniques such as dropout are often a prerequisite to avoiding overfitting. The underlying principle of deliberately constraining the extent to which a model can fit to training data in order to prevent overfitting manifests in different ways in different models, but is fundamental to training models that generalize well to unseen data.

Regularization: Ridge and Lasso Regression

Regression models that are lower degree polynomials have fewer parameters and the gradient descent optimization procedure can find low variance estimates for them, even with a limited training set. However, such a model could potentially underfit the training data. Higher degree polynomials have far greater flexibility, but variance on the estimates of the optimized parameters could be high, which are signs of an overfitted model. An overfitted model, such as a degree 10 polynomial fitted on 10 (x,y) points, has parameters whose values are very large (both positive and negative).

One approach to control model complexity, then, is to penalize large (in the absolute value sense) parameter values, so that the final model has coefficients that do not grow without bound. Penalizing large parameter values by modifying the loss function used during optimization is called **regularization.**

The L2-regularized loss function for linear regression is,

$$Loss(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - h\left(x^{(i)}\right)\right)^2 + \frac{\lambda}{m}\sum_{j=1}^{d}\theta_j^2$$

so called because the L2 norm[9] is used to penalize the components of θ. Note that h(x) is the model prediction for x, and is the dot product of the parameter vector θ with [1;x]. The first term of the regularized loss function is the mean squared error of the model over the training data (as introduced in the section on "The Structure of a Supervised Machine Learning Algorithm"), which is also called the unregularized loss. The second term is the penalty function, which is the squared length of θ (the vector of parameters) without the intercept parameter θ_0. The larger the values of the parameters, the larger this term will be, which will in turn increase the value returned by the loss function. The penalty term is scaled by the factor λ which weights it relative to the unregularized loss. When λ is very low, the second term has negligible impact on the loss, so the components of θ can grow large. As λ increases, the second term dominates the loss function, and the optimizer focuses on keeping the components of θ as small as possible, ignoring the impact on the mean squared error term. As λ tends to infinity, all components of θ except for the intercept term are driven to zero. The model then simply predicts the mean of the training data for all new points x.

To choose an appropriate value of the regularization parameter λ, the training data are randomly divided into a *training set* and a *validation set* (as introduced in the section on "The Machine Learning Workflow: Components of a Machine Learning Solution"), typically in the ratio of 90/10. A sweep is conducted through potential values of λ in the log space, as shown in Fig. 6.8, to find the best value for the regularization constant—one that achieves the lowest loss over the validation set. The regularized loss is shown on the y-axis over the training and validation sets, and the natural logarithm of λ is shown in the x-axis. There is a range of λ values that are suitable for the model, and the convention is to choose the lowest value in the range. Regularization with search for the appropriate λ hyper-parameter, allows complex models to be trained on data sets without overfitting, essentially by

[9] The L2 norm gives the length of a vector from its origin and is calculated as the square root of the sum of this vector's squared components (in two dimensions this length would be that of the hypotenuse of a right-angled triangle with sides adjacent to the right angle corresponding the vector's components on the x and y axes). With L2 regularization, the sum of the squared components is used directly, without applying the square root.

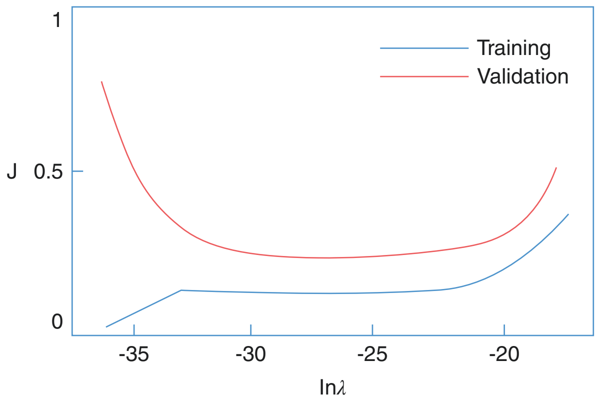


Fig. 6.8 Grid search to identify the optimal regularization parameter. The y axis shows the regularized loss J, and the x axis shows the natural logarithm of the regularization parameter, λ. Validation loss improves until a λ value of around 10^{-30}, suggesting this may be a good choice to achieve a model that generalizes well to unseen data. The extremes of the x axis suggest poorly fitted models. The leftmost extreme suggests an overfitted model, with near zero training loss with high validation loss at low λ. The rightmost extreme suggests an underfitted model, with increasing loss on both training and validation sets at high λ

limiting effective model complexity. Put another way, regularization drives higher order terms in the polynomial regression function to zero—thus, the learning procedure is given the ability to fit a ninth order polynomial, but the penalty term in the regularized loss function will drive the optimization process to select only terms no higher than degree 3, consistent with the amount of training data that are available.

L2-regularized regression is also known as ***ridge regression***. However, the penalty term in the regularized loss function need not be limited to the L2 norm of θ. A widely used penalty function is the L1-norm,[10] and the corresponding L1-regularized loss function for linear regression is

$$Loss(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - h\left(x^{(i)}\right)\right)^2 + \frac{\lambda}{m}\sum_{j=1}^{d}\left|\theta_j\right|$$

L1-regularized regression is called ***lasso regression***. Lasso regression has the special property that it drives components of θ exactly to zero [9], given sufficiently large values of the regularization parameter λ. Thus, sparse models that only involve a subset of the features in input vector **x** are obtained. Since lasso regression performs automatic feature selection, it is in wide use in clinical settings, where predictive models with the fewest number of predictors (for a given level of performance) are prized. For example, Walsh and Hripcsak describe a series of readmission risk prediction models in which lasso regression resulted in an average of a fivefold reduction in the number of features considered [10]. Lasso models can be readily

[10] The L1 norm of a vector is the sum of its absolute coordinate values.

incorporated into clinical workflow with minimal computational requirements when relevant features can be extracted in real-time directly from data sources such as electronic records [10, 11].

Linear Models for Classification

A common task in biomedical machine learning involves assigning categorical labels, such as diagnosis names or types of predicted outcomes on the basis of observed data. This task is referred to as **classification**. In classification, the outcome variable is discrete rather than continuous. There is therefore a need to modify the class of prediction functions **H** as well as the loss function to accommodate the change in outcome type. A formal description of the components of a classification problem is provided in Box 6.4.

Box 6.4 Mathematical Components of a Classification Problem
Given

- a finite data set of pairs (**x**,y), where **x** is a vector of predictor variables, and y, is the associated discrete class or category drawn from a finite set C of labels
- a class of functions **H**: **X** → C which map vectors **x** in **X** to discrete values in C
- a loss function L: C × P(C) → R which maps a true category and the predicted distribution over the categories to a real number denoting the distance between true and predicted classes

Find

- a function h* in **H** which minimizes empirical loss

$$h^* = argmin_{h \in H} \frac{1}{m} \sum_{i=1}^{m} L\left(y^{(i)}, h\left(x^{(i)}\right)\right)$$

The only difference between regression and classification is in the target of the prediction function—regression functions produce continuous value predictions, while classification functions predict discrete values, or probability distributions on a discrete value set. Thus, the class of functions **H** and the loss function L are modified to handle this change in the target of prediction. A good way to visualize a classification function h in **H** is as a partition of the input space **X** into decision regions, each associated with a member of C. Linear models of classification learn **hyperplanes** in the input space dividing it into different decision regions, while linear models with expanded basis functions learn non-linear decision boundaries.

A very special case of classification is **binary classification** when the set C consists of exactly two elements {0,1} called the negative and positive class respectively. The goal of learning in binary classification is to find the best decision rule that predicts the correct class given the vector **x** of predictors.

Classification problems are frequent targets of machine learning applications in medicine, with typical examples including classifying patients by diagnosis on the basis of their imaging data (e.g. detecting diabetic retinopathy in retinal images [2]; for further examples see Chap. 12), classifying clinical notes with respect to the nature of their content (e.g. identifying clinical notes containing goals-of-care discussions [12]; see also *text classification*, Chap. 7, section "Overview of Biomedical NLP Tasks"), and predicting clinical outcomes (e.g. readmission within 30 days of discharge [10]; for further examples see Chap. 11).

Linear models of classification come in two flavors, based on whether they learn the posterior distribution P(y|**x**) (e.g. the probability of a side effect after some drug has been observed) or the joint distribution P(**x**y) (e.g. the overall probability of both the drug and side effect being observed together) from the paired (**x**, y) training data. The difference between these estimates may not be obvious at first. For a given binary predictor, $\mathbf{x_i} \in$ {True,False} (e.g. presence of a drug), the posterior distribution P(y|$\mathbf{x_i}$) will correspond to the proportion of observations of $\mathbf{x_i}$ in which y (e.g. presence of a side effect) is also true. In contrast, the joint distribution P($\mathbf{x_i}$y) corresponds to the proportion of *all examples* in which both $\mathbf{x_i}$ and y are true. In the example, P($\mathbf{x_i}$y) would be low when the side effect in question occurs many times *without* the drug being taken—but P(y|$\mathbf{x_i}$) may still be high if the side effect occurs frequently in cases where the drug has been taken. Models that learn the posterior distribution are called **discriminative models**, while models that learn the joint distribution are called **generative models**.

Discriminative Models: Logistic Regression

A classic example of a linear discriminative binary classification model is **logistic regression**. Rather than predicting an unbounded value as with linear regression, logistic regression models P(y|**x**), the posterior distribution of the binary outcome y given input **x** as the following function: a linear computation (the dot product of a parameter vector θ with the input vector **x,** $\theta^T\mathbf{x}$) followed by a non-linear "squashing" of that dot product into the range [0,1] to represent a probability. This can be interpreted as the probability of a class of interest, such as a diagnosis or outcome.

$$P\left(y = 1|x|;\theta\right) = g\left(\theta^T x\right) = \frac{1}{1+\exp\left(-\theta^T x\right)}$$

The particular nonlinear squashing function *g* used in logistic regression is called the **sigmoid**. As shall be seen later, this function is a fundamental building block of deep neural networks. Given a training data set D, and an appropriate loss function, the optimal value of the parameter vector θ which characterizes the classification function can be found. Given a new vector **x**, the posterior probability of y = 1 given **x**, i.e., P(y = 1|**x**), is evaluated, and if that probability is greater than or equal to 0.5, the new **x** is classified as positive (belong to class 1, which may indicate a diagnosis or outcome of interest). Since the sigmoid function g(z) equals 0.5 when z is 0, the decision boundary separating class 0 from class 1 is defined by the hyperplane

$$\theta^T x = 0$$

The equation above defines a linear separating hyperplane for binary classification. When $\theta^T x \geq 0$, **x** lies on the positive side of the plane, and when $\theta^T x < 0$, **x** lies on the negative side (it is worth noting that the use of a bias term θ_0 allows the model flexibility in setting a threshold for classification—for $\theta^T x$ to equal zero, $\theta_{1\ldots n}{}^T \mathbf{x}_{1\ldots n} = -\theta_0$). The decision boundary learned by logistic regression on a nearly linearly separable data set composed of (x, y) pairs where each x is a point on a plane, is shown in Fig. 6.9. Points belonging to the positive class are marked with a + sign, while points belonging to the negative class are marked with a – sign.

The loss function for training a logistic model can be derived by the maximum likelihood principle, in which a model is trained to maximize the probability of the

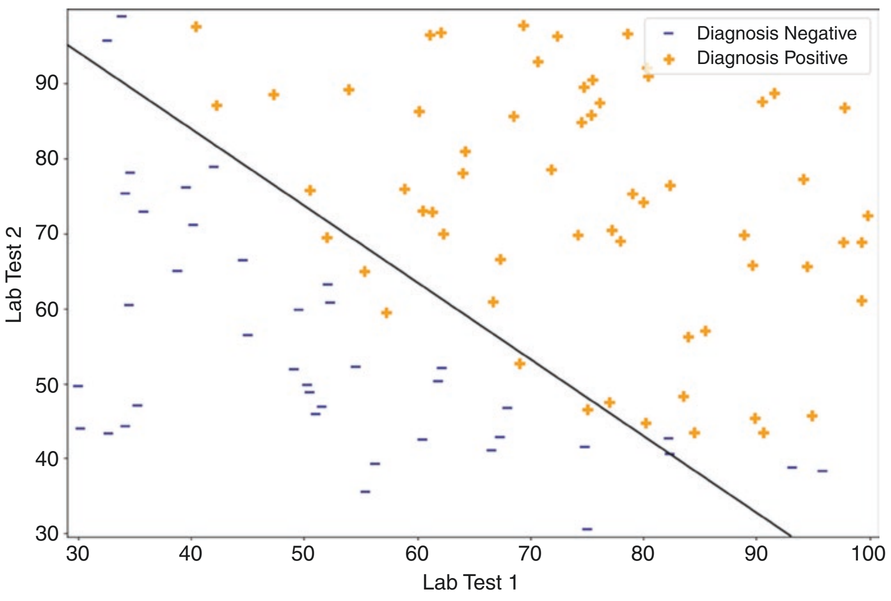


Fig. 6.9 Illustration of a decision boundary for a logistic regression classifier with two classes. Examples that fall to the right of the boundary ($\theta^T x > 0$) corresponding to estimated P(y|**x**) > 0.5), are classified as positive

observed data, assuming that $P(y = 1|x;\theta) = h(x) = g(\theta^T x)$. The logarithm of the *negative likelihood* of data set D composed of (**x**,y) pairs, where y is in the set {0,1} can be shown to be the cross entropy function.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h\left(x^{(i)}\right)\right)\right] + \left[\left(1-y^{(i)}\right)\log\left(1-h\left(x^{(i)}\right)\right)\right]$$

As this function is not only fundamental to logistic regression but is also widely used in training deep neural networks amongst other models, it is worth unpacking here. $\frac{1}{m}\sum_{i=1}^{m}$ indicates that an average over m data points is taken, and the rest of the right-hand side of the equation is a compact way of describing the log likelihood of the data. If $y^{(i)}$, the label of the example i, is equal to one, only the leftmost term is considered, because $(1 - y^{(i)})$ in the rightmost term will be zero. In this case, the likelihood will be the log of the predicted probability h(x) = *P(y = 1|x)* assigned to this example. Conversely, with $y^{(i)} = 0$, only the rightmost term will be considered, and the likelihood will be the log of the predicted probability *P(y = 0),* or *1 − P(y = 1). So, the log likelihood of the dataset is the average of the logs of the predicted probabilities of the correct labels across all examples*. The negative log likelihood reverses the polarity of this estimate, converting a maximization problem into a minimization one.

Gradient descent minimizes $J(\theta)$ to find the optimal value of the parameter vector θ. Since the cross-entropy function $J(\theta)$ is a convex function, it has a global minimum which can be computed by standard optimization algorithms. It is therefore guaranteed that θ found by minimizing the cross-entropy function represents the optimal classifier for the data set in the infinite space of parametric functions **H**.

Regularized Logistic Regression: Ridge and Lasso Models

The decision boundary in Fig. 6.9 applies readily to situations in which high values of the tests concerned indicate a diagnosis. However, circumstances may arise in which both high and low values of a laboratory test have implications for the prediction at hand. For example both high and low white cell counts may portend adverse outcomes. A range of modeling approaches that apply to classification in these circumstances are discussed in the section on "Non-linear Models". For the current discussion, it is noteworthy that basis function expansion—the same approach that was introduced as a way to augment feature representations to model non-linear functions with linear regression in the section on "Bias and Variance"—is also applicable to classification problems when logistic regression is used.

When a data set is not linearly separable in the (x_1, x_2) plane as shown in Fig. 6.10, it is possible use the basis function expansion trick to expand the space of predictors. Each point (x_1, x_2) could be mapped into a 15-dimensional space of all sixth-order polynomial combinations of $x_1$and x_2, to learn a linear separating hyperplane

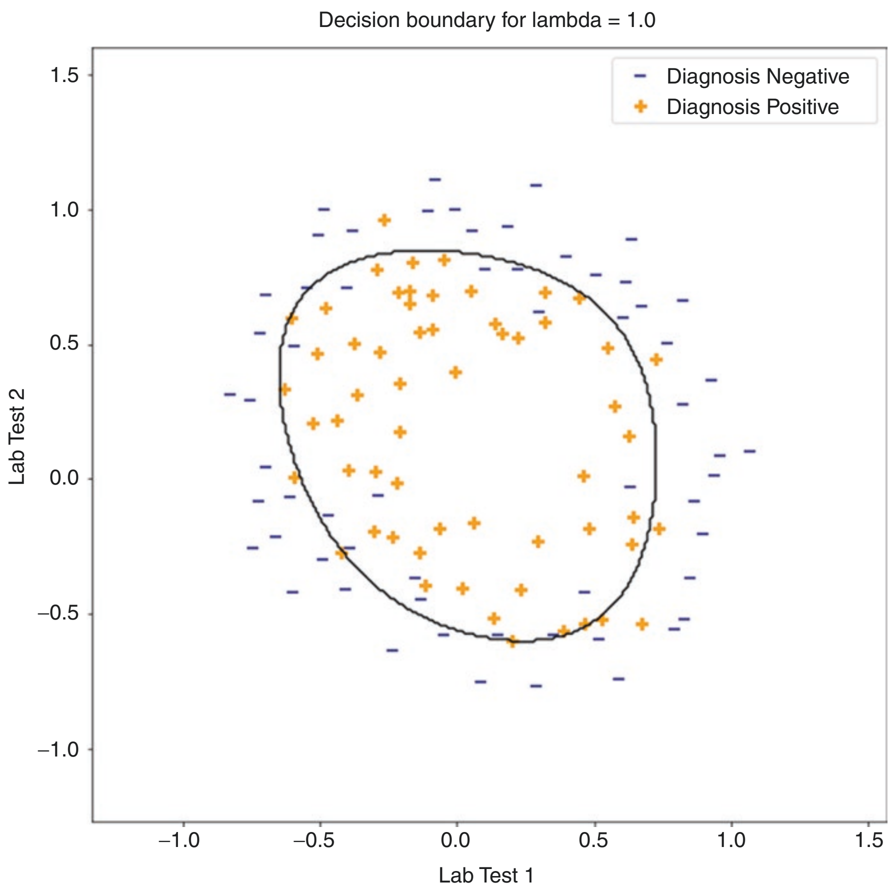


Fig. 6.10 Using feature expansion to learn non-linear decision boundaries with a (linear) logistic regression model. The black ovoid boundary is the decision boundary learned by the classifier, which separates the positive (+) and negative (−) classes reasonably cleanly, despite these classes not being linearly separable in the plane

in that 15-dimensional space. One benefit of this expansion is that it allows the model to consider the magnitude of a feature value out of context of its polarity—the polynomial expansion x^2 will be high for both highly negative and highly positive values of *x,* enabling the model to learn decision boundaries that are ovoid or circular in relation to the unexpanded features. The projection of that decision boundary in two dimensions is shown in Fig. 6.10. Basis function expansion allows us to consider rich models with low bias; therefore, to prevent overfitting it is necessary to strongly regularize the models. Ridge and lasso penalty terms are added to the cross-entropy function, just as in linear regression, to control the growth of the parameter vector.

Analogously to regularized linear regression, L2-regularized ridge logistic regression is characterized by the following loss function

$$J(\theta)=\left[-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h\left(x^{(i)}\right)\right)\right]+\left[\left(1-y^{(i)}\right)\log\left(1-h\left(x^{(i)}\right)\right)\right]\right]+\frac{\lambda}{m}\sum_{j=1}^{d}\theta_j^2$$

while L1-regularized lasso logistic regression penalizes the parameter vector θ using the absolute value.

$$J(\theta)=\left[-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h\left(x^{(i)}\right)\right)\right]+\left[\left(1-y^{(i)}\right)\log\left(1-h\left(x^{(i)}\right)\right)\right]\right]+\frac{\lambda}{m}\sum_{j=1}^{d}\left|\theta_j\right|$$

Note that in both L1 and L2 regularization, the intercept component θ_0 is not penalized because the intercept accounts for the overall mean of the data points, effectively setting the threshold for classification. The best value of the regularization constant λ can be determined by a cross-validation procedure as was done in the case of linear regression. By varying λ, it is possible to adjust the relative importance of minimizing error on the training data (first term of $J(\theta)$, representing the averaged log likelihood) and driving the coefficients of θ to zero (second term of $J(\theta)$, representing the penalty on the coefficients of θ). As λ approaches zero (no regularization), the solution found by the optimizer is very likely overfitted, especially when the number of training data points is small. As λ approaches infinity, the training data are ignored, and the coefficients of θ are driven to zero, leading to an underfitted model, such as the one shown in Fig. 6.11.

A Simple Clinical Example of Logistic Regression

This example is derived from data associating male lung cancer and smoking [13]. There is one binary predictor: whether someone is a smoker or not, and the outcome is also discrete with two values: cancer, or no-cancer. For this simple problem, it is easier to present summary statistics of the data as shown below—such a table is called a **contingency table**.

	Lung-cancer	No-cancer
Smoker	647	622
Non-smoker	2	27

The structure of the logistic model to predict the probability of cancer given smoking status is

$$\begin{aligned} p = P\left(cancer = 1 \mid smoking;\theta_0,\theta_1\right) &= g\left(\theta_0+\theta_1 smoking\right) \\ &= \frac{1}{1+\exp\left(-\left(\theta_0+\theta_1 smoking\right)\right)} \end{aligned}$$

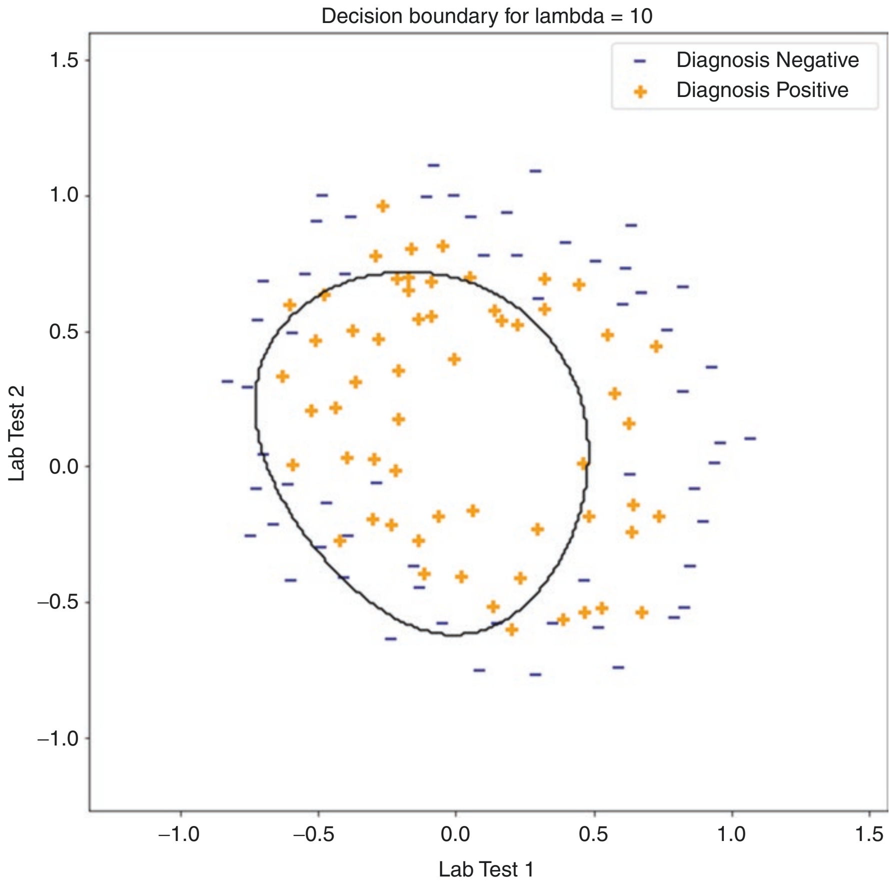


Fig. 6.11 Underfitting of a logistic regression model on account of excessive regularization. The black ovoid boundary is the decision boundary learned by the classifier, which does not cleanly separate the positive (+) and negative (−) classes

This model represents the natural logarithm of the odds ratio of cancer given smoking as a linear function of smoking.

$$\log\left(\frac{p}{1-p}\right) = \theta_0 + \theta_1 smoking$$

The parameter θ_0 represents the baseline log odds of getting cancer regardless of smoking status, and θ_1 characterizes the increment in log odds of getting cancer for the smoking cohort. Unregularized logistic regression can be used, since there is only one predictor and 1298 examples, so there is no need to penalize the loss function. The optimizer finds the values for the parameter vector θ shown below.

$$\log\left(\frac{p}{1-p}\right) = -2.6025 + 2.6419\, smoking$$

This model states that the log odds of developing cancer for non-smokers is −2.6025 (log (2/27)), while the incremental odds of developing cancer for smokers is 2.6419. That is, for a smoker, the predicted log odds of cancer goes up by 2.6419 over the baseline odds, −2.6025 + 2.6419, which is 0.039 (or log (647/622)). The estimated odds ratio of cancer for smokers versus non-smokers is $e^{2.6419} = 14.04$. That is, smokers are 14 times more likely to get cancer than non-smokers. The mechanics of logistic regression and the probabilistic interpretation of the parameter vector are revealed in this example, where summary statistics of the data suffice to estimate the parameters.

A Multivariate Clinical Example of Logistic Regression

Now consider the more complex problem of predicting whether or not a Type 2 diabetes patient's condition worsens over the course of a year based on a set of 10 baseline predictors: {age, gender, body mass index (bmi), average blood pressure (bp), and six blood serum measurements: total serum cholesterol (tc), low-density lipoprotein (ldl), high-density lipoprotein (hdl), total cholesterol/hdl (tch), logarithm of triglyceride level (ltg), blood glucose level (glu)}. These data for 442 patients along with the outcome variable, which is a quantitative measure of disease progression 1 year after the baseline are publicly available.[11] For this example, a discrete outcome variable y will be defined by labeling patients whose progression evaluations are more than one standard deviation above the cohort mean as positive ($y = 1$), and the others as negative ($y = 0$). That is, the model to be estimated has the form

$$P(y=1|\mathrm{x};\theta) = g\left(\theta^T \mathrm{x}\right) = \frac{1}{1+\exp\left(-\theta^T \mathrm{x}\right)}$$

where **x** is the 11-element vector [1,age,gender,bmi, bp,tc,ldl,hdl,tch,ltg,glu] and the parameter vector θ has 11 components, the first being the intercept (θ_0), and the other 10 associated with the predictor variables in **x** ($\theta_{1\ldots10}$). An L1-regularized logistic model is learned, finding the optimal choice for λ, the regularization parameter, by five-fold cross-validation. The model learned has only five non-zero coefficients, with bmi, ltg, and bp being the most significant coefficients in the model.

$$P(y=1|\mathrm{x};\theta) = g\left(\theta^T \mathrm{x}\right) =$$
$$\mathrm{g}\left(-1.89 + 18.29\, bmi + 9.3\,\mathrm{bp} + 1.66\, hdl + 11.91\, ltg + 1.38\, glu\right)$$

[11] https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html (accessed August 19, 2022).

The five-fold cross-validated AUROC of this model is 0.9 ± 0.05, indicating that it will predict well on new patients, if they are drawn from the same distribution of patients as the training set. Beyond accuracy, the model provides *interpretability* (see Chap. 8): the coefficients indicate both the magnitude and the direction of each parameter's influence on model predictions, albeit with the important caveat that unlike the binary feature in the previous example, the incoming features may be represented on different scales, which will affect the coefficients also.

Regularized logistic regression models are widely used for risk stratification in clinical settings. For example, a multivariate L1-regularized logistic regression model has been used to predict SARS-Cov2-related death within a year using ten baseline characteristics: gender, age, race, ethnicity, body mass index, Charlson comorbidity index, diabetes, chronic kidney disease, congestive heart failure and the Care Assessment Need score [14]. The model was trained on over 7.6 million Veteran's Affairs enrollees with training data obtained from the period May 21 to September 2020, and a testing cohort obtained from the period October 1 to November 2, 2020. The AUROC of the model on the test data was 0.836, outperforming a simple age-based stratification strategy with an AUROC of 0.74. The model was learned from structured data readily extractable from EHR records. Further, the model is easily interpretable since the coefficients of the ten predictors serve as the log-odds ratio of the effect of that predictor on the final outcome. The model was ultimately integrated into the clinical workflow with a built-in web-based risk calculator and used for prioritizing vaccinations among veterans. The model is estimated to have prevented 63.5% of deaths that would occur by the time 50% of VA enrollees are vaccinated. The model also adheres to the four ethical principles outlined by the Advisory Committee on Immunization Practices [15]. It maximizes benefits (by targeting those at highest risk for vaccine allocation), promotes justice (by identifying older adults or those with a high comorbidity burden who will require focused outreach for vaccination), mitigates health inequities (by assigning higher priority to racial and ethnic minorities directly reflecting their higher risk of mortality), and promotes transparency (by using an evidence-based model with explicit parameters). The use of ethnicity as a variable in this model compensates for known differences in health risk across populations, making a preventative intervention more readily available, without the benefit of additional contextual knowledge such as socio-economic status and other specific risk factors. Further discussion of the ethical implications of the use of such variables in predictive models is presented in Chap. 18.

Generative Models: Gaussian Discriminant Analysis

Generative models learn the full joint distribution P($\boldsymbol{x}y$) from training data pairs ($\boldsymbol{x}$,y) where $\boldsymbol{x}$ is a vector of predictor variables and y, a discrete outcome. For binary classification problems, y takes on one of two values {0,1}, while for multiclass classification problems, y can take some finite number, greater than two, of values.

Since $P(xy) = P(x|y)P(y)$ using probability theory, the following generative model component distributions are estimated.

- prior probabilities: $P(y = 1)$, $(P(y = 0) = 1 - P(y = 1))$
- class conditional distributions: $P(\boldsymbol{x}|y = 1)$, $P(\boldsymbol{x}|y = 0)$

from the training data. Armed with the component distributions, the information to generate new examples from both classes is available. To generate a new positive example, a random **x** from the distribution $P(\boldsymbol{x}|y = 1)$ is drawn. To generate a new negative example, the distribution $P(\boldsymbol{x}|y = 0)$ is used. This is why the model is called 'generative'—it can construct new examples from both classes. A discriminative model, which only estimates $P(y = 1|\boldsymbol{x})$ from training data, can classify a new example, but it cannot construct one *de novo*. Furthermore, unlike discriminative models, generative models learn the distributions of individual feature inputs, permitting one to pose questions such as "what are the characteristics of a patient with worsening type 2 Diabetes". This has inherent advantages for the generation of synthetic datasets and can support the construction of causal models that capture cause-effect relationships between individual variables (see Chap. 10). However, these capabilities come at a cost in that large amounts of data are required to correctly estimate the prerequisite distributions, which often cannot be robustly estimated with the relatively small datasets used for clinical machine learning.

Parametric generative models make assumptions about the (parametric) form of the prior probability distribution and the class conditional distributions. A common choice for the prior probabilities for binary classification problems is the Bernoulli distribution (the distribution used for modeling a coin toss), and for continuous predictors **x**, the class conditional densities are modeled as multivariate Gaussian distributions with a mean and covariance for class y = 0 and for class y = 1. These two assumptions characterize Gaussian discriminant analysis (GDA); one of the simplest parametric generative models in the field. A simple two-class example of GDA with two-dimensional **x** vectors is shown below. The learned multivariate normal class conditional densities associated with the classes are drawn as ellipses. Each ellipse is a contour plot of a two-dimensional Gaussian distribution learned from data, representing an iso-probability line. The center of the ellipses is the mean, and the shape of the ellipse is determined by the covariance matrix of the two-dimensional Gaussian. The means of the two multivariate normals in Fig. 6.12 for the two classes are different, but their covariances are the same.

The decision boundary between the two classes is computed using Bayes rule with the learned prior and class conditional distributions (the negative class distribution is represented in the denominator of the equation, which indicates the probability of x summed across both possible values {0,1} of the label y).

$$P(y=1|x)=\frac{P(x|y=1)P(y=1)}{\sum_y P(x|y)P(y)}$$

Readers with a clinical or biostatistics background may be familiar with Bayes rule applied in this way, as it provides a means to convert the *sensitivity* of a test—$P(x|y = 1)$, where $y = 1$ indicates a positive case in a population, and x indicates a

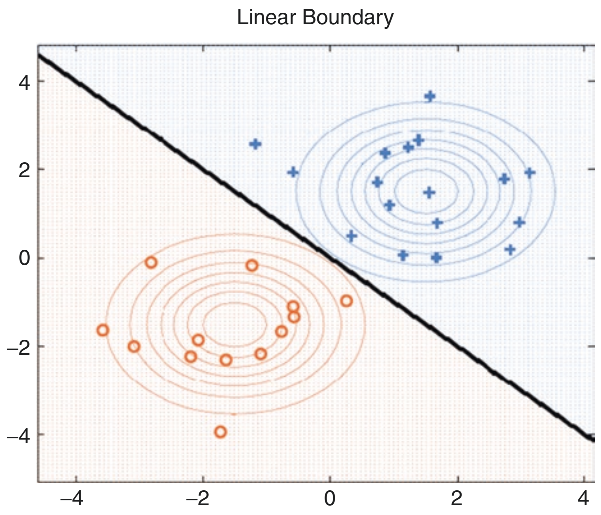


Fig. 6.12 Class-conditional distributions for a binary classification problem. The ellipses demarcate regions of probability that the members of each class (**+** and **o**) will have particular values. The centers of the ellipses are class means, and the shapes of the ellipse are determined by the distribution of class members in relation to these means. In this case a single pattern of distribution has been estimated for both classes, but these have different means

positive test result—into a clinically actionable *positive predictive value*, $P(y = 1|x)$, or the probability that a patient has the disease given a positive test result.

When the class conditional distributions are Gaussians, with tied covariances, as shown in Fig. 6.12 above, the decision boundary between the classes is a hyperplane (line in two dimensions). When both the means and the covariances individually for both classes are estimated, the decision boundary is a quadratic (parabola). Gaussian discriminant analysis can be used to learn generative models for multiclass problems, with a combination of tied and independent covariances for the different classes, as shown in Fig. 6.13.

In sum, Gaussian discriminant analysis, a parametric generative model, is excellent for data that mostly conforms to a multivariate Gaussian distribution. When this assumption about the training data holds, GDA is the best classification method—it yields the most accurate classifier with the least amount of data. Discriminative models, like logistic regression, are less sensitive to assumptions about the distribution of the data in X, and therefore need a lot more examples to build models of comparable performance.

Factored Generative Models: Naive Bayes

Multivariate distributions of the form P(**x**|y) are difficult to handle, both analytically and computationally. One approach around this difficulty is to assume conditional independence between the features of the vector $x \in R^d$, and model the multivariate

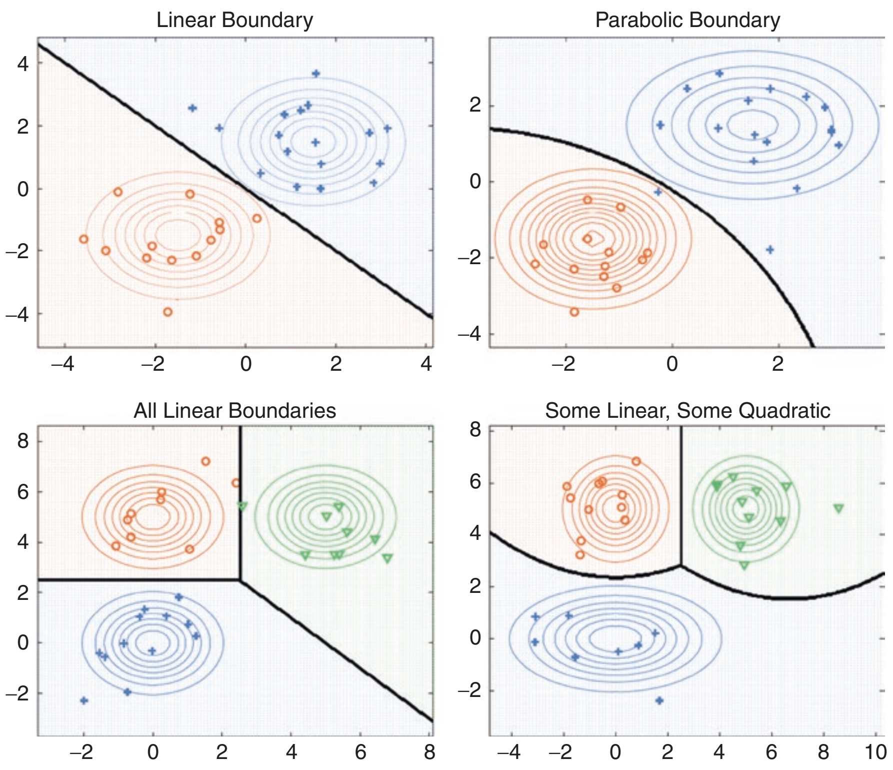


Fig. 6.13 Learning more complex decision boundaries with Gaussian discriminant analysis (GDA). This figure shows the boundaries learned from two class problems with tied (left panels) and individually estimated (right panels) covariance patterns for binary (top panels) and three-class classification (bottom panels). This illustrates the ability of GDA to fit to the unique characteristics of different classes, and address multi-class classification problems

distribution P(**x**|y) as the product of d one dimensional distributions of the form P(x_i | y). This is a strong and often unfounded assumption—for example, when modeling clinical data this would mean ignoring relationships between the values of different liver function tests that in practice may be of considerable diagnostic utility, as well as other important relationships such as drug interactions and relationships to comorbid diagnoses. Nonetheless, despite their “naivete”, models embodying this assumption can lead to surprisingly accurate predictions.

$$P(x|y) = \prod_j P(x_j|y)$$

This factorization works well for both discrete and continuous valued vectors **x**, even when the conditional independence assumptions between the components of **x** conditioned on the class y, do not hold. Naive Bayes models are the industry-standard for the problem of spam detection and for text classification (with over 300,000 articles on Google Scholar as of August 2022) and have been widely used for machine learning in medicine. Recent applications include suicide risk prediction [16], and predicting neonatal sepsis [17].

For illustrative purposes, consider the following problem: there are 1.13 cases of bacterial meningitis per 100,000 population in the US. Assume that 60% of patients with bacterial meningitis report the characteristic symptom of a stiff neck, but that 15% of patients *without* meningitis also report this symptom. Given that a new patient reports a stiff neck, what is the probability that they have meningitis? It is possible to write the facts of this problem as the following probability statements:

1. $P(y = meningitis) = 1.13 \times 10^{-5}$; $P(y = not\text{-}meningitis) = 1 - 1.13 \times 10^{-5}$
2. $P(stiff\ neck = 1 | y = meningitis) = 0.6$
3. $P(stiff\ neck = 1 | y = not\text{-}meningitis) = 0.15$

The first set of equations describe the prior probabilities of the two classes {meningitis, non-meningitis}, and the next two describe the class conditional distributions with respect to a primary symptom of meningitis—stiff neck. Bayes Theorem can be used to calculate the probability that a patient presenting with stiff neck has meningitis:

$$P(y = m | stiff\ \ neck = 1) = \frac{P(stiff\ \ neck = 1 | y = m) P(y = m)}{\sum_d P(stiff\ \ neck = 1 | y = d) P(y = d)}$$
$$\text{where } m = \text{meningitis}$$

The d in the denominator has one of two possibilities: 1 indicating meningitis, and 0 indicating "not meningitis". Since the prior probability of meningitis is very low, and stiff neck is four times more likely to occur in meningitis patients, the posterior probability of the patient having meningitis rises to 4.5×10^{-5}, if a patient presents with a stiff neck alone. However, stiff neck is only one of the symptoms of meningitis. By taking other symptoms into account, such as high fever, nausea, etc., it is possible to improve the estimation of the posterior probability of meningitis in a patient. Suppose k Boolean features are used, representing the presence or absence of specific symptoms—let us call the features $x_1, \ldots, x_k$. To keep things simple, each of these will take on values in {0,1} denoting absence or presence of a symptom. Now every patient is represented by a Boolean vector **x** of length k, with every position in the vector denoting whether a specific symptom is present or absent. To learn the distribution P(**x**|y = meningitis) or P(**x**|y = not-meningitis), it is possible to use

$$P(\mathrm{x} | y = m) = \frac{P(\mathrm{x}, y = m)}{P(y = m)} = \frac{Number\ \ of\ \ m\ patients\ \ represented\ \mathrm{as}\ vector\ \mathrm{x}}{Number\ \ of\ \ m\ patients}$$
$$\text{where } m = \text{meningitis}$$

Since the vector **x** is discrete, it is possible to simply count the number of meningitis patients represented by the vector **x**, and divide it by the total number of patients, to get the proportion of patients of the form **x**, representing a specific absence/presence pattern of symptoms. If k (the number of symptoms) is 20, the possible number of configurations of vector $\mathbf{x}$ is 2^{20}. Estimating the distribution requires estimation of $O(2^{20})$ entries, an intractable task with typically available EHR data.

Now the power of the Naive Bayes assumption can be seen. The multivariate discrete distribution $P(\mathbf{x}|y)$ is factored as the product of k univariate Bernoulli distributions of the form $P(x_j = 1|y = \textit{meningitis})$ and $P(x_j = 1 \mid y = \textit{not meningitis})$. For a k-dimensional vector $\mathbf{x}$, only $2k$ parameters are needed to characterize the class conditional distributions. The reduction of parameters from $O(2^k)$ to $2k$ makes generative modeling of patients a tractable proposition.

$$P(y = m|x) = \frac{P(y = m)\prod_j P(x_j|y = m)}{P(y = m)\prod_j P(x_j|y = m) + P(y = \textit{not } m)\prod_j P(x_j|y = \textit{not } m)}$$

where m = meningitis, *not m* = not meningitis

For the price of estimating $2k + 1$ parameters (the 1 is for P(y = meningitis)), Bayes rule can be used as above to diagnose meningitis. The real challenge in building Naive Bayes classifiers for diagnosis or more generally any classification problem, is choosing good features to characterize a patient. For example, Google's Gmail has a proprietary list of thousands of features that it extracts from each email to stay ahead of the arms race with spammers.

To avoid underflow problems that arise from multiplying thousands of probabilities in the numerator of the posterior probability calculation, the computation is performed in log space.[12] That is, to classify a new patient $\mathbf{x}$ as having meningitis, the following inequality is evaluated, which also eliminates computing the denominator of the expression above, which is identical for both classes.

$$\begin{aligned}&\log\left(P(y = \textit{meningitis})\right) + \sum_j \log\left(P\left(x_j|y = \textit{meningitis}\right)\right) > \\ &\log\left(P(y = \textit{not meningitis})\right) + \sum_j \log\left(P\left(x_j|y = \textit{not meningitis}\right)\right)\end{aligned}$$

Recently, Naive Bayes classifiers for detecting patients at increased risk of suicide were constructed using structured data on 3.7 million patients across five diverse health care systems [16]. The model detected a mean of 38% of cases of suicide attempts with 90% specificity at a mean of 2.1 years in advance of the attempt. This model used univariate Gaussian distributions to model continuous variables obtained from structured health records, and Bernoulli or multinomial distributions for the discrete variables.

[12] This is a common computational optimization that works because $\log(ab) = \log(a) + \log(b)$—so we can add instead of multiplying, obviating the underflow that occurs when repeatedly multiplying by small number; and $\log(a) > \log(b)$ for all $a > b$—so we will assign the class with the highest posterior probability given the data x.

Bias and Variance in Generative Models

It is possible to either underfit or overfit a generative model. Continuing with the meningitis classification example, a model with low bias has a very large number of symptom features. With a limited amount of training data, the estimates of the class conditional probabilities for this large feature set are likely to have high variance. On the other hand, a model with very few features to represent a patient has high bias, and its parameters can be stably estimated, even with a limited amount of data. However, a model with a limited feature set is unlikely to generalize well to new data. Finding the right tradeoff between bias and variance amounts to finding the right feature set (both in content and size) to balance generalization performance and reliability of estimation of the distributional parameters, with respect to the available training data. Feature selection (see Chap. 11) is thus a critical aspect of construction of generative models.

The use of Bayes rule for performing classification of new examples raises a novel problem unique to generative models. Suppose one of the class conditional probabilities corresponding to a specific feature is estimated to be zero given the training data. This situation occurs quite frequently in email classification when a chosen word feature does not appear in the training corpus. Should a new piece of email contain that feature, the Bayes rule computation will yield a zero, since one of the probability terms in the numerator is zero. To guard against this situation, a regularization process called **Laplace smoothing** is performed on probability estimates. Instead of starting word counts at zero in the estimation procedure, counts are started at 1 (or another small constant). So, no class conditional probability is ever estimated to be zero, regardless of the limitations on the training data.

Recap of Parametric Linear Models for Classification

Given a training data set composed of pairs (**x**,y) where **x** is a vector of d dimensions in a continuous/discrete space, and y is a label drawn from the set {0,1}, there are two distinct approaches to building functions that predict y given a new **x**

- **Discriminative models** learn the posterior probability $P(y = 1|\mathrm{x}) = g(\theta^T \mathrm{x})$ as a parametric function and optimize the value of the parameter vector θ to make the predicted distribution of y as close as possible to the true distribution. The learned parameter vector describes the linear classification boundary $\theta^T\mathrm{x} = 0$ between the two classes (0 and 1). Logistic regression belongs to this family of models.
- **Generative models** learn the full joint distribution $P(\mathrm{x}y)$ in terms of its components $P(y)$ and $P(\mathrm{x}|y)$. Generative models come in two forms: full models, and factored models which assume that the components of **x** are independent of one another given the class. Factored models are easier to estimate and work with and are widely used in a range of text classification and clinical decision-making tasks. The decision boundaries they learn can be characterized by a hyperplane in the domain of the input vectors **x**.

Non-linear Models

Linear classification models assume a monotonic and proportional relationship between input variables and the probability of an output label. Models of this sort cannot learn that both high and low blood pressure can be useful predictive features for the onset of renal failure, nor can they learn that the probability of this outcome increases exponentially once a particular blood pressure is reached. However, they can be configured to do so, by transforming the incoming data to enrich the space of features.

Consider the following classification problem in two dimensions (Fig. 6.14 (left)). Can the two classes be separated by a linear boundary?

Clearly, there is no linear separating hyperplane in the original feature space (x_1, x_2). However, it is possible to use the basis function expansion trick introduced in the section on "Augmenting Feature Representations" to map each (x_1, x_2) pair into a new feature space z_1, z_2 defined as

$$\left(z_1, z_2\right) = \left(x_1 * x_1, x_2 * x_2\right)$$

Now a linear hyperplane defined by $z_1 + z_2 \leq \mathrm{R}^2$ where R is the radius of the black circle in Fig. 6.14,[13] achieves perfect separation as shown below.

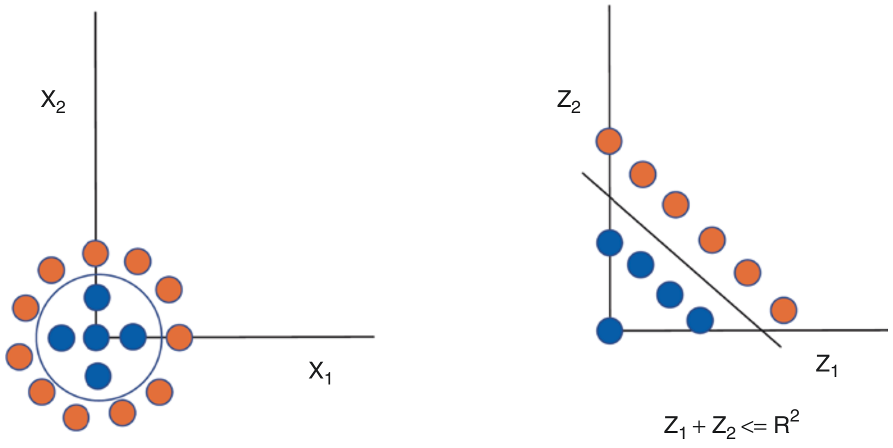


Fig. 6.14 (Left) A non-linear classification boundary in the original (x_1, x_2) feature space. (right): A linear boundary in the (z_1, z_2) feature space where $z_1 = x_1 \times x_1$ and $z_2 = x_2 \times x_2$. Note that the linear boundary with expanded bases (right) corresponds to the non-linear boundary in the original feature space (left)

[13] Summing the squares of x and y returns the square of the distance from the origin, which will be less than R^2 for the examples in the innermost class.

There are two approaches to constructing non-linear classifiers.

1. Stay within the framework of linear classifiers with their optimality guarantees, but manually construct appropriate non-linear feature spaces (basis functions) in which the classification boundaries are linear. The example shown above illustrates this idea. However, the downside of this approach is that it places the entire burden of designing basis functions for data representation on the human modeler.

2. Bypass the explicit construction of basis functions and build non-linear classifiers directly. What is lost in this approach are the optimality/convergence guarantees of linear models. However, there are a number of well-established approaches to apply, which can be broadly categorized as either **kernel methods** or stacked models.

   – Kernel methods represent examples not by their features, but in relation to other examples in the training set. **Kernel regression** and **support vector machines** belong to this family of methods.
   – Stacked models are constructed by chaining or layering simpler learning models. Layered logistic regression models, also known as **deep feedforward neural networks**, are an example of this class of techniques.

Kernel Methods

Consider a binary classification problem with points (x_1,x_2) drawn from a two-dimensional plane, with labels from the set {0,1}, where points in class 0 are colored orange, and points in class 1 are colored blue (see Fig. 6.15). A set of L landmarks from the training data are selected; in Fig. 6.15, the landmark points are labeled l_1, l_2, l_3, i.e., L = 3 for this example. One can think of these landmarks as paradigmatic positive and negative examples for a decision problem. In clinical datasets, such landmarks may correspond to textbook expositions of a disease or condition.

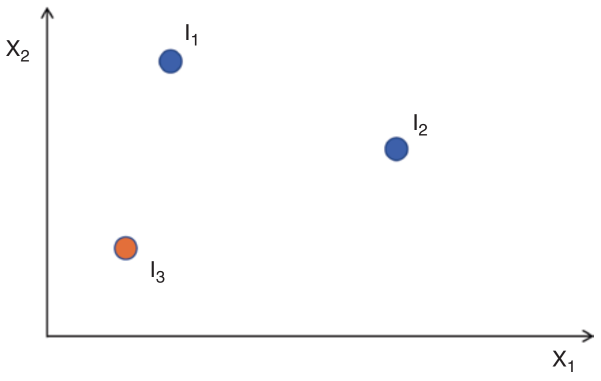


Fig. 6.15 A binary classification problem on points in a plane (training points not shown), with three selected landmark examples (l_1, l_2 and l_3) chosen from the two classes (blue and orange). Unseen data can then be classified on the basis of their relationships to these selected landmarks

Each point in the training data has the form $((x_1,x_2),y)$ and every method discussed so far predicts y from (x_1,x_2) using a function parameterized by a vector θ. For example, logistic regression without basis function expansion would learn θ such that the posterior probability of $y = 1$ given x is well approximated by the sigmoid of the dot product of θ and x.

$$P\left(y=1|;\left(x_1,x_2\right)|;\theta\right)=\sigma\left(\theta_0+\theta_1x_1+\theta_2x_2\right)$$

Instead of describing a training example x by its intrinsic properties—i.e., its location in the $x_1 - x_2$ plane, let us represent it by its "similarity" to the three landmark examples shown in Fig. 6.16. That is, a similarity function *sim* on pairs of points in the $x_1 - x_2$ plane is first defined as follows. This similarity function or ***kernel***, is called a radial basis function.

$$sim\left(x,l\right)=\exp\left(-\frac{x-l^2}{2\sigma^2}\right)$$

sim(x,l) is characterized by a fixed bandwidth parameter σ(unrelated to the sigmoid product above), which is a real number that takes on values in the range [0,1]. The value 0 is achieved when x is far away (in terms of Euclidean distance) from l, and the value 1 is obtained when x is identical to l. In short, *sim(x,l)* characterizes how similar x is to landmark l, for points x and l in an n-dimensional space. Note that *sim* is a symmetric function; $sim(x, l) = sim(l, x)$. This is a required condition for all

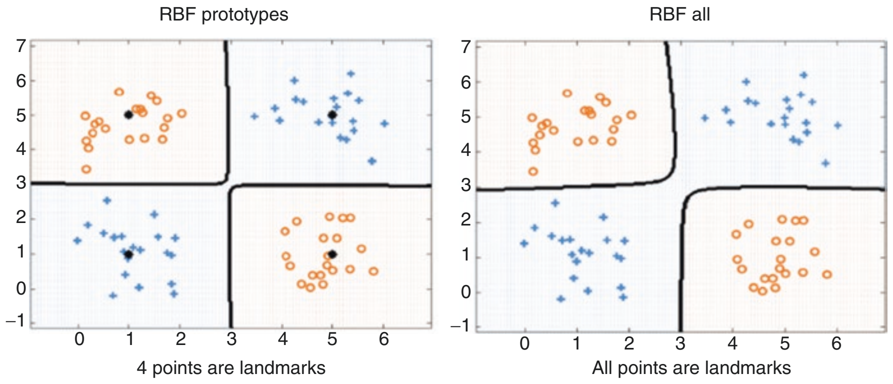


Fig. 6.16 Kernel logistic regression for a binary classification problem. On the left four landmarks are chosen, indicated by the black circles. On the right, all the data points serve as landmarks. The similarity between each data point and the landmarks concerned has been measured using the Radial Basis Function (RBF), described in the section on "Similarity Functions for Kernel Methods". Note that in this case the boundaries are similar, and that both lead to perfect separation of the classes

kernels. The new representation for point x is a three-dimensional vector, characterizing how similar x is to the three chosen landmarks

$$\left(sim\left(x,l_1\right),sim\left(x,l_2\right),sim\left(x,l_3\right)\right)$$

This representation of x is a sequence of pairwise comparisons to three landmark examples: instead of the classical representation in terms of the features of x alone. In this new representation, the posterior probability of y = 1 will be approximated for the example as

$$P\left(y=1|;x|;\theta\right)=\sigma\left(\theta_0+\theta_1 sim\left(x,l_1\right)+\theta_2 sim\left(x,l_2\right)+\theta_3 sim\left(x,l_3\right)\right)$$

The decision rule for classification is a linear function in θ which can be learned by classical logistic regression.

$$\left(\theta_0+\theta_1 sim\left(x,l_1\right)+\theta_2 sim\left(x,l_2\right)+\theta_3 sim\left(x,l_3\right)\right)\geq 0$$

Note that the features associated with the decision function no longer pertain to x alone, but to its similarity to a set of landmark examples. When the radial basis function is used as a similarity function, the linear hyperplane in the similarity feature space forms complex non-linear boundaries in the original x_1–x_2 coordinate space. As an example, consider the task of separating the points laid out in an XOR configuration (XOR is a Boolean function with two inputs that will be true if and only if they are different) as shown in Fig. 6.16. Given points on the plane drawn from two classes, the task is to learn a decision surface that separates the two classes correctly. Clearly, the two classes cannot be separated by a linear hyperplane in the original feature space. Suppose four landmarks l_1, l_2, l_3, l_4, are presciently selected, which are centroids of the four clusters of training points, indicated by a larger filled black circle in the figure. Then, every point $x = (x_1,x_2)$ is mapped into

$$x\rightarrow\left(sim\left(x,l_1\right),sim\left(x,l_2\right),sim\left(x,1_3\right),sim\left(x,l_4\right)\right)$$

Given the 80 points in x labeled 0 or 1 the predictors are transformed into an 80 × 4 matrix K, and the label vector of length 80 containing 0s and 1s denoting members of class 0 or class 1. It is possible to use regularized logistic regression on (K,y), and optimize the penalized cross-entropy loss function to learn the parameter vector θ characterizing the linear hyperplane in the feature space of K. This approach is called **kernelized logistic regression**. To make a prediction on a new example x, x is transformed into a four-dimensional vector kx with the mapping above, and $\sigma(\theta^T kx)$ is computed to obtain its classification. It is possible to project the linear hyperplane into the original feature space as shown in Fig. 6.16 (black lines), and observe that a near-perfect approximation of the XOR function has been learned. This is mostly due to a very judicious choice of landmarks. However, when all training data points are chosen as landmarks, so that the transformed predictor matrix K is of dimension 80 × 80, an excellent approximation to the true function is still obtained.

There are three approaches to landmark selection for kernelized logistic regression:

1. use all examples in the training set as landmarks and choose a large regularization constant in the penalty function, especially if the number of examples is large. Regularization is critical to reduce the risk of overfitting.
2. cluster the examples in the training set and select cluster centers as landmarks. When the number of examples is very large (in the millions), this approach works better than choosing all examples as landmarks.
3. have domain experts suggest landmarks.

Kernelized logistic regression has been applied to predict the effects of drugs by representing them in terms of their similarity to other drugs. For example, McCoy and Perlis describe the application of logistic regression to drug representations that include the similarities between the side-effect profiles of a drug and those of a curated panel of six drugs that affect the central nervous system, in order to predict which drugs will cross the blood-brain barrier [18]. This work exemplifies the expert-driven approach to landmark selection.

Similarity Functions for Kernel Methods

The success of kernelization is closely tied to the choice of the similarity or kernel function. A kernel function k measures how similar two d-dimensional vectors are.

$$k : R^d \times R^d \to R$$

It takes two vectors as arguments and returns a real number measuring the similarity between the two input vectors. The **radial basis function** is a popular general-purpose kernel for vectors in R^d. It has been rediscovered in many applied areas of science, and is known by a variety of names, including the Gaussian kernel.

$$k(x,l) = \exp\left(-\frac{x - l^2}{2\sigma^2}\right)$$

Another popular kernel is the polynomial kernel of order *n*, defined as

$$k(x,l) = \left(x^T l + 1\right)^n$$

Note that both functions are symmetric. Kernel functions are often designed with specific applications in mind; this activity is called kernel engineering. Consider the problem of predicting DNA sequences in the human genome that encode proteins. A supervised machine learning approach to this problem casts the problem in the framework of binary classification. It entails gathering a training set of protein-encoding DNA sequences and DNA sequences known not to encode proteins (e.g., sequences drawn from regulatory regions). The success of the machine learning approach is deeply tied to how the input DNA sequences are represented. A poor representation leads to predictive models with poor generalization performance.

One idea for representing the training data sequences is to have expert biologists manually engineer features (e.g., counts of specific short subsequences which may/may not be indicative of protein regions). Another idea is to move away from a representation tied to a single DNA sequence to a comparative representational approach. That is, rather than trying to describe a DNA sequence on its own, a kernel function *k* is defined on DNA sequences which evaluates how similar two sequences are. Such kernel functions are much easier to design than features on an individual DNA sequence. Next, L landmark DNA sequences from both classes (protein-coding and non-coding) are selected and each DNA sequence is represented as a vector of length L denoting its similarity to these L landmarks. It is then possible to apply penalized logistic regression, Gaussian Discriminant Analysis, or Factored Naive Bayes models to learn the prediction function from the kernel representations. The reader is directed to the following textbook for further discussion of this approach [19].

The practical significance of working in kernel space, rather than in an expanded basis function space is revealed through the following image processing example. Suppose images of size 16 × 16 are available for a binary classification task. If all fifth order polynomial terms in the 16 × 16 pixel space are considered as features, the size of the feature space will expand to $\sum_{k=1}^{5}\binom{256}{k} \approx 10^{10}$. Instead, working with a polynomial *kernel* of degree 5, it is possible to compute $(x^T l + 1)^5$ between image x, and landmark l in $O(16 \times 16)$ time—take the dot product of image x and image l, and add 1 to the value, and raise it to the fifth power. The basis function space of all fifth order polynomials is never explicitly materialized, but the effect of working in that space is obtained, with simply O(256) amount of work! This is the magic of kernels.

Recap: How to Use Kernels for Classification

Given a labeled training set D of (**x**,y) pairs,

- Choose L landmarks from D
- Choose a kernel function *k* that captures similarity between pairs of examples
- Represent each **x** in D by a vector of length L + 1 of the form $(1, k(x, l_1), \ldots, k(x, l_L))$. The prepended 1 is used for the intercept term θ_0 in the learned model. The new training set K has (**kx**,y) pairs, where **kx** is the kernelized representation of **x**.

- Use a discriminative (penalized logistic regression) or generative (Gaussian discriminant Analysis, Naive Bayes) model to estimate a parameter vector from the kernel transformed data K and label vector y.
- To predict on a new example **x**, map it to its kernel form **kx**, and use the learned parameter vector with **kx** to compute the classification.

Sparse Kernel Machines and Maximum Margin Classifiers

Kernelization can yield models with poor generalization performance, particularly if landmarks are chosen poorly. Too many landmarks could potentially result in an overfitted model, and too few landmarks result in underfitted models. Conceptually, the most important landmarks are arguably those that are close to the boundary between classes, because it is these landmarks that will help to distinguish between examples that are hardest to classify. A geometric view of the classification problem gives us insight into how to simultaneously select good landmarks and build a high-performance classifier.

Consider the points on a plane drawn from two classes as shown in Fig. 6.17. The points are linearly separable, and an infinite collection of boundary lines drawn in the space between the two sets of points is a perfect classifier. Of all these lines, only one, shown as a dotted black line in Fig. 6.17, maximizes the distance between the points in the two classes. The separating hyperplane is equidistant from the closest points to it in both classes. By formulating the problem of finding the decision boundary that *maximally* separates two (separable) classes as an optimization problem, it is possible to find a *unique* solution to the problem of finding the "best" classifier. In Fig. 6.17, the position and orientation of the maximum margin separating line is determined by just two of the nine training data points—i.e., the points at the

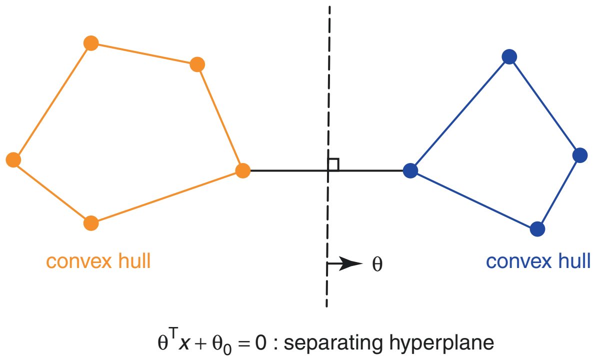


Fig. 6.17 The optimal decision boundary separating points from two classes on the plane. This boundary is the maximum margin separating line and is the perpendicular bisector of the line joining the closest points on the **convex hulls** of the two linearly separable classes. Note that this decision boundary can be identified using a single landmark from each class only

ends of the line perpendicular to the decision boundary. These two points are called *support vectors* and form the sparse set of landmarks needed to describe the optimal, margin-maximizing decision boundary. The problem of finding the (sparse) set of landmarks and the optimal placement of the decision hyperplane is thus solved jointly.

To understand how to set up the optimization problem of maximizing the geometric margin between two sets of separable points, it is helpful to review some geometry as shown in Fig. 6.18. The margin r of a point x from a hyperplane defined by $\theta^T x + \theta_0 = 0$ is the perpendicular distance of x from the plane. θ is the slope of the hyperplane, and θ_0 is the intercept.

The *sign* of the distance of a point from a decision boundary is determined by whether it is on the positive or the negative side of the hyperplane (the distance for x would be positive, but one might imagine the reflection of x falling on the negative side of the boundary, to the left of it). The label set {0,1} will be mapped to the set {−1,1} to write a single formula for the margins of members of both classes. With y as an element of this set (i.e., a label in {−1,1}), the margin of a point (x,y) in a dataset D is

$$r = y\frac{\theta^T x + \theta_0}{\|\theta\|}$$

Note that this new definition of margin r is greater than 0 for points on both sides of the boundary since the (negative) distance is multiplied with label $y = -1$ for points in the negative class. A dataset D is correctly classified, if and only if the margins of

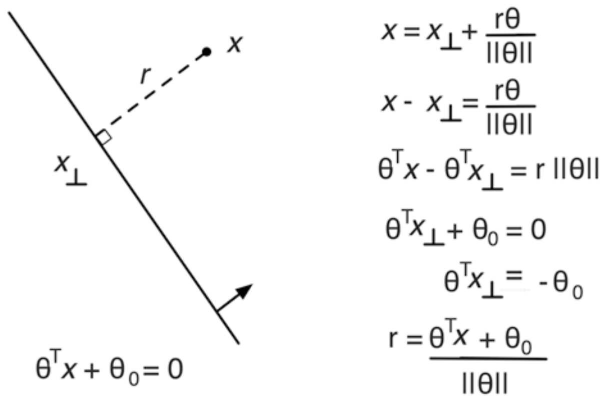


Fig. 6.18 r is the perpendicular distance of the point x from the hyperplane $\theta^T x + \theta_0 = 0$. The unit vector perpendicular to the plane is $\frac{\theta}{\|\theta\|}$. A perpendicular line (of length r, and parallel to this unit vector) dropped from the point x meets the plane at $x_\perp$. So, x is the vector sum of $x_\perp$ and r $\frac{\theta}{\|\theta\|}$. Rearranging terms, and taking the dot product on both sides by θ, the given expression for the scalar distance r is derived

all the points in D are greater than 0. The margin of the entire dataset is the smallest margin among all its elements.

$$margin(D) = \min_{(x,y)\, in\, D}\, y \frac{\theta^T x + \theta_0}{\|\theta\|}$$

To separate the two classes maximally, the slope θ and intercept θ_0 of the separating hyperplane to maximize *margin(D)* must be selected. That is, there is a need to find

$$(\theta, \theta_0) = argmax_{(\theta,\theta_0)} \min_{(x,y)\, in\, D}\, y \frac{\theta^T x + \theta_0}{\|\theta\|}$$

Unfortunately, this specification of the optimization problem yields infinitely many solutions for θ and θ_0—because for every k, if θ and θ_0 are solutions, so are $k\theta$ and $k\theta_0$. That is, the optimal solution is agnostic to the length of the vector defining the hyperplane. To obtain a unique solution, a scaling factor is defined such that

$$\min_{(x,y)\, in\, D}\, y\left(\theta^T x + \theta_0\right) = 1$$

for the point (x,y) in the dataset D that is closest to the decision boundary. So, the optimization problem is reduced to finding

$$(\theta, \theta_0) = argmax_{(\theta,\theta_0)} \frac{1}{\|\theta\|}$$

subject to the constraints that $y(\theta^T x + \theta_0) \geq 1$ for all points (x,y) in D. That is, all points in D must be at a distance of one or greater from the decision boundary. Maximizing $\frac{1}{\|\theta\|}$ is equivalent to minimizing $||\theta||$, so the maximization problem is converted into a constrained quadratic minimization problem and can be solved using a classical numerical solver.

$$\min_{\theta,\theta_0} \frac{1}{2}\|\theta\|^2 \text{ subject to } y\left(\theta^T x + \theta_0\right) \geq 1 \text{ for all} (\text{x,y}) \text{ in D}$$

The solution obtained from the solver has the form

$$\theta = \sum_{(x,y)\, in\, D} \alpha\, yx$$

which associates a weight α with each point (x,y) in D. The slope and intercept of the maximum margin separator depends only on those (x,y) for which $\alpha > 0$, as illustrated in Fig. 6.19. These points are the support vectors for the classifier and form a sparse set of landmarks for the dataset. The dark line in Fig. 6.19 is the

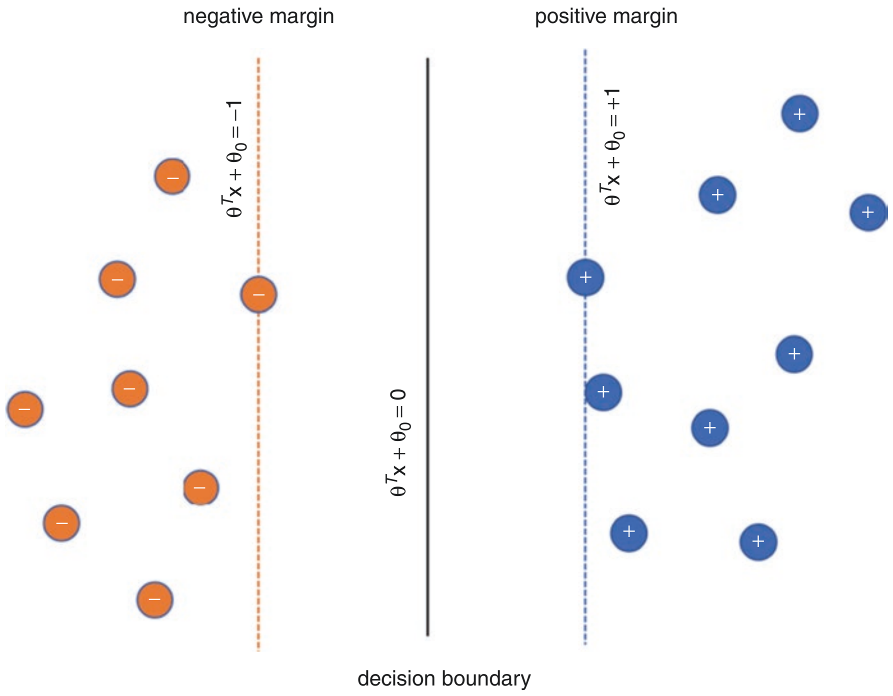

negative margin
positive margin
$\theta^T x + \theta_0 = -1$
$\theta^T x + \theta_0 = 0$
$\theta^T x + \theta_0 = +1$
decision boundary

Fig. 6.19 Two support vectors for a two-class problem in two dimensions. The position and orientation of the decision boundary is determined by the two support vectors, which form a sparse set of landmarks for the dataset shown. The support vectors are computed by solving the quadratic optimization problem of finding the maximum margin separator between the two classes. The dotted blue line, called the positive margin, is parallel to the decision boundary. All positive (blue) examples lie on or to the right of the positive margin. The dotted orange line is the negative margin. A. negative (orange) examples lie on or to the left of the negative margin. The positive and negative examples are separated by a distance equal to the margin width. The larger the margin width, the more robust the classifier

decision boundary, and the dotted lines at +1 and −1 denote the positive and negative margin lines. The resulting classifier is called a **support vector machine** (SVM).

To classify a new point x' with a SVM the side of the decision boundary the point lies on is computed,

$$sign\left(\theta^T x' + \theta_0\right) = sign\left(\sum_{(x,y)\, in\, D} \alpha\, y x^T x' + \theta_0\right)$$

Only the support vectors with non-zero α are involved in this computation, even though the summation as written is over all (x,y) in D. Effectively, the position of the new point relative to *all* of the support vectors is used to make classification decisions, taking into account both which side of each support vector the point lies on, and how far it is from this support vector. To build a non-linear SVM, the linear dot product $x^T x'$

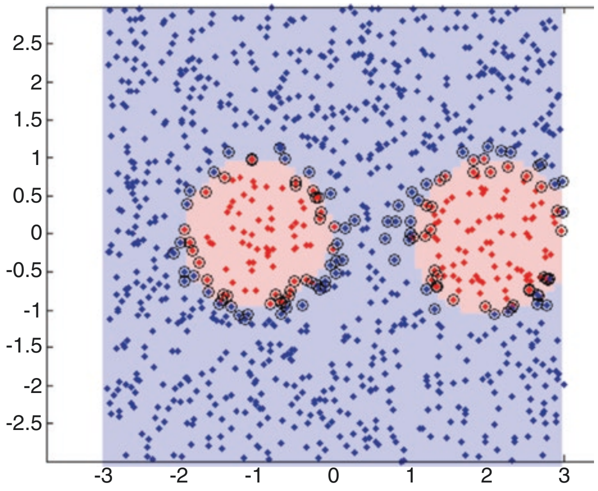


Fig. 6.20 Non-linear decision boundaries (demarcating the red and the blue shaded regions) and support vectors (circles enclosed in a purple ring) for a 2D data set learned with a radial basis function

is replaced by a kernel function $k(x,x')$, such as the radial basis function. It is possible to learn non-linear decision boundaries in two-dimensional data as shown in Fig. 6.20 and identify a sparse set of support vectors using radial basis function as kernels, rather than the linear dot product kernel described in the equation above.

In all the discussion so far, it has been assumed that the two classes are separable, either linearly with a dot product kernel, or non-linearly with a polynomial or radial basis function kernel. To make SVMs practical for real world data sets which are not linearly separable to begin with, the formulation of maximum margin classifiers is extended to allow misclassified points. This tolerance for misclassification also relates to the recurring theme of overfitting. A solution that permits misclassification of points in the training set may generalize better to other data than one that fits the training set perfectly (Fig. 6.21).

The optimization objective of SVMs will be adapted to relax the margin constraint on some of the points. Margin width of the final classifier can be traded off with the number of points in the data set that are allowed to violate margin constraints, i.e., fall on the wrong side of their class boundary.

Margin violation $\xi_{(x,y)}$ of a point (x,y) is defined as the distance of x from its class margin (Fig. 6.22). Then, the margin maximization optimization problem is formulated as

$$\min_{\theta,\theta_0} \frac{1}{2}\|\theta\|^2 + C\sum_{(x,y) \text{ in } D} \xi_{(x,y)} \text{ subject to}$$
$$y\left(\theta^T x + \theta_0\right) \geq 1 - \xi_{(x,y)} \text{ and } \xi_{(x,y)} \geq 0 \text{ forall } (x,y) \text{ in} D$$

A point that is correctly classified will have a margin violation of exactly zero, incorrectly classified points will have margin violations greater than zero. The regularization constant C is a measure of the willingness to allow misclassifications.

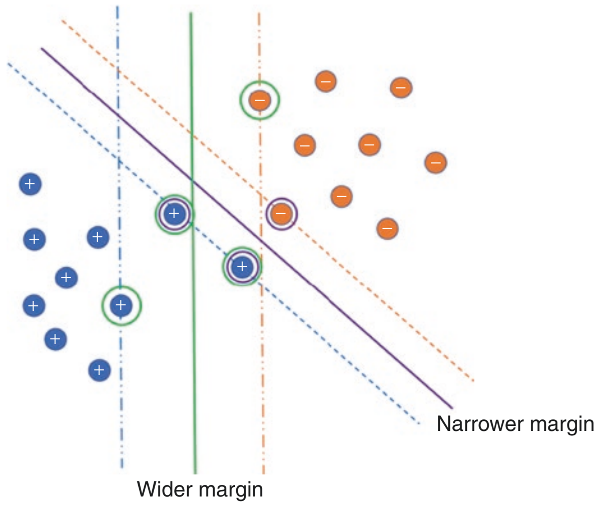


Fig. 6.21 Given the two-class dataset on the plane, the maximum margin separator derived by an SVM is shown by the solid purple line and the small-dashed margins. This SVM has a narrow margin and is defined by three support vectors, that are circles, enclosed in a purple ring. If the two circled positive class points in the center are ignored, it is possible to construct an SVM classifier with a much wider margin indicated by the broken dotted blue and orange lines. The support vectors for the wider margin classifier are circled in green. Wider margin classifiers have better generalization performance

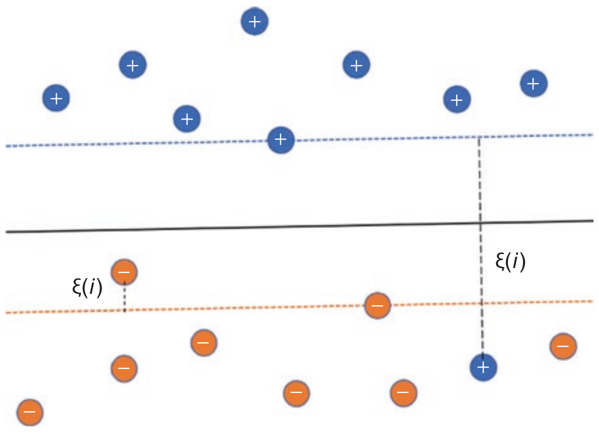


Fig. 6.22 The orange point outside the orange margin, and the misclassified blue point outside the blue margin, have both violated their margins. The extent of violation, defined as the distance from the point to its class margin, is traded off against margin width in the penalized objective function for an SVM for non-separable data

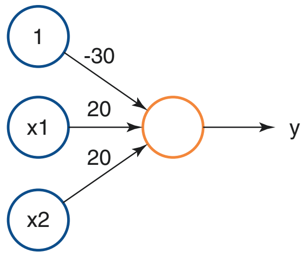
This is analogous to the approaches to regularization introduced in the discussion of linear models, in the sense that permitting misclassifications prevents a model from overfitting to the training set, improving generalization to new datasets. If C is high (imposing a high penalty for misclassification), then the margin violation term dominates, and the optimizer is forced to reduce the margin width concomitantly; while if C is low, the optimizer will construct a wide margin classifier since margin violations are not penalized so heavily. The penalized version of the SVM optimization yields a soft-margin SVM. For any value of C, which expresses the tradeoff between margin width and misclassification rate, the optimizer produces a unique solution to θ and θ_0 and identifies the sparse set of landmarks (i.e., support vectors).

Neural Networks: Stacked Logistic Models

In this section, neural networks will be introduced. As noted throughout this volume, neural networks—and, in particular, **deep neural networks**—have led to rapid advancements in automated image recognition, speech recognition, and natural language processing, amongst other areas. A remarkable feature of contemporary neural network models is that the underlying models are composed of individual units that are relatively simple in their design. It is the interaction between large numbers of such units that gives deep neural networks their representational power.

The fundamental building block of deep neural networks is the logistic model, shown in Fig. 6.23. As was seen in the section on "Discriminative models: Logistic Regression", the logistic model characterized by the parameter vector θ captures the posterior probability $P(y = 1|x;\theta)$ from a dataset composed of vectors $x \in R^d$ with associated labels $y \in \{0, 1\}$. It predicts class membership y from the input x by computing $g(\theta^T x)$ where g is the sigmoid function. The sigmoid function squashes the dot product $\theta^T x$ which can be an arbitrary real number, into the range [0,1]. The dot product here provides a concise way to express a sequence of operations in which the values of each blue input node (1 for the bias term, x_1 and x_2) are multiplied by a corresponding weight $1(-30) + x_1 (20) + x_2 (20)$ and added together to produce the input to the orange node, which applies the sigmoid function as its **activation function**. A logistic model is a linear classifier—it can only capture linear boundaries separating classes $y = 0$ and $y = 1$.

The model in Fig. 6.23 maps $x \in \{0, 1\}^2$ to $y \in \{0, 1\}$. The network computes the Boolean AND of the two components of x. Note that the sigmoid function $g(a) \approx 0$ for $a \ll 0$ and $g(a) \approx 1$ for $a \gg 0$. Thus, when both components of x are 1, the linear



$$y = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} = g(-30 + 20x_1 + 20x_2)$$

Fig. 6.23 A logistic model, drawn as a two-layer neural network. The first layer is the Boolean input vector **x** and the second layer is the output y composed of a single logistic unit. This network computes the Boolean AND of the components of x

dot product $\theta^T x = 10$; and $y = g(10) \approx 1$. When any one, or both of the components of x are zero, the corresponding dot product $\theta^T x \leq -10$, so $y = g(-10) \approx 0$.

In 1943, McCollough and Pitts [20] demonstrated that a stacked assembly of logistic units can represent **any** Boolean function—and by extension any function that can be calculated on a classical computer. An example of a stacked assembly composed of three layers is shown in Fig. 6.24. The network represents the nonlinearly separable XNOR function (the logical complement of the XOR function—true if and only if both inputs are identical), i.e., the output $y = x_1$ *XNOR* x_2.

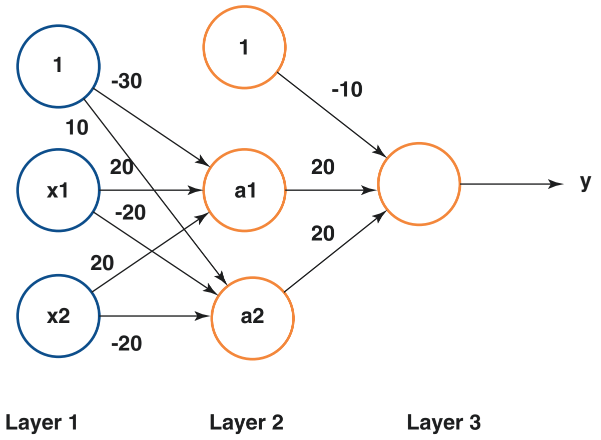
Given an input Boolean vector x, the intermediate outputs a_1 and a_2 of the second (hidden) layer are calculated, and the output y of the final layer is then computed in terms of a_1 and a_2:

$$a_1 = g\left(-30 + 20\,x_1 + 20\,x_2\right)$$

$$a_2 = g\left(10 - 20\,x_1 - 20\,x_2\right)$$

$$y \;= g\left(-10 + 20\,a_1 + 20\,a_2\right)$$

Computation of the outputs of the network proceeds sequentially, layer by layer, with outputs of layer $i + 1$ computed from the outputs of layer i. This network is



X_1	X_2	a_1	a_2	y
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Fig. 6.24 A three layer, fully connected, stacked assembly of logistic units with an input layer, a hidden layer (composed of a bias unit set to 1, and two logistic units with outputs a_1 and a_2) and an output layer with a single logistic unit with output y. The network computes the nonlinear function x_1 XNOR x_2, which cannot be represented by a single logistic model. The parameters defining the model are the weights on the edges connecting the units to one another, and it is these weights that define the behavior of the model. The attentive reader will recognize the AND subnetwork from Fig. 6.23 leading to output a_1

fully connected, i.e., every logistic unit in layer $i + 1$ is connected to all the units in the layer i below. In the example network shown in Fig. 6.24, layer 2 outputs are computed from layer 1 (the inputs), and the layer 3 output is computed from the values of units in layer 2. Note that the units a_1 and a_2 in layer 2 are connected to all units (including the bias unit) in the input layer. Fully connected stacked assemblies of logistic units are therefore called **feedforward multilayer networks**. The final output y is a highly non-linear function of the input vector $\boldsymbol{x}$, viz.,

$$y = g\left(-10 + 20\,g\left(-30 + 20\,x_1 + 20\,x_2\right) + 20\,g\left(10 - 20\,x_1 - 20\,x_2\right)\right)$$

The parameters of the feedforward network are defined in Fig. 6.24 using two parameter matrices, one defining the weighted connections between layer 2 and layer 1 (the matrix $\Theta^{(1)}$), and the other between layer 3 and layer 2 (the matrix $\Theta^{(2)}$) shown below (Fig. 6.24).

$$\Theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \\ 10 & -20 & -20 \end{bmatrix}; \quad \Theta^{(2)} = \begin{bmatrix} -10 & 20 & 20 \end{bmatrix}$$

Note that $\Theta^{(1)}$ has dimension 2×3, reflecting the fact that there are two logistic units in layer 2 connected to the two input units in layer 1 and the additional bias unit. $\Theta^{(2)}$ has dimension 1×3, since there is a single output unit in layer 3 connected to the two logistic units and the bias unit in layer 2.

Parameterizing Feedforward Networks and the Forward Propagation Algorithm

Given a new input vector **x**, it is possible to define the vector $a^{(l)}$ of activations of the units in each layer l in the network with the following system of matrix operations.

$$a^{(1)} = \boldsymbol{x}$$

$$z^{(l+1)} = \Theta^{(l)} * \left[1; a^{(l)}\right], \quad l = 1 \ldots L - 1$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right), \quad l = 1 \ldots L - 1$$

The symbol z in these equations corresponds to the sum of the inputs to a layer (or individual unit within a layer) *before* the activation function g has been applied. In the running example, the value z—known as the **logit**—for unit a_1 would be $-30 + 20\,x_1 + 20\,x_2$. The equations provide a symbolic representation of a sequence of steps in which the input, x, provides initial activation values for the input layer of the network. Then, the logits, z, for a subsequent layer are calculated by multiplying these inputs by the weighted connections, Θ, to each of its units, and adding the results. The matrix product in the equation provides a shorthand notation for performing this operation across all units in a layer and describes the way in which this operation is generally implemented to take advantage of efficient numerical computations provided by Graphical Processing Units (GPUs) and other specialized hardware. Finally, the activation function, g, is applied to the logits, to estimate the activations for this layer, which may in turn provide the input for a layer to follow.

The system of equations for forward propagation shown above can be used to calculate the final output of a fully connected, L layered feedforward network. Such a network is characterized by L − 1 parameter matrices $\Theta^{(l)}$, $l = 1 \ldots L - 1$. Each layer l has $S^{(l)}$ units and a single bias unit. Matrix $\Theta^{(l)}$ connects layer l to layer $l + 1$ and has dimensionality $S^{(l+1)} \times (S^{(l)} + 1)$. For example, in the network depicted in Fig. 6.24, $S^{(1)} = 2$, $S^{(2)} = 2$, $S^{(3)} = 1$, $L = 3$. This network has two parameter matrices of sizes $2 \times (2 + 1)$ and $1 \times (2 + 1)$ and a total of 9 parameters.

By increasing the number L of layers, as well as by increasing the number of logistic units in each layer, it is possible to represent nonlinear functions of ever-increasing complexity. A deep feedforward network encodes a set of prior beliefs about the structure of the function that maps vectors $\boldsymbol{x}$ to the class y. The intermediate layers represent underlying factors of variation, which in turn can be expressed in terms of simpler factors, all the way down to the input components in $\boldsymbol{x}$. Fully connected, feedforward networks are universal function approximators, capable of representing any mapping from inputs to outputs to within any specified tolerance ε of the true function [21].

Any Boolean function can be represented by a three-layer network, such as the one shown in Fig. 6.24. Any continuous function on the reals can be approximated by a three-layer network, but it may require a very large number of units in each layer [22, 23]. Finally, any function (including discontinuous functions) can be approximated by a four-layer network with enough units in each layer. Surprisingly, these representation theorems hold not just for logistic units (in which the non-linearity $g(x)$ is the sigmoid function), but for **rectified linear units** (called ReLUs), defined as $g(x) = \ max\ (0, x)$ as well. ReLUs are the most widely used non-linearity in feedforward neural networks, because the maximum is cheaper to compute than the sigmoid, and it has been experimentally found to accelerate the convergence of stochastic gradient descent for parameter learning [24].

While the representation theorems assure us that a network with four layers is sufficient to capture any mapping, they place no bounds on the "width" of each layer. In practice, networks of much greater depths are built, trading off the number of units in each layer for depth. While a feedforward neural network of sufficient depth and width can represent any function in principle, in practice, it is not guaranteed to find those parameter settings with the training algorithms.

In the networks discussed up to this point, the parameters of the networks have been pre-configured to approximate particular Boolean functions. However, neural networks must learn how to approximate functions of interest for AIM applications, such as radiological image recognition. As is apparent from the network in the running example, the behavior of a feedforward neural network is dictated by its weights, as these determine how input from one layer proceeds to activate units in the next. Therefore, it is the weights that must be adjusted to change the behavior of a neural network during the process of training it. The section that follows will explain how the backpropagation algorithm—the mainstay of training neural networks—is used to accomplish this end.

Learning the Parameters of a Feedforward Network

A standard way of presenting the backpropagation algorithm uses calculus to calculate a sequence of derivatives, each of which measures how much the result of one operation (such as multiplication by a set of network weights, or the application of an activation function) influences the operation that follows it. As with linear regression, the key idea is to update each model parameter in accordance with its influence. However, with linear regression this influence is straightforward to estimate, as it depends only upon the (constant) slope of the line concerned, and the input features. With nonlinearities such as the sigmoid function, the extent to which a particular change in input influences the output of the function differs depending upon the value of the function beforehand (Fig. 6.25).

With knowledge of the extent of the influence of each neural network weight on the output of the model—and hence the loss function—it is possible to update individual parameters to steer the model toward accurate classification of the data in a training set. Beginning with this standard presentation, some other perspectives on the algorithm will be provided to help to build intuition about backpropagation. While this constitutes more detail than has been provided with some of the other algorithms under discussion, this is warranted here because backpropagation is fundamental to training deep neural networks, which have become—or are becoming—the dominant approach to many problems in AIM. As ultimately there is a

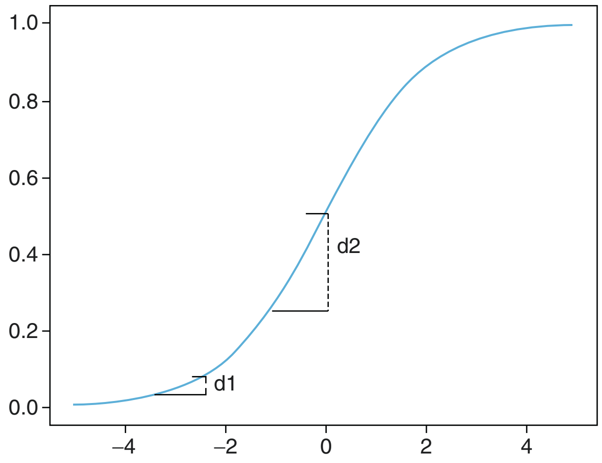


Fig. 6.25 The extent to which changing the input of the sigmoid function (x axis) influences its output (y axis) depends upon the value of the function beforehand. Both d1 and d2 represent the change in y after adding 1 to x, but d2 is much larger

need to know how much each parameter influences the loss function, the presentation will begin there.

$$J(\theta) = -\left(\left[y\log\left(h(x)\right)\right] + \left[(1-y)\log\left(1-h(x)\right)\right]\right)$$

Recall the cross entropy loss function for a single example *(x,y)*, shown above. The prediction function with parameter vector θ is

$$h(x) = g\left(\theta^T x\right)$$

The parameters θ (referred to as weights in neural network parlance) can be learned from a given data set by gradient descent on the cross-entropy loss function

$$\theta \leftarrow \theta - \frac{dJ(\theta)}{d\theta}$$

The gradient of the cross-entropy loss function with respect to the component θ_j is

$$\frac{dJ(\theta)}{d\theta_j} = \left(h(x) - y\right)x_j$$

The first term is the error in prediction on the *(x,y)* pair (how "wrong" the model was), and the second term is the j-*th* component of the input vector x (this determines how changing the parameters in θ will affect a model prediction—for example, if $x_j^{(i)} = 0$, changing the parameter $\theta_j^{(i)}$ will not improve classification of this example). Therefore, the gradient tells us how each parameter in θ influences how well the prediction for a specific example approximates the correct label, y. This gradient can be used for updating the parameters connecting the final two layers of a feedforward network, which function as a simple logistic model. However, with deeper networks there is a need to estimate the influence of weights in proximal layers on model error, to determine how these weights should be updated. The simple feedforward network in Fig. 6.26 will be used to explain how derivatives of the loss function can be computed with respect to all the weights in the network.

The input vector x has two components, and the hidden layers have two logistic units each. The final output is a scalar $a^{(4)}$. Here are the forward equations to calculate the output $h((x_1,x_2))$ of the network, repeating the sequence of equations from the section on "Parameterizing feedforward networks and the forward propagation algorithm" (input → logit (z) → activation ($g(z)$) for each layer of the network.

$$a^{(1)} = \left(x_1, x_2\right)$$

$$z^{(2)} = \Theta^{(1)} * \left[1; a^{(1)}\right]$$

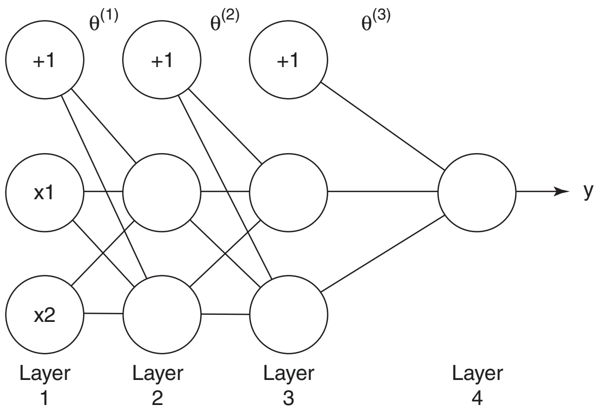


Fig. 6.26 A simple four-layer feedforward neural network to illustrate the backpropagation algorithm and the computation of the derivative of the loss function with respect to all the network parameters

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$z^{(3)} = \Theta^{(2)} * \left[1; a^{(2)}\right]$$

$$a^{(3)} = g\left(z^{(3)}\right)$$

$$z^{(4)} = \Theta^{(3)} * \left[1; a^{(3)}\right]$$

$$a^{(4)} = g\left(z^{(3)}\right)$$

First, the derivative of the loss function $J(\theta)$ with respect to $\theta^{(3)}$ is calculated, using the chain rule of differentiation. This estimates the influence of the weights in the penultimate layer ($\Theta^{(3)}$) on the loss function. As the output of the third layer will be ingested by the fourth layer to generate the output of the network, this estimation must consider the operations of the output unit also.

$$\frac{dJ}{d\Theta^{(3)}} = \frac{dJ}{da^{(4)}} \frac{da^{(4)}}{dz^{(4)}} \frac{dz^{(4)}}{d\Theta^{(3)}}$$

Consequently, the first term is the derivative of the cross-entropy loss with respect to the network output, the result of applying the sigmoid activation function to the logit of the output unit, $z^{(4)}$. The second term is the derivative of this non-linear sigmoid function with respect to the logit, and the final term is the derivative of the logit with respect to the $\theta^{(3)}$ parameter. The steps of the calculation are as follows:

$$J(\theta) = -\left[\left[ylog\left(a^{(4)}\right)\right] + \left[(1-y)\log\left(1-a^{(4)}\right)\right]\right]$$

$$\frac{dJ}{da^{(4)}} = -\frac{\left(y - a^{(4)}\right)}{a^{(4)}\left(1 - a^{(4)}\right)} \text{ —hint}[14]$$

$$\frac{da^{(4)}}{dz^{(4)}} = a^{(4)}\left(1 - a^{(4)}\right)$$

$$\frac{dz^{(4)}}{d\Theta^{(3)}} = \left[1; a^{(3)}\right]$$

$$\frac{dJ}{d\Theta^{(3)}} = \left(a^{(4)} - y\right)\left[1; a^{(3)}\right]$$

which mirrors the derivative of a single logistic unit with input *x* replaced by the activation of the layer right below the output layer ([1;a$^{(3)}$], with 1 indicating the relevant bias term). The numerator of the first derivative calculated, $\frac{dJ}{da^{(4)}}$, incorporates the class label: ($y - a^{(4)}$) indicates both the direction and the magnitude of the desired correction to the output of the model. If the label *y* is one, this will measure how far from one the output was, and the sign will be positive. If the label *y* is zero, this will measure how far from zero the output was, and the sign will be negative. The denominator is the derivative of the sigmoid function,[15] $g(x)(1 - g(x))$.

This is also the second derivative calculated, because it governs the influence of the logit it activates, $z,^{(4)}$ on the loss function. The influence of the weights in the preceding layer, $\theta^{(3)}$, on this logit depend upon the activations they are multiplied by, $a^{(3)}$. When these three derivatives are multiplied together, as dictated by the chain rule, the term $a^{(4)}(1 - a^{(4)})$ cancels out, leaving $\frac{dJ}{d\Theta^{(3)}}$. Essentially, the chain of influence is traversed in reverse, from the error function through the logistic unit, and finally to the weights of the third layer. From here it is possible to proceed recursively.

A second view of the backpropagation algorithm considers it from the perspective of how responsibility for error is allocated across the weights of the network. If the counterpart of model error (the $a^{(4)} - y$ term) for the hidden units in the network is known, it is possible to compute derivatives of the loss function with respect to $\theta^{(2)}$ and $\theta^{(1)}$ as well. Error at the output layer is defined as

$$\delta^{(4)} = \left(a^{(4)} - y\right)$$

[14] Recall (perhaps from distant calculus) that the derivative of d/dy log(y) = 1/y. Proceed algebraically from this step by cross-multiplying the summed fractions to reach this derivative.

[15] The sigmoid function $g(x) = 1/(1 + e^{-x}) = (1 + e^{-x})^{-1}$. Its derivative $dg(x)/dx = (-1)(1 + e^{-x})^{-2} de^{-x}/dx = (-1)(1 + e^{-x})^{-2}(-1)e^{-x} = 1/(1 + e^{-x})(1 - 1/(1 + e^{-x})) = g(x)(1 - g(x))$.

The key idea behind the backpropagation algorithm is that each of the hidden nodes in Layer 3 is responsible for some fraction of the error in the output node, and the extent of this responsibility depends upon the weights connecting them to it. For example, the error at hidden node *j* in Layer 3 is determined by $\theta_j^{(3)}$, the weight connecting unit *j* to the output unit, and the error at the output unit, $\delta^{(4)}$, modulated by the extent to which changing the logit of this node will affect the output of the activation function that follows it.

$$\delta_j^{(3)} = \theta_j^{(3)} \delta^{(4)} \left(\frac{da^{(3)}}{dz^{(3)}} \right)_j$$

In vector form, this can be rewritten as

$$\delta_j^{(3)} = \left(\theta_j^{(3)} \right)^T \delta^{(4)} \odot \frac{da^{(3)}}{dz^{(3)}}$$

where the $\odot$ operator is the Hadamard product of the two vectors—a pointwise multiplication of the components of two equally sized vectors. One can see the correspondence between the the *j*th element of $\delta^{(3)}$ shown above, and vectorized form below. Generalizing, for $l = 2 \ldots \mathrm{L} - 1$,

$$\delta^{(l)} = \left(\theta_j^{(l)} \right)^T \delta^{(l+1)} \odot \frac{da^{(l)}}{dz^{(l)}}$$

Gradient descent is used to update the parameters of the network. The learning procedure which incorporates both forward and backward propagation is shown below. Initialization of gradients for all unit pairs *ij* is with unit *j* in layer *l* and unit *i* in layer *l + 1*, for $l = 1 \ldots \mathrm{L} - 1$.

$$\Delta_{ij}^{(l)} = 0$$

For every example pair *(x,y)* in the training data

1. set $a^{(1)} = x$
2. use forward propagation to compute $a^{(2)},\ldots,a^{(L)}$
3. set $\delta^{(l)} = (a^{(L)} - y)$
4. use backpropagation to compute $\delta^{(l-1)},\ldots,\delta^{(2)}$
5. update gradients using $\Delta_{ij}^{(l)} \leftarrow \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ for every *i,j,l*
6. update parameter by gradient descent using $\theta_{ij}^{(l)} \leftarrow \theta_{ij}^{(l)} - \alpha \Delta_{ij}^{(l)}$ for every *i,j,l*

This algorithm can be extended to work not just on feedforward networks, but on general computation graphs, providing a third perspective on the algorithm. The nodes of a computation graph are operations (e.g., sums, products, reciprocals, exponentiations) and the leaves of the graph are the operands (the quantities upon which these operations are applied). The computation graph for the sigmoid of the dot product of two vectors $\theta, x \frac{1}{1+\exp\left(-\left(\theta_0+\theta_1 x_1+\theta_2 x_2\right)\right)}$ is shown in Fig. 6.27.

Schematically, this could represent the components of the neural network in Fig. 6.26 running from layer three to the output node, after receiving, summing, and transforming input from the previous layer. To keep the quantities manageable and readable, the inputs −1 and −3 are used, though in practice these would be values between 0 and 1 if a sigmoid activation function were used. Feeding forward (from left to right), these incoming values as well as a 1 representing the bias term are then multiplied by the weights of the third layer, $\theta^{(3)}$ (i.e. the vector [−3,1,−2]), which connects this layer to the output node. The resulting products are then added to generate the logit $z^{(4)}$. This logit provides the input to the sigmoid function (1/1 + exp(−x)), which in the computation graph is decomposed into a series of individual operations, first reversing the sign, exponentiating, adding one and taking the reciprocal. The result is $a^{(4)}$, which gives the output of the network for this example: a predicted probability of 0.88 that this example belongs to the positive class.

In order to update the weights of the network the cross-entropy loss (CE) is first measured. For an example from the positive class this is calculated as $-log(p)$, with p as the predicted probability. The next step is to proceed back across the graph (from right to left), multiplying this value by the derivative of the operation in the node concerned. These derivatives are as follows:

$$\text{For } f(z)=1/z, \frac{df(z)}{dz}=-1/z^2$$

$$\text{For } f(z)=z+c, \frac{df(z)}{dz}=1$$

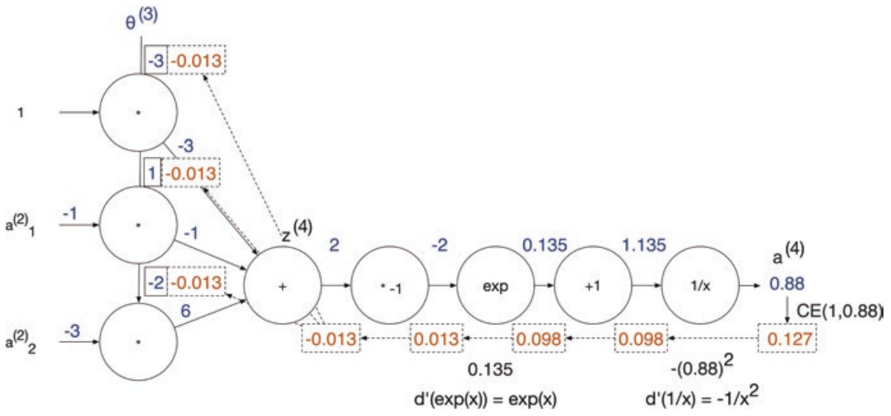


Fig. 6.27 The computation graph for the sigmoid of the dot product of two vectors is used to demonstrate how to run forward propagation and backward propagation for general network architectures

$$\text{For } f(z) = e^z, \frac{df(z)}{dz} = e^z$$

$$\text{For } f(z) = az, \frac{df(z)}{dz} = a$$

To compute the derivative of J with respect to the first operation (*1/x)*, the chain rule is used, and incoming derivative 0.127 is multiplied with the derivative of *1/x,* which is $-1/x^2 = -0.88^2$, to give 0.098. The derivative of the +1 operation is 1, so the incoming derivative 0.098 moves unchanged across that operation. Again, multiplying the incoming derivative 0.098 with $e^z = 0.135$, it is possible to propagate the product 0.013 across the *exp* operation. To propagate the incoming derivative 0.013 across the *−1 operation, the fourth derivative in the table of derivatives above is used, obtaining −0.013. Figure 6.28 shows how weights learned from training data using backpropagation can be used to approximate the XNOR function.

To propagate the derivative across the sum and product operations, recall that

$$\text{For}(x,y) = x + y, \frac{df}{dx} = 1, \frac{df}{dy} = 1$$

$$\text{For } f(x,y) = xy, \frac{df}{dx} = y, \frac{df}{dy} = x$$

To complete the example, it is evident that the incoming derivative is being propagated unchanged across the sum operator, and being multiplied by the input on the other branch across the product operation. This provides part of the information required to update each weight in $\theta^{(3)}$: the extent to which changing this weight would influence the loss function if it were multiplied by one in the forward pass. However, this is not the case—the weights in $\theta^{(3)}$ are multiplied by the vector [1,−1,−3]. This vector is multiplied by −0.013 (as well as the learning rate) to determine the update to each weight.

In modern deep learning frameworks, such as PyTorch and TensorFlow, one must specify only the forward propagation computation. The computation is internally represented as a directed acyclic graph and derivatives are propagated over the operations by the chain rule using automatically computed derivatives such as the ones in the tables above. More details on automatic differentiation can be found in Chap. 6 of Goodfellow et al.'s comprehensive deep learning text [9].

The core ideas behind the backpropagation algorithm have been around since the late 1980s [25, 26]. The success of deep learning networks today can be attributed to the ready availability of massive data sets needed to train the millions of parameters in modern networks, as well as efficient vector and matrix computations (including special-purpose (tensor processing) hardware) to accelerate the forward and backward computations. Another major innovation is the replacement of the sigmoid nonlinearity in networks by the rectified linear unit [27] (ReLU) defined as ReLU(x) = max(0,x). As can be observed in Fig. 6.25 the sigmoid function g(z)

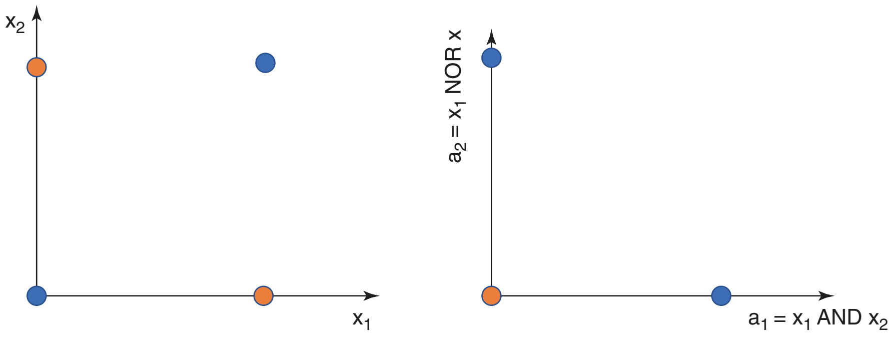


Fig. 6.28 In the original feature space (x_1,x_2) on the left, the four values of the XNOR function are depicted with orange and blue dots. Blue dots represent the value 1, and orange dots represent the value 0. Note that when x_1 and x_2 are the same (both 0 or both 1), the XNOR function has value 1. The blue and orange dots cannot be separated by a single hyperplane in the (x_1,x_2) space. However, in the (a_1,a_2) space shown on the right, where a_1 and a_2 are the intermediate outputs of the stacked assembly of logistic units shown in Fig. 6.26, the XNOR function is linearly separable and thus can be represented by a single output unit y. The new basis functions a_1 and a_2 can be learned automatically by the backpropagation algorithm from training data representing the XNOR function

saturates beyond a narrow range of z values about the origin (i.e., it asymptotes to 0 for negative z values, and to 1 for positive z values). Outside this narrow range, small changes to z (represented by the x axis on Fig. 6.25) will have negligible effects on the output (represented by the y axis on this figure). Hence, the derivative of the sigmoid function beyond this narrow interval is 0, and the gradient descent training algorithm essentially stalls. This problem is called the **vanishing gradient problem**. The use of the ReLU non-linearity significantly reduces this problem, since the gradient is one if the result of the linear logit computation is positive.

Convolutional Networks

This section describes the Convolutional Neural Network (CNN), a specialized deep neural network architecture that is especially effective at modeling imaging data and as such underlies many of advances in medical image processing that are described in Chap. 12. CNNs offer advantages over standard architectures in their ability to leverage the innate 2D correlational structure of image-related data sources. Multi-layer, fully connected feedforward networks of the appropriate depth and width have the power to represent any function from a set of inputs to an output set (continuous or discrete). However, the architecture forces all inputs, including those with 2D or 3D structure, such as still images and videos, to be flattened into one-dimensional vectors, for processing by the network. Spatial and temporal structure inherent in two-dimensional image arrays or three-dimensional video streams (the third dimension is time) are lost in this representational transformation. Convolutional neural networks preserve local correlations in the input and use convolution in place of full matrix multiplication in some of the layers. Convolution is a well-known mathematical operation in which a function called a kernel or filter is applied to an input, yielding an **activation map**. It is widely used in computer vision applications where kernels are designed to detect specific features (such as oriented edges) in images. Figure 6.29 shows a 2D kernel of size 2 × 2 applied to an input matrix of size 5 × 5, yielding a 4 × 4 activation map. The kernel or filter is swept horizontally across the input starting at the top left, with a specified shift (called stride) to the right, until it reaches the right edge of the image. The values in the 2 × 2 kernel are pointwise multiplied and summed with the 2 × 2 part of the image underneath it, yielding a scalar. In a trained model, this scalar indicates the extent to which this part of the image maps to a feature the filter has learned to identify. For instance, when the kernel is aligned with the left most corner of the input, the convolution yields 2 × 1 + 3 × 1 + 1 × 2 + 4 × 2 = 15. This is the first element of the activation map which is computed by sweeping the kernel across and then down by the specified stride. The 2 × 2 kernel yields four elements in the activation map for each horizontal sweep (as each position occupies two columns of the input matrix, there are four possible horizontal positions). Since the vertical stride is also one, the activation map is of size 4 × 4, with each cell indicating the strength of activation of the filter in one of its possible positions in the input matrix.

Convolution networks have far fewer parameters than a conventional feedforward network on the same inputs. Continuing with the example in Fig. 6.29, the number of units to represent the 5 × 5 input would be 26 (5 × 5 + 1 bias unit) and the number of units to represent the next layer, that is, the 4 × 4 activation map would be 17 (16 + 1). In a standard neural network architecture, every one of the 16 units in the activation layer would need to be connected to the 26 units below, yielding 16 × 26 = 416 parameters. Instead, there are just four parameters (the

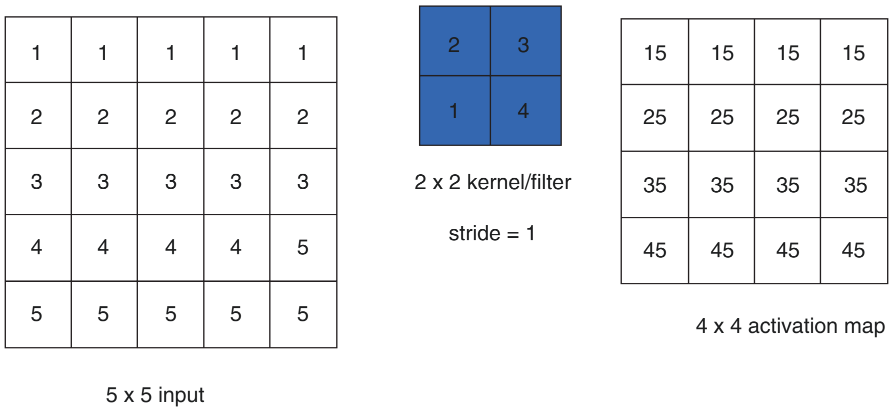


Fig. 6.29 An example of a 2D convolution with a 5 × 5 input matrix and a 2 × 2 kernel. The resulting activation map is of size 4 × 4 since the stride is 1. The first row of numbers in the activation map is generated by sweeping the kernel across the first two rows of the input starting at the left corner and moving one column horizontally to the right. The pointwise products of the filter values with the input values under the filter are taken and summed to yield the values in the activation map

kernel values or weights) capturing the entire interaction between the input and the activation layer. Since the kernel computes the same function across all the input units, the activation map values are akin to the output of a feature detector. Convolutional networks embody **translational invariance** because the kernel identifies specific features no matter where they occur in the input. This is intuitively appealing for image classification problems, as ideally the network would learn to recognize important features, such as cavitation in chest radiographs, irrespective of where they occur within a particular training image. The **parameter sharing** with the use of convolutional layers with small kernels or filters becomes even more compelling when working with larger inputs of size 224 × 224 × 3 (such as the ImageNet collection [28]—the "×3" indicates three channels, for red, green and blue color encoding). The input is padded with zeros around the edge and a 3 × 3 × 3 kernel is used to obtain an activation map of size 224 × 224 × 3 (224 + 2 – 3 + 1 for each dimension). A fully connected model will need 224 × 224 × 3 × 224 × 224 × 3 = 2.3 × 10^{10} parameters or weights, while the convolutional layer only has 3 × 3 × 3 = 27 parameters! By sweeping a small kernel by a small stride across a large image, **sparsity** is obtained in the connections between layers, because not every unit in a layer is connected to all units in the following layer. The filter thus defines a receptive field that moves across the entire image.

To specify a convolutional layer in a deep network with input of size H × H × D, several hyper-parameters must be defined: K, the number of filters or kernels, F, the size of the filter (typically a square matrix), S, the stride of the filter (typically 1 or 2), and P, a zero padding around the edges of the input to ensure that the activation map size $A \times A$ where $A = \frac{H + F - 2p}{S} + 1$ is an integer. The total number of parameters defining a convolutional layer for an input of size H × H × D is (K × F × F × D) parameters for the K filters and K bias parameters (one per filter).

Filter weights in convolutional networks trained by backpropagation can be visualized as color images, as shown in Fig. 6.30. The filters come to resemble the

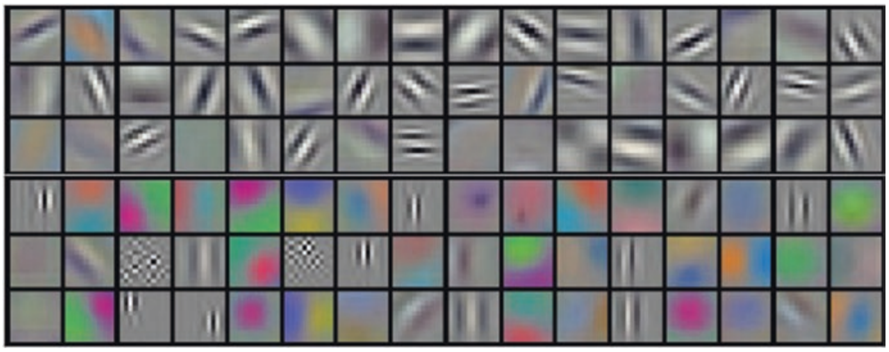


Fig. 6.30 A visualization of the filter weights of the first layer of a VGGNet trained on the ImageNet classification task [29]

features they detect, because the signal that propagates forward from a matrix multiplication will be highest when the matrices have similar values. This capability to learn feature representations from data is an important advantage of deep learning models, especially considering these feature representations can be of value for tasks beyond those the network was originally trained. Zeiler and Fergus [29] pioneered this visualization technique in the context of the ImageNet recognition task, rendering the 96 11 × 11 × 3 filters in the convolutional layer next to the input layer as color images of size 11 × 11 each. Oriented edge detectors and color patches are automatically learned by the machine, by minimizing the cross-entropy loss at the output layer and propagating loss functions derivatives back to the first layer. Similarly, activation maps at higher hidden layers can be projected back to the input, to reveal the regions of the input image that contribute most to the final classification decisions [30], as illustrated in Fig. 6.31.

The convolution operation is linear; and once the linear activation map is computed, a nonlinearity, such as ReLU, is applied. Convolutional/ReLU layers are generally followed by a pooling layer which reduces the dimensionality of the activation map. A commonly used pooling kernel is of size 2 × 2 with horizontal and vertical stride of two, which selects the maximum value in its receptive field. That is, only the signal from the region that most strongly activates a filter propagates forward to the next layer of the network. MaxPool is illustrated in Fig. 6.32, where it clearly functions as a non-linear downsampler. When a convolutional layer follows a pooling layer, the network learns filters on a wider receptive field than on the original input.

Classical convolutional networks for K-class object recognition problems such as VGGNet [32] are a sequence of convolution/ReLU/MaxPool layers that progressively map the input image through a series of reduced dimensional hidden outputs into a penultimate layer which is flattened and fully connected to an output layer of size K.

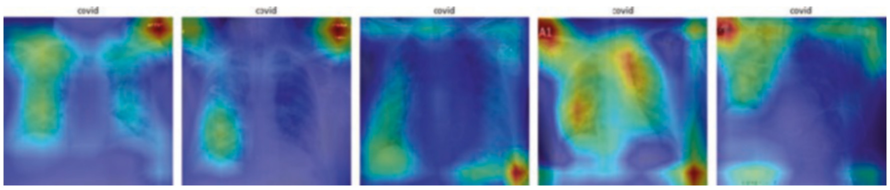


Fig. 6.31 GradCAM [30] visualization of examples from a popular and somewhat controversial set of radiological images used to train deep learning models to detect COVID-positive patients, with high accuracy reported in several evaluations. Pixels in red have the highest importance in the classification decision. Of note, the models are often attending to regions outside the lungs themselves, which contain metadata denoted in different ways across institutions. As "healthy control" counterexamples were often drawn from different sources to the COVID-positive cases, the ability to identify image provenance explains much of the model's ostensibly strong performance [31]

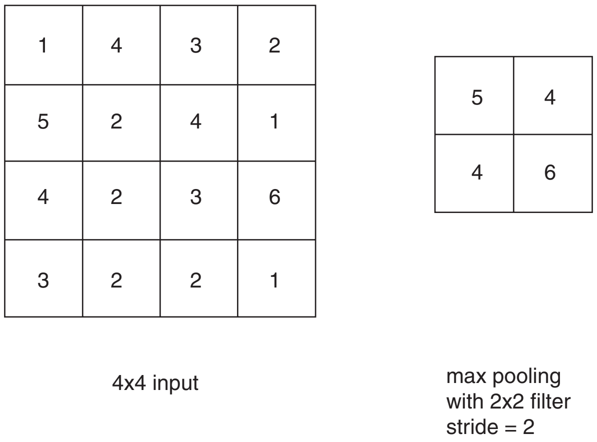


Fig. 6.32 The max pool filter selects the maximum value in a 2 × 2 block starting at the left corner of the input. The maximum value in the first 2 × 2 block of the input is 5. The next step is to stride the filter by 2 horizontally, and obtain 4 as the maximum value. The resulting output is a 2 × 2 matrix since the vertical stride of the filter is also 2. Max pooling reduces a n × n input matrix into a n/2 × n/2 output

Other Network Architectures

While convolutional networks with convolution/ReLU/MaxPool layers, with fully connected layers at the output form the dominant architectural paradigm, a host of variations have been proposed in the literature to solve problems beyond discrete object recognition. One class, called UNets [33] are specially engineered for solving image segmentation problems, i.e., problems in which exact localization of features is important, unlike a simple object recognition problem. Details of the UNet architecture are beyond the scope of this chapter, and the interested reader is directed to the original paper [33].

For handling time series data such as video streams in medical imaging, recurrent networks are the appropriate architecture. Unlike the feedforward systems studied thus far, in which the computations flow in one direction from inputs to the final outputs, recurrent networks allow the final output of a network to serve as input with a time delay. Chapter 10 of the Goodfellow et al. text [34] on deep learning offers an excellent introduction to this family of networks, and its many variations such as LSTMs and echo state networks.

In natural language processing, the Transformer architecture has emerged as an important approach to processing text sequences underlying widely used models such as **Bidirectional Encoder Representations from Transformers (BERT)** [35, 36]. Transformers generate context-specific representations of textual input, by allowing words (or parts of words) in a sequence to influence the representations of other words as they progress through the network. This provides an intuitive way to model ambiguous words such as "cold" (virus vs. temperature), because representations of this word that are close to the output layer will be informed by contextual cues. The relative influence of specific contextual cues is learned by the model, so that those cues that are useful for particular tasks will be emphasized when the model is fine-tuned accordingly. For discussion of some key applications of neural Transformer models, see Chap. 7.

Putting It All Together: The Workflow for Training Deep Neural Networks

Deep learning models have been used in a wide range of clinical applications ranging from diagnosis, to risk assessment and treatment. A large fraction of them are end-to-end models that start with images (optical, CT, Xray) as inputs and culminate in a classification output layer with intervening hidden layers consisting of convolutional/ReLU/pooling layers together with fully connected feedforward layers at the end of the pipeline. In this section, the workflow for setting up and training deep neural networks for supervised learning problems in clinical applications is elaborated.

1. **Decide on the network architecture**: for a supervised learning problem, the type and arrangement of layers is determined by the inputs and outputs and the nature of the prediction problem: e.g., whether it is classification into a small set of discrete categories, image segmentation, or risk estimation. Modern deep learning frameworks such as Pytorch [37], Keras (keras.io) and TensorFlow (tensorflow.org) allow easy parametric specification of layers ranging from simple fully connected to convolution/ReLU and MaxPool composites, as well as more exotic layers to support specialized applications. It is important to avoid the data arrogance trap, particularly if the available training sets are orders of magnitude smaller in size than the number of parameters in the network model. The use of pre-trained networks with parameters optimized for related tasks is crucial to obtain robust generalization performance. An example is the use of a VGGNet and Resnet architecture trained on the ImageNet dataset with 1.2 million images in 1000 categories as a feature extractor for classification of pneumonia from chest X-ray images [38].
2. **Choose an appropriate loss function**: for regression problems, mean squared error is the standard choice, while cross-entropy loss is the usual choice for binary classification. The softmax loss function, which is a generalization of cross-entropy loss, is used for multi-class classification. Weighted versions of these loss functions are available in standard deep learning frameworks, allowing designers to accommodate problems with class imbalance, or problems where the costs associated with false positive errors and false negative errors are quite different.

3. **Choose a regularization approach**: One technique for regularization is to use penalized loss functions, where the L1 or L2 norm of the network weights is added to the chosen loss function. Another technique is called dropout [39], and can be added as a layer in modern deep learning frameworks. During training, some of the units (and their connections) are stochastically dropped during weight update, which encourages sparsity in the network weights.
4. **Initialize network parameters**: weights on bias units are typically initialized to zero. The most used technique [40] for initializing weights connecting units in layers *l−1* and *l* is to select them from a uniform distribution $[-b,+b]$ where $b = \sqrt{6} / \sqrt{S^{(l)} + S^{(i-1)}}$. $S^{(l)}$ denotes the number of hidden units in layer *l* of the network. Proper initialization of network parameters is still an open problem in the field.
5. **Select pre-processing steps for training data**: to make the inputs well-conditioned, it is traditional to normalize training inputs, e.g., subtracting means from images, or more generally, using standard scaling of each column of the input to make it have zero mean and unit variance. The choice of pre-processing step requires domain knowledge and understanding of how the inputs were generated, and the elimination of input artifacts that could cause overfitting in the models. In many medical image classification problems, such as Gleason grading of prostate cancer from whole slide images [41], semi-automated label cleaning is employed, in which erroneously graded training examples are excluded from the training sets. Another significant pre-processing step in image classification tasks is to break up an input image into smaller patches and learn models on the patches. A second model learns to integrate feature responses from the patches to make a final classification.
6. **Determine if data augmentation is needed**, and if it is, determine how to augment training data. If the number of parameters in the chosen network architecture far exceeds (i.e., is an order of magnitude greater than) the product of the number of training examples and the size of each example, there is a need to augment the training set. One of the easiest ways to perform data augmentation to is apply affine transformations: translations and small rotations to the existing training set data to force the network to be robust in the face of perturbations of the input. Yet another approach is to inject a small amount of white noise to all the inputs to encourage better generalization performance.
7. **Decide on a stopping criterion**: It is customary to set aside a small portion of the training set, called a validation set, and calculate the loss function on both the training and validation sets for each epoch of training. In a single epoch, network parameters are updated after iterating through the whole training set. Training loss decreases with the number of training epochs, eventually tending to zero. The validation loss, on the other hand, first decreases and then increases, indicating that the network has been overfitted to the training data. The optimal stopping point for training is when the validation loss achieves its minimum.

8. **Tuning the learning hyper-parameters**: these include choice of learning rate, the gradient descent optimization algorithm, and example batch size for gradient estimation. Tuning hyper-parameters is still an art and is extremely computationally intensive. Close monitoring of the training and validation loss using a visualization framework such as Tensorboard[16] is crucial to find good values for the hyper-parameters. Tools such as AutoML[17] can automatically perform coarse to fine grained searches in the hyper-parameter space to find good combinations of values.
9. **Train the model**: Run the model with the (augmented) training data with the chosen hyper-parameters and architecture until the optimal stopping point.
10. **Test the model**: evaluate the predictive performance of the model on the set aside test set (or on the set aside chunk for N-fold cross-validation). In datasets with class imbalance, cross-validation has to be designed carefully to avoid overfitting and overestimating model performance [42]. Generating artificial examples to overcome imbalance in classification problems can introduce inadvertent biases in the model. An interesting example of this phenomenon occurs in the domain of predicting synergistic drug interactions. While there are tens of thousands of known drug compounds, very few documented examples of synergistic interactions exist (rare class problem). Further, drug pairs that do not have synergistic interaction are never documented. Researchers often use random drug pairs as negative examples. These models rarely perform well outside the training set, because they merely learn to distinguish random drug pairs from ones that have synergistic interactions, instead of generalizing patterns present in useful drug combinations [43]. Deep learning models in computer vision are vulnerable to adversarial attacks in which small perturbations in inputs cause large variations in outputs [44] (such as misclassifying a stop sign image with a few pixel alterations as a 30 mph speed limit sign). In healthcare predictive analytics, algorithms for generating adversarial examples for biomedical text classification have been devised [45] to test the robustness of deep models. Adversarial example generation for healthcare applications is an active area of research.
11. **Interpret/Visualize the model**: Visualize the network weights and generate activation maps as well as GradCAM maps for both correctly classified and incorrectly classified members of the test set to build an understanding of the generalization performance of the network model. GradCAM maps reveal whether relevant areas of the input contribute to the final decision made by the network. When irrelevant features (such as a date or patient name on a clinical image) are highlighted by GradCAM, the input data is reengineered to eliminate these noise features and the system retrained on the cleaned data.

[16] tensorflow.org/tensorboard (accessed August 19, 2022)
[17] automl.org (accessed August 19, 2022)

Ensembling Models

So far, several model families for supervised learning have been presented, ranging from simple models such as linear and logistic regression with basis function expansion, dense and sparse kernel methods which project examples into appropriately chosen similarity spaces, and non-linear adaptive basis functions with universal approximation properties, exemplified by deep neural networks. For each family, a loss function was optimized to obtain the best model for the given labeled data set. In this section, model ensembles are introduced. Ensembles improve prediction by combining several models by weighted averaging for regression models or simple majority/weighted majority voting for classification models. Two conditions are necessary for an ensemble of classification models to perform better than a single model. First, the error rate (i.e., probability of misclassification) of each model in the ensemble must be less than 0.5. Second, the errors made by each member of the ensemble must be uncorrelated with the others. If the highest error rate of an individual binary classifier in an ensemble of size L is ϵ, then the error rate of the entire ensemble with simple majority voting is

$$\sum_{i=L/2+1}^{L} \binom{L}{i} \epsilon^{i} (1-\epsilon)^{L-i}$$

For $L = 21$ and $\epsilon = 0.3$, the error rate of the ensemble is 0.026! This is a direct consequence of the strong assumption that the errors of the ensemble members are uncorrelated. This assumption, unfortunately, generally does not hold for human committees, leading to the general belief that committee decisions are inferior to an individual member's decision. Thus, the key to making good ensembles is to devise ways to decorrelate the errors of individual members.

There are two major approaches to constructing ensembles: **bagging** and **boosting**.

- In bagging [46], L bootstrap samples are created from the given training data set D with m elements of the form (x,y). A bootstrap sample is constructed by uniformly sampling m times, with replacement, from D. A bootstrap sample has the same size as the original dataset D but may have duplicates. L classifiers are constructed with each of the bootstrap samples, and a simple majority rule is used for final classification. Bagging can be easily parallelized since the construction of the bootstrap sample and the associated classifier can occur independently. The random forest algorithm [47] builds bagged ensembles of decision trees and it has found wide acceptance in medicine because of its impressive performance in clinical decision-making tasks [48].
- In boosting [49], ensemble members are learned sequentially, with each member focusing on the errors made by the previously learned members of the ensemble. Weights $w^{(i)}$ are associated with every example $(x^{(i)}, y^{(i)})$ in a dataset

D containing *m* pairs. Initially all weights are 1, and a classification algorithm that minimizes weighted cross-entropy loss is learned in each round. Examples misclassified by a classifier are weighted higher for the next classifier in the sequence; examples correctly classified are down weighted for the next classifier. The same data set *D* with the new weights is used to learn the next classifier in the sequence, with the process terminating with the error rate of the learned classifier exceeds 0.5. Finally, the predictions are combined by a weighted voting scheme where each classifier's voting weight reflects its overall predictive accuracy. There are many boosting algorithms in the literature [49–51]. Each of them is characterized by (1) specification of the initial example weights, and how weights are up- and down-weighted after each round of classifier learning, (2) how the voting weight of each classifier in the ensemble is determined, (3) the specific loss function (e.g., weighted cross-entropy loss) for learning each classifier, and (4) a termination criterion (which determines how many members will be included in the boosted ensemble). The most popular boosting algorithm in use today is XGBoost [52]—it is readily scalable to large data sets and has achieved state-of-the-art results on many machine learning challenges. A recent example of its use is the prediction of adverse outcomes in Type 2 diabetes patients with administrative health data [53]: on a training dataset of over a million patients, an XGBoost model on over 700 features extracted from administrative data predicted 3-year risk of diabetes complications in with an AUROC of 0.77 on held out validation and test sets of over a quarter million patients.

Conclusion

This chapter has introduced supervised machine learning algorithms for solving clinical decision-making problems with labeled data. The types of problems that are best suited for supervised learning and workflow sequence for model construction and validation have also been identified. Although machine learning systems have shown success in a range of retrospective studies, relatively few are deployed in practice. An interesting exception is Google's neural network detector of diabetic retinopathy in retinal fundus photographs [2]. One of the many challenges faced in translation of research algorithms to the clinical context is that systems are often trained on data that are subject to extensive cleaning and curation, and thus quite unlike data in a real-world clinical setting.

Randomized controlled trials and prospective studies are now being pursued to ease the transition from the computational lab to patient bedside. More refined, context-specific measures of performance, beyond F1-scores and AUROCs, are being developed for evaluating ML systems. A recent study uses the percentage of time pediatric Type 1diabetic patients spend inside their target glucose range as a way of evaluating a learning system that manages real-time insulin dosing [54]. Simple linear models such as logistic regression have the potential for ready deployment since the coefficients of the model can be easily converted into a score card system for risk stratification. A recent review of clinical prediction models shows the wide-spread use of logistic models in medicine [55].

The most visible recent successes of ML have been in image interpretation with deep neural networks, in the domains of radiology, pathology, gastroenterology and ophthalmology. In these problem areas, clinicians generally find it difficult to articulate decision-making criteria for classification. Thus, end-to-end learning systems such as deep neural nets that take pairs of the form *(raw image, overall decision)* as training inputs, and learn appropriate intermediate features by optimization of well-chosen loss functions, are a winning alternative that have even outperformed clinicians in some evaluations [56, 57].

An open problem is how to get doctors, as well as patients, to trust the decisions made by ML systems. Clearly, interpretable and explainable models will be key (see Chap. 8), and stronger prospective validation guidelines developed jointly by ML scientists and by clinicians, and then endorsed by regulatory bodies, will go a long way to bridging the trust gap. For example, the use of GradCAM visualizations of deep neural net image classification models have been important for convincing clinicians and regulators of the validity of a model's decisions beyond performance scores such as AUROC and F1. Equally important are ethical considerations concerning data use and equity (see Chap. 18), particularly the need for standards of diversity and inclusion in the design of training data for machine learning systems. A 2021 study of underreporting and underrepresentation of diverse skin types in present-day skin cancer databases reveals gaps in training sets that limit the applicability of predictive models for people of color [58].

Many technical, legal, ethical and regulatory problems need to be addressed before predictive ML systems are routinely incorporated into clinical workflow (see Chaps. 17 and 18). There are open questions in accountability assignment: who is to be held responsible for a model's mistakes? Do we turn to the ML engineers who build the model, the clinicians who use the model, the regulators who cleared the model for use, or others? As these issues are raised and solved in specific clinical contexts, supervised learning will be a major enabler of improved access to high-quality healthcare at a global scale.

Questions for Discussion

- How does one integrate prior knowledge about a clinical decision-making problem in the formulation of a supervised learning approach to it? Under what circumstances are we likely to obtain high performing models using data alone?
- One of the few useful theoretical results in supervised machine learning is the "no free lunch" theorem [59]—there is no single best model that performs optimally for all problems. Do deep neural networks with their universal approximation properties negate this theorem?

- Modern machine learning algorithms can build high-performing models by picking up on incidental correlations in training data. An apocryphal story from the early days of machine learning is about a neural network learning algorithm that distinguished images of enemy tanks from friendly tanks by picking up the blue skies at the top edge of the friendly tank images. Flushing out confounding variables in a high-dimensional dataset is still an art. Can you provide examples of confounders in clinical decision-making tasks? How can one systematically eliminate such features from consideration during model construction?
- The ImageNet dataset has 12 million examples of over a thousand object categories, and the best performing neural nets trained on ImageNet (an ensemble of Resnet50 networks) have error rates of under 3% on set-aside test sets. Why do deep neural networks require millions of examples to learn robust models of objects, when humans can generalize from very few examples? Why is it that humans generalize so well with very few examples? Hint: a new area called few-shot learning concerns an attempt to reduce the sample complexity of deep neural networks.
- What, in your opinion, are the primary barriers to the adoption of machine learning systems in a clinical context? Are the barriers lower in some areas of medicine than in others? If so, why?
- Obtaining high quality labeled data is a bottleneck in the design of supervised machine learning systems for clinical decision-making. The quality of the learned model is determined completely by the quality of the associated labels/decisions associated with each case. What approaches can be used to assess consistency and quality of data labels before one embarks on model construction?
- What are potential uses of unsupervised learning (learning from unlabeled data) in the clinical context?

Further Reading

Goodfellow I, Bengio G, Courville A. Deep learning. MIT Press; 2016.

- The definitive text on deep learning available online at deeplearningbook.org. It has three major parts. Part 1 is a concise yet comprehensive of review of all the mathematics needed to understand machine learning algorithms and a summary of ML algorithms before the deep learning era. Part 2 is a deep dive into modern deep learning networks starting from feedforward multilayer networks through convolutional networks and recurrent networks. This part combines a clear exposition of the theoretical foundations of deep networks with practical tips on network design and training. Part 3 covers advanced topics including representation learning, autoencoders, and deep generative models, including generative adversarial networks.

Murphy K. Probabilistic machine learning: an introduction. MIT Press; 2022.

- A new two volume, comprehensive, reference textbook from an authority in the field, available online at probml.ai. The first book covers the foundational mathematics, linear models for regression and classification, deep neural networks, non-parametric models including ensemble models and unsupervised learning. The second book, to be released in 2023, will cover advanced topics in prediction, generative models, causality, and reinforcement learning.

Bishop CM. Pattern recognition and machine learning. Springer; 2021 (old edition 2006 available online).

- An extremely well-written textbook on classical machine learning algorithms including feedforward neural networks. The latest edition covers graphical models and approximate inference.

James G, Witten D, Hastie T, Tibshirani R. An Introduction to Statistical Learning with Applications in R. 2nd ed. Springer; 2017.

- A basic textbook on machine learning which goes deep into linear and logistic regression, tree-based models, basis function expansion, ensemble techniques and clustering methods. It has excellent practical end-of-chapter exercises. It is available as a free download online at hastie.su.domains/ElemStatLearn.

Nielsen M. Neural networks and deep learning, online book at neuralnetworksanddeeplearning.com.

- This book is an excellent introduction to neural networks. It has the clearest explanation of backpropagation and through a hands-on approach elucidates why neural networks are universal function approximators. This book should be required reading for all machine learning enthusiasts.

References

1. Johnson AEW, et al. MIMIC-III, a freely accessible critical care database. Sci Data. 2016;3:160035.
2. Gulshan V, et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. JAMA. 2016;316:2402–10.
3. Esteva A, et al. Dermatologist-level classification of skin cancer with deep neural networks. Nature. 2017;542:115–8.
4. Golden JA. Deep learning algorithms for detection of lymph node metastases from breast cancer: helping artificial intelligence be seen. JAMA. 2017;318:2184–6.
5. FitzHenry F, et al. Creating a common data model for comparative effectiveness with the observational medical outcomes partnership. Appl Clin Inform. 2015;6:536–47.
6. Hersh W. Information retrieval: a health and biomedical perspective. Springer; 2008.
7. Kelly-Hayes M. Influence of age and health behaviors on stroke risk: lessons from longitudinal studies. J Am Geriatr Soc. 2010;58:S325–8.
8. Otis AB, Fenn WO, Rahn H. Mechanics of breathing in man. J Appl Physiol. 1950;2:592–607.
9. Hastie T, Tibshirani R, Friedman JH, Friedman JH. The elements of statistical learning: data mining, inference, and prediction, vol. 2. Springer; 2009.

10. Walsh C, Hripcsak G. The effects of data sources, cohort selection, and outcome definition on a predictive model of risk of thirty-day hospital readmissions. J Biomed Inform. 2014;52:418–26.
11. Milea D, et al. Artificial intelligence to detect papilledema from ocular fundus photographs. N Engl J Med. 2020;382:1687–95.
12. Howell K, et al. Controlling for confounding variables: accounting for dataset bias in classifying patient-provider interactions. In: Shaban-Nejad A, Michalowski M, Buckeridge DL, editors. Explainable AI in healthcare and medicine: building a culture of transparency and accountability. Springer; 2021. p. 271–82. https://doi.org/10.1007/978-3-030-53352-6_25.
13. Doll R, Hill AB. Smoking and carcinoma of the lung. Br Med J. 1950;2:739.
14. Ioannou GN, et al. Development of COVIDVax model to estimate the risk of SARS-CoV-2–related death among 7.6 million US veterans for use in vaccination prioritization. JAMA Netw Open. 2021;4:e214347.
15. Dooling K, et al. The Advisory Committee on Immunization Practices' interim recommendation for allocating initial supplies of COVID-19 vaccine—United States, 2020. Morb Mortal Wkly Rep. 2020;69:1857.
16. Barak-Corren Y, et al. Validation of an electronic health record–based suicide risk prediction modeling approach across multiple health care systems. JAMA Netw Open. 2020;3:e201262.
17. Joshi R, et al. Predicting neonatal sepsis using features of heart rate variability, respiratory characteristics, and ECG-derived estimates of infant motion. IEEE J Biomed Health Inform. 2019;24:681–92.
18. McCoy TH, Perlis RH. A tool to utilize adverse effect profiles to identify brain-active medications for repurposing. Int J Neuropsychopharmacol. 2015;18.
19. Istrail S, Pevzner PA. Kernel methods in computational biology. MIT Press; 2004.
20. McCulloch WS, Pitts W. A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys. 1943;5:115–33.
21. Nielsen M. Deep learning. 2017. http://neuralnetworksanddeeplearning.com/.
22. Cybenko G. Approximation by superpositions of a sigmoidal function. Math Control Signals Syst. 1989;2:303–14.
23. Hornik K. Approximation capabilities of multilayer feedforward networks. Neural Netw. 1991;4:251–7.
24. Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. Adv Neural Inf Process Syst. 2012;25:1097–105.
25. Rumelhart DE, McClelland JL, Group PR. Parallel distributed processing, vol. 1. New York: IEEE; 1988.
26. Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. Nature. 1986;323:533–6.
27. Jarrett K, Kavukcuoglu K, Ranzato M, LeCun Y. What is the best multi-stage architecture for object recognition? In: 2009 IEEE 12th international conference on computer vision. IEEE; 2009. p. 2146–53.
28. Fei-Fei L, Deng J, Li K. ImageNet: constructing a large-scale image database. J Vis. 2009;9:1037.
29. Zeiler MD, Fergus R. Visualizing and understanding convolutional networks. In: Fleet D, Pajdla T, Schiele B, Tuytelaars T, editors. Computer vision—ECCV 2014. Springer; 2014. p. 818–33.
30. Selvaraju RR, et al. Grad-cam: visual explanations from deep networks via gradient-based localization. In: Proceedings of the IEEE international conference on computer vision; 2017. p. 618–26.
31. López-Cabrera JD, Orozco-Morales R, Portal-Diaz JA, Lovelle-Enríquez O, Pérez-Díaz M. Current limitations to identify COVID-19 using artificial intelligence with chest X-ray imaging. Health Technol. 2021;11:411–24.

32. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. ArXiv Prepr. ArXiv14091556. 2014.
33. Ronneberger O, Fischer P, Brox T. U-net: convolutional networks for biomedical image segmentation. In: International conference on medical image computing and computer-assisted intervention. Springer; 2015. p. 234–41.
34. Goodfellow I, Bengio Y, Courville A. Deep learning. MIT Press; 2016.
35. Vaswani A, et al. Attention is all you need. Adv Neural Inf Process Syst. 2017;30.
36. Devlin J, Chang M-W, Lee K, Toutanova K. Bert: pre-training of deep bidirectional transformers for language understanding. ArXiv Prepr. ArXiv181004805. 2018.
37. Paszke A, et al. Pytorch: an imperative style, high-performance deep learning library. Adv Neural Inf Process Syst. 2019;32.
38. Victor Ikechukwu A, Murali S, Deepu R, Shivamurthy RC. ResNet-50 vs VGG-19 vs training from scratch: a comparative analysis of the segmentation and classification of Pneumonia from chest X-ray images. Glob Transit Proc. 2021;2:375–81.
39. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res. 2014;15:1929–58.
40. Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings; 2010. p. 249–56.
41. Bulten W, et al. Artificial intelligence for diagnosis and Gleason grading of prostate cancer: the PANDA challenge. Nat Med. 2022;28:154–63.
42. Santos MS, Soares JP, Abreu PH, Araujo H, Santos J. Cross-validation for imbalanced datasets: avoiding overoptimistic and overfitting approaches [research frontier]. IEEE Comput Intell Mag. 2018;13:59–76.
43. Zitnik M, Agrawal M, Leskovec J. Modeling polypharmacy side effects with graph convolutional networks. Bioinformatics. 2018;34:i457–66.
44. Eykholt K, et al. Robust physical-world attacks on deep learning visual classification. In: 2018 IEEE/CVF conference on computer vision and pattern recognition. IEEE; 2018. p. 1625–34. https://doi.org/10.1109/CVPR.2018.00175.
45. Mondal I. BBAEG: towards BERT-based biomedical adversarial example generation for text classification. In: Proceedings of the 2021 conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies; 2021. p. 5378–84.
46. Breiman L. Bagging predictors. Mach Learn. 1996;24:123–40.
47. Breiman L. Random forests. Mach Learn. 2001;45:5–32.
48. Subudhi S, et al. Comparing machine learning algorithms for predicting ICU admission and mortality in COVID-19. NPJ Digit Med. 2021;4:1–7.
49. Schapire RE. The boosting approach to machine learning: an overview. In: Denison DD, Hansen MH, Holmes CC, Mallick B, Yu B, editors. Nonlinear estimation and classification. Springer; 2003. p. 149–71. https://doi.org/10.1007/978-0-387-21579-2_9.
50. Friedman JH. Stochastic gradient boosting. Comput Stat Data Anal. 2002;38:367–78.
51. Friedman J, Hastie T, Tibshirani R. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). Ann Stat. 2000;28:337–407.
52. Chen T, Guestrin C. XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. Association for Computing Machinery; 2016. p. 785–94. https://doi.org/10.1145/2939672.2939785.
53. Ravaut M, et al. Predicting adverse outcomes due to diabetes complications with machine learning using administrative health data. NPJ Digit Med. 2021;4:1–12.
54. Nimri R, et al. Insulin dose optimization using an automated artificial intelligence-based decision support system in youths with type 1 diabetes. Nat Med. 2020;26:1380–4.
55. Chen L. Overview of clinical prediction models. Ann Transl Med. 2020;8:71.
56. Zhou D, et al. Diagnostic evaluation of a deep learning model for optical diagnosis of colorectal cancer. Nat Commun. 2020;11:2961.

57. Gong D, et al. Detection of colorectal adenomas with a real-time computer-aided system (ENDOANGEL): a randomised controlled study. Lancet Gastroenterol Hepatol. 2020;5:352–61.
58. Guo LN, Lee MS, Kassamali B, Mita C, Nambudiri VE. Bias in, bias out: underreporting and underrepresentation of diverse skin types in machine learning research for skin cancer detection—a scoping review. J Am Acad Dermatol. 2021.
59. Wolpert DH. The supervised learning no-free-lunch theorems. In: Roy R, Köppen M, Ovaska S, Furuhashi T, Hoffmann F, editors. Soft computing and industry: recent applications. Springer; 2002. p. 25–42. https://doi.org/10.1007/978-1-4471-0123-9_3.