Chapter 7

Introduction to Machine Learning

Yalın Baştanlar and Mustafa Özuysal

Abstract

The machine learning field, which can be briefly defined as enabling computers make successful predictions using past experiences, has exhibited an impressive development recently with the help of the rapid increase in the storage capacity and processing power of computers. Together with many disciplines, machine learning methods have been widely employed in bioinformatics. The difficulties and cost of biological analyses have led to the development of sophisticated machine learning approaches for this application area. In this chapter, we first review the fundamental concepts of machine learning such as feature assessment, unsupervised versus supervised learning and types of classification. Then, we point out the main issues of designing machine learning experiments and their performance evaluation. Finally, we introduce some supervised learning methods.

Key words Machine learning, Supervised learning, Unsupervised learning, Clustering, Classification, Regression, Model complexity, Model evaluation, Performance metrics, Dimensionality reduction

1 Introduction

1.1 What Is Machine Learning?

In many scientific disciplines, the primary objective is to model the relationship between a set of observable quantities (inputs) and another set of variables that are related to these (outputs). Once such a mathematical model is determined, it is possible to predict the value of the desired variables by measuring the observables. Unfortunately, many real-world phenomena are too complex to model directly as a closed form input–output relationship. Machine learning provides techniques that can automatically build a computational model of these complex relationships by processing the available data and maximizing a problem dependent performance criterion. The automatic process of model building is called "training" and the data used for training purposes is called "training data." The trained model can provide new insights into how input variables are mapped to the output and it can be used to make predictions for novel input values that were not part of the training data.

To be able to learn an accurate model, machine learning algorithms often require large amounts of training data. Therefore, an important first step in using machine learning techniques is to collect a large set of representative training examples and store it in a form that is suitable for computational purposes. Recent advances in digital data gathering, storage, and processing capacity have made the application of machine learning possible in many domains such as medical diagnosis, bioinformatics, chemical informatics, social network analysis, stock market analysis, and robotics.

There is usually more than one computational model that can be trained for a given machine learning problem. Unfortunately, there is no fixed rule to select a particular model or an algorithm. The performance of a specific model depends on many factors such as the amount and quality of training data, the complexity and form of the relationship between the input and output variables, and computational constraints such as available training time and memory. Depending on the problem, it is often necessary to try different models and algorithms to find the most suitable ones. Fortunately, there are standard software packages that combine different algorithms into the same framework such as [1–4]. Once the available data is prepared in a suitable format, these packages make it simpler to try the different alternatives.

As an example, consider the problem of labeling a candidate nucleotide sequence as miRNA or not. One simple approach would be to determine a set of short nucleotide sequences that are parts of the known miRNA and non-miRNA sequences and to construct a set of rules based on the existence of these nucleotide "words." For example, one such rule can state that a sequence containing "AGCACU" is more likely to be a miRNA than not. Then one could simply label candidate sequences using these rules. In practice, constructing such a rule based system is very difficult as there are many possible nucleotide words and the mapping is very complex. Instead of manually specifying a complex set of rules, machine learning methods can automatically build a statistical model using these nucleotide words. These models can then be trained using large samples of biological data since the training process is automated. For machine learning, such rules (here a nucleotide hexamer) are determined from features which need to be defined for the input data.

1.2 What Are Features?

The observable quantities that are input to a machine learning algorithm are called "features." The algorithm learns a mapping from these features to the desired output variables by tuning the model parameters using the available training data. Therefore, it is important that the features are relevant to the prediction of the outputs.

For some machine learning problems, there are thousands of features that can be used to predict the output variables, e.g., gene expression in microarray experiments can be considered as features (*see* Chapters 6, 17, and 18). However, using all available features may not be the best approach. Features that are loosely related to the output might adversely affect the learning process by decreasing the effect of the important ones. Features that are strongly coupled with other features do not provide extra information and unnecessarily bias the result. These can further lower training performance by straining computational resources such as time and memory.

The first step in selecting good features is using expert judgment. An expert that knows the problem domain well can select a compact set of relevant features for input to the machine learning algorithm. This is especially important in the data gathering stage since collecting training data can be time consuming and costly. However, extra caution is required not to eliminate potentially important features. It is important to note that feature selection and extraction requires experience and is often an iterative process. As additional insight into the problem is gained, it might be necessary to add or remove features to improve the performance [5]. It is also possible to automate this feature selection and extraction process. Such automated techniques are detailed in Subheading 2.5.

For the miRNA identification problem, features can be the existence or the frequency of a selected set of nucleotide sequence "words" of small length within the candidate sequence. Again it is important to include all the available information that might help with the prediction. So in actuality, more features such as those that describe the number of base pairs, bulges, loops and asymmetric loops in different parts of the candidate sequence may also be included in the analysis [6].

After features, that well model the problem, have been defined, machine learning algorithms need to be chosen and in the field of miRNA detection supervised methods have been applied widely (*see* Chapters 10, 12, and 15–18).

1.3 What Is Unsupervised Versus Supervised Learning?

Machine learning techniques can be broadly classified into two main categories depending on whether the output values are required to be present in the training data.

1.3.1 Unsupervised Learning

Unsupervised learning techniques require only the input feature values in the training data and the learning algorithm discovers hidden structure in the training data based on them. Clustering techniques that try to partition the data into coherent groups fall into this category. In bioinformatics, these techniques are used for problems such as microarray and gene expression analysis. In general, market segment analysis, grouping people according to their social behavior, and categorization of articles according to their topic are popular tasks involving clustering. Typical clustering algorithms are K-means [7], hierarchical clustering [8], and spectral clustering [9].

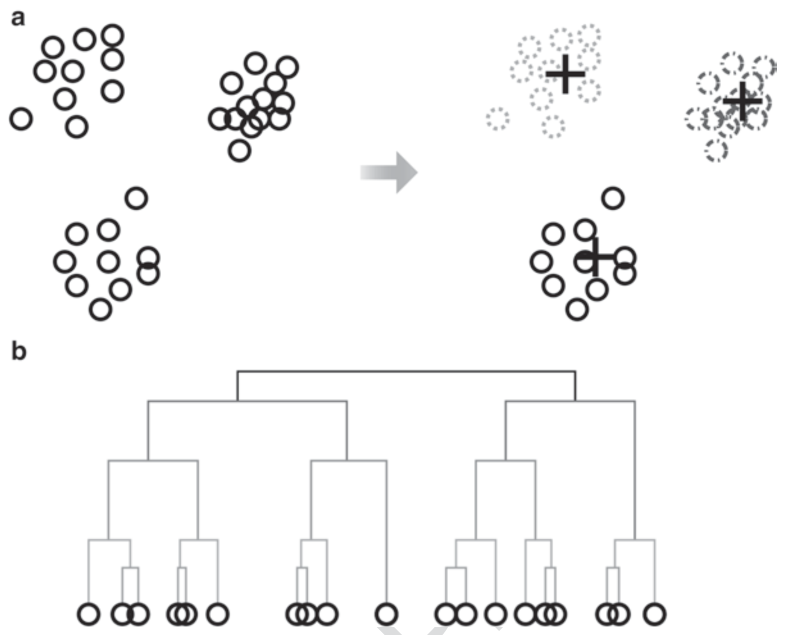

Fig. 1 Unsupervised clustering of data points (marked with *circles*). (**a**) The K-means algorithm groups the data into a given number of clusters (*K*) such that each data point is closer to the mean of its own cluster (depicted by *plus signs*) than any other cluster's centroid. (**b**) The hierarchical clustering method performs multiple rounds of clustering; merging the closest clusters or dividing the clusters of points at each round. The resulting clusters can be analyzed at multiple scales to find meaningful structures in the data distribution

It is not possible to directly measure the performance of clustering because the correct output labels are not known a priori. Instead, the performance depends on whether interesting trends in the data have been captured by the clusters or not. Since the output labels are not needed, it is often easier to collect a large training dataset for unsupervised algorithms.

Figure 1a shows an example result of clustering using the K-means algorithm. Let us briefly explain the steps of the algorithm. Firstly, user needs to define the number of clusters and initializes the centroid of each cluster (usually performed in a random manner). Then, each sample is assigned to the closest cluster centroid (cluster assignment step) and cluster centroids are recomputed using assigned samples (move centroids step). These two steps are iteratively performed until no further changes occur. Hopefully, in the end the clusters are well separated. However, K-means can get stuck in local optimum due to an unlucky initialization. Also, it is not very effective when the number of clusters (K) is not clear.

Hierarchical clustering is more suitable for the cases where the clusters are not well separated, i.e., the number of clusters is not obvious.

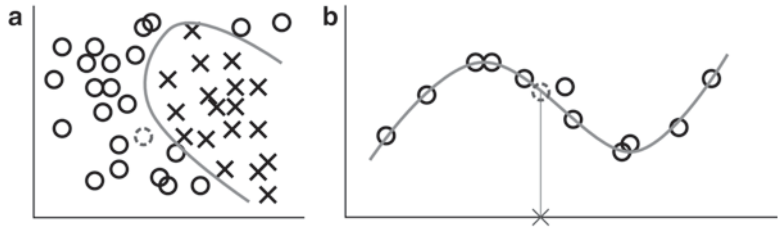

Fig. 2 Supervised machine learning problems. (**a**) In a classification problem, the training data belongs to one of several possible classes (the *solid circles* or *crosses*). A decision boundary (the *curve*) that best separates these data points is learned during training. At testing time, a novel data point (*dashed circle*) is classified as belonging to one of the classes depending on which side of the decision boundary it is on. (**b**) The goal in regression problems is to find a mapping from the inputs to the continuous output variable. A regression function (the *solid curve*) is fit to the training data (the *solid circles*). Afterwards it can be used to transform novel inputs (the *cross*) into output predictions (the *dashed circle*)

It performs multiple rounds of clustering; merging the closest clusters or divides the clusters at each round. Figure 1b shows a so-called dendrogram which can represent the result of hierarchical clustering. Any desired number of clusters can be obtained by "cutting" the dendrogram at the desired level.

1.3.2 Supervised Learning

Supervised learning methods require the value of the output variable for each training sample to be known. As a result, each training sample comes in the form of a pair of input and output values. The algorithm then trains a model that predicts the value of the output variables from the input variables using the defined features in the process. If the output variables are continuous valued then the predictive model is called a "regression function." For example, predicting the air temperature at a certain time of the year is a regression problem. If the output variables take a discrete set of values then the predictive model is called a "classifier." A typical classification problem is automated medical diagnosis for which a patient's data need to be classified as having a certain disease; or whether a given input is a miRNA. Figure 2 illustrates these two kinds of problems.

For supervised learning problems, it is possible to quantify the performance of the learned model by measuring the difference between the known output values and the predicted ones. However, the error for this performance evaluation must not be measured on the training data but on a separate test set. This ensures that the algorithm performance on novel data can be estimated correctly and gives an idea about the generalization of the learned model. The training and test procedures are discussed in more detail in Subheading 2.2.

Since it is much easier to gather unlabeled data, there are also semi-supervised learning methods that combine a small supervised training dataset with a larger unsupervised one. While training a predictive model, these algorithms can exploit both the supervised output values and the data distribution in the unlabeled data. However, these algorithms make additional assumptions to take advantage of the unlabeled data, which may or may not be suitable for the problem at hand [10].

As pointed out above, supervised learning with discrete results is called classification and the number of expected classes affects the choice of machine learning algorithm.

1.4 What Are Multi-class, Binary Classifications, and One-Class Density Estimation?

There are many machine learning problems with the objective of classifying the inputs into one of two categories. Often one category represents data points with a special property and the other category plays the role of a "background" class that includes everything else. Usually, the class representing the category of interest is termed the "positive" class and the background class is called the "negative" one. The miRNA identification problem is such an example with a category that represents miRNA sequences and another representing those that are not.

Classifiers that use machine learning techniques are often designed for such binary classification problems. This greatly simplifies their design and analysis. During training, these classifiers learn a decision boundary (*see* Fig. 2a) in the feature space that separates data points of the two classes as well as possible. Once the training is complete, they can predict the class of a new data point by comparing its location in the feature space with the learned decision boundary.

When there are more than two possible classes, the classification problem is said to be multi-class. Some machine learning techniques such as Decision Trees (DTs) can naturally handle the existence of multiple classes. Others such as Support Vector Machines (SVMs) can only handle binary problems in their original design. There are several ways to extend a binary classifier to handle multiple classes. A general approach is to turn a multi-class problem into multiple binary classification problems each in the form of "one class against all the others." When classifying test data, all binary classifiers are evaluated and the one with the highest confidence score wins (Fig. 3).

1.4.1 One Class Density Estimation

For some classification problems, it is not possible to collect reliable data belonging to one of the classes. Assume that you are working on diagnosing a rare type of cancer using some features obtained from the body cell. To do that, you develop a machine learning algorithm which would give "positive" as a result when the patient has cancer. To perform an effective training for your algorithm you would try to collect as many samples as possible.

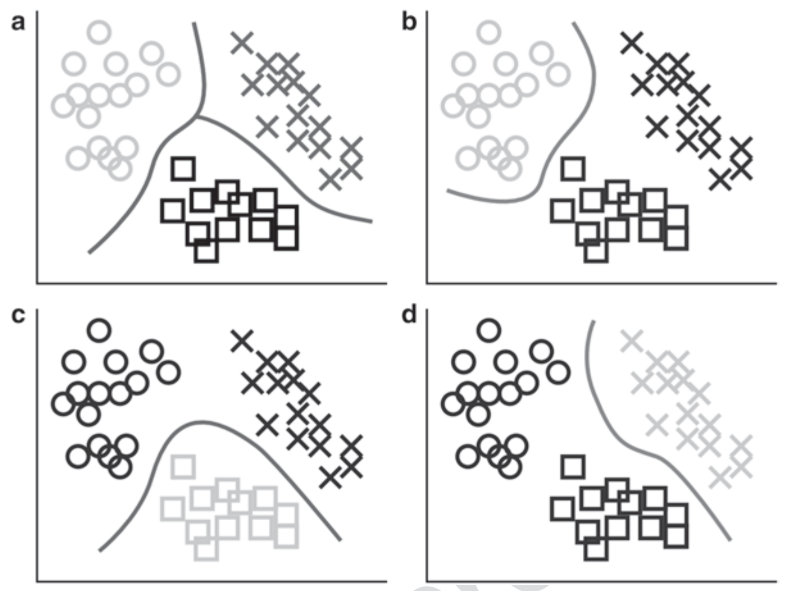

Fig. 3 Multi-class classification. (**a**) Some classification algorithms can handle multiple classes naturally to fit a complex decision boundary that separates all the classes from each other. (**b**–**d**) Some algorithms are designed to work with only two classes. In this case a multi-class problem can be decomposed into several binary classification problems with separate decision boundaries

However, in such a case, you probably would end up with many more "negative" samples than the "positive" ones. In other words, your dataset does not have a balanced amount of samples from different classes.

Estimation of one-class densities is also referred to as "anomaly detection" since the rare class indicates an anomaly within the huge amount of "normal" samples. Labeling a sample as "normal" is not as easy as labeling an "anomalous" one, because concealed anomalies may exist. A machine learning algorithm in such a case is trained to discover the common properties of the normal class to distinguish the anomalous samples from the rest.

For example, microRNAs can be identified experimentally but it is not currently feasible to clearly state that a given hairpin from a genome is not a miRNA (*see* Chapter 10) so the miRNA classification problem is also essentially a one class density estimation problem.

2 Design of Machine Learning Experiments

2.1 Model Complexity and Generalization

When given a dataset and a machine learning technique, we need to perform experiments to examine if the algorithm is working properly on the data and to gain insight on how to improve its performance.

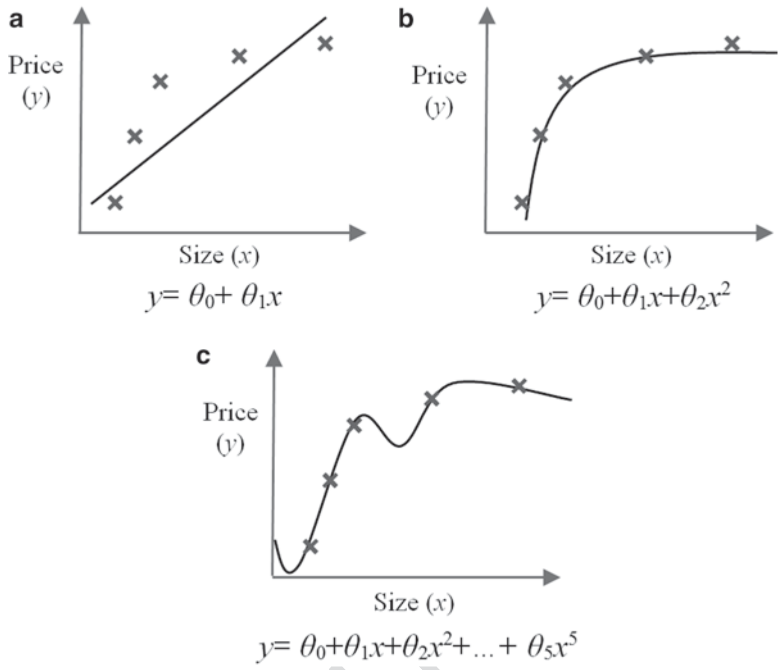

Fig. 4 Three hypotheses with different complexities for the house price prediction problem. (**a**) Underfit, hypothesis is a *line*. (**b**) Proper fit, hypothesis is a second degree *polynomial*. (**c**) Overfit, hypothesis is a fifth degree *polynomial*

We evaluate several hypotheses (models) to choose the best among them and this process is called model selection.

We consider the fact that the aim of a machine learning algorithm is to generate the correct output for sample data points outside the training set. The ability of the model to predict correct output for new samples after trained on the training set is defined as generalization. For best generalization, we should match the complexity of the model with the complexity of the function underlying the data [11].

To give a concrete example, let us consider a regression problem to predict the price of a house when its size is given. For the sake of simplicity, the size is the only feature in this example. In Fig. 4, crosses represent the data that we use to train our model and three models (hypotheses) with different complexities are shown with solid lines/curves.

If the hypothesis is not complex enough to model the samples, we have underfitting. Figure 4a shows the result of fitting a line to the data sampled from a high order polynomial. As we increase the complexity (the number of θ parameters) of our model, the training error decreases and we reach a better fit as shown in Fig. 4b. The error, here, can be defined as the sum of the squared distances between the data samples and the polynomial model (Fig. 5).

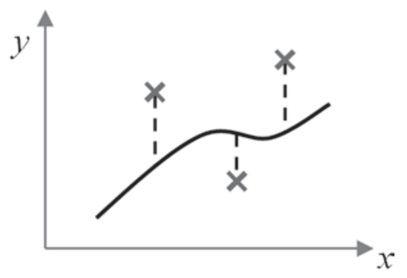

Fig. 5 The representation of the regression error. *Dashed lines* show the distances between the data samples and the polynomial. The error is the sum of the squares of these distances. Note that these are not the shortest distances to the model but the distances in *y* coordinate. This is correct since our estimate is the *y* coordinate (price) for a given *x* coordinate (size)

On the other hand, if the hypothesis is too complex and the data are not enough to constrain it, we may again end up with an improper model. For example, fitting a fifth order polynomial to some data sampled from a lower order polynomial (Fig. 4c). This is called overfitting. The hypothesis may fit the training set very well and we have quite low training error, however it fails to generalize to new samples (predicting prices of other houses).

For a given model complexity, the overfitting problem becomes less severe as the size of the dataset increases [12]. Ideally, when we have enough samples, a higher order polynomial becomes close to a lower order polynomial after training, so it resembles a proper fit. However, in most cases we cannot guarantee the sufficiency of data. Moreover, most of the time, the complexities of the model and data distribution cannot be visually compared like in this toy example. Therefore, we use other methods to evaluate the model as we will see in the following.

2.2 Using the Dataset for Evaluation

As explained previously, the generalization ability of a model can only be evaluated using samples outside the training set. To respect this requirement, we usually divide the training data into two parts. We use one part for training (e.g.: fitting a polynomial), and the remaining part is used to compute the error for that model to test its generalization.

The first part is called the training set and usually represents the bigger portion of the data (say 70 %). The second part is called the validation set. The model that gives the least error on the validation set is assumed to be the best.

In our regression example, to find a proper degree of the polynomial (this is the complexity of our model), we evaluate a number of candidate polynomials of different degrees (d), and find the coefficients (θ) using the training set for each of these polynomials (hypotheses/models). Let us denote the parameters

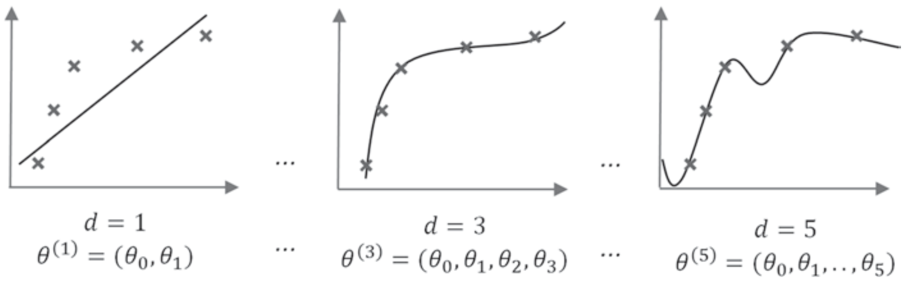

Fig. 6 Candidate models with different degrees for the polynomial regression problem. In this example, we evaluate *polynomials* with degree from *1* to *5*

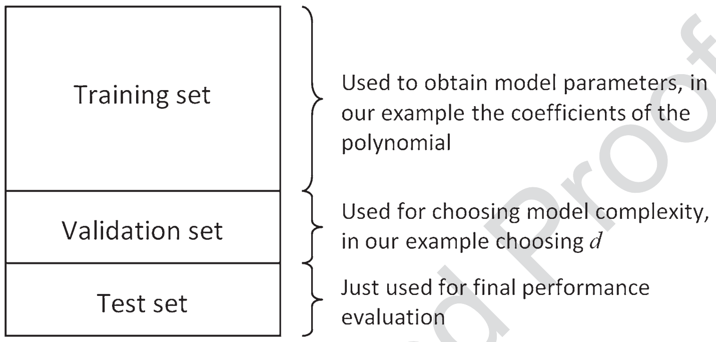
Training set
Used to obtain model parameters, in our example the coefficients of the polynomial
Validation set
Used for choosing model complexity, in our example choosing d
Test set
Just used for final performance evaluation

Fig. 7 Approximate distribution of the entire dataset for training, validation, and test sets

for the ith order polynomial with $\theta^{(i)}$. Figure 6 shows some of the candidate polynomials with different degrees. The next step is to compute the errors of these models on the validation set, and take the one that has the least validation error as the best polynomial. Let us say the model with $d = 3$ and $\theta^{(3)}$ generated the lowest error. In this case, we choose the third order polynomial as our model.

Although we have chosen the best performing model, we still do not know the real performance of the model since we have used the training set to estimate the parameters and the validation set to decide on the model complexity d (which is essentially a parameter as well). The test should contain new data. Therefore, while reporting the expected performance of our trained model, we use a third set, the test set, containing examples not used in training or validation.

In practice, most of the data is used for training and about one fifth for validation and again one fifth for testing (Fig. 7). Referring to our regression example, if we chose a third order polynomial as our model then, computing the error of $\theta^{(3)}$ on the test set gives us a fair error measure of the selected model.

2.2.1 Bias Versus Variance

In short, a model with high bias is an "underfit" one and a model with high variance is an "overfit" one. To better visualize the

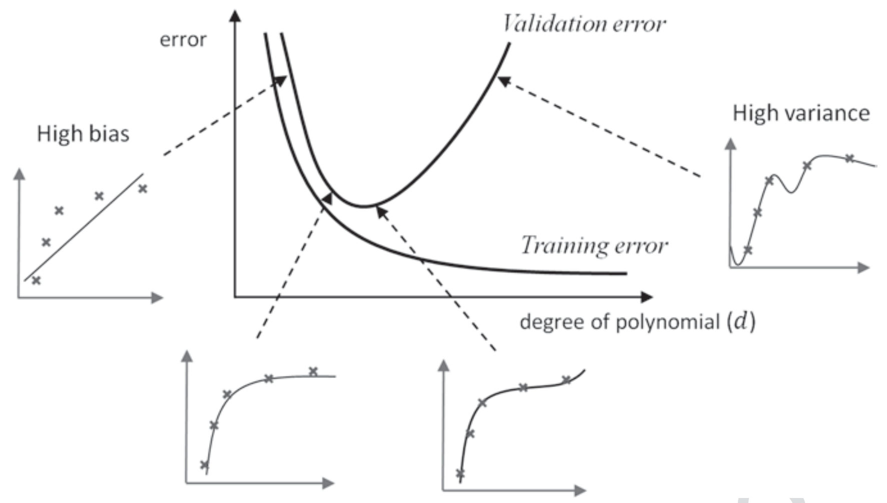

Fig. 8 Training and validation error curves for increasing model complexity. Simple models (small *d* in our example) have the risk of high bias, where the error is high both in training and validation sets. Complex models have the risk of high variance (fluctuations on the polynomial), where the training error is low since the model fits better to the training data, but the validation error is high

relationship between training and validation (generalization) errors, let us examine Fig. 8. Starting from a less complex model, as the complexity of the model increases, the training error decreases since the model fits better to the data. When we consider the error on the validation set, it initially decreases as it possibly fits better to the validation set as well. But then, as we move further to more complex models it increases again. High variance (fluctuations on the polynomial) in the complex models causes a poor fit (overfitting) for the novel data in the validation set. The bottom of the bowl on the validation error curve is the point where the generalization error is the minimum.

2.3 Dimensionality Reduction

The polynomial regression example in the previous section had just one input variable (x). For practical applications of machine learning, on the other hand, we have to deal with spaces of high dimensionality consisting of many input features [12].

This *multivariate* structure of the data generally causes problems for computation or visualization, and therefore this situation is referred to as the *curse of dimensionality* [13].

Dimensionality reduction is one of the major tasks in the analysis of multidimensional data, which is the step that we decrease the number of dimensions/features. The main motivations for performing dimensionality reduction are the following:

- Computation is faster with fewer features. Genomic data can be given as an example. If all the genes in a genome are considered as features, then we would have several thousand features.

- If we find out that one or more features are not discriminative, eliminating them saves time, effort, and increases prediction accuracy.
- Two or three dimensional projections help us (1) to visually represent our data to gain insight, (2) to screen data for obvious outliers and (3) to observe cluster tendencies when using unsupervised learning.

Since we wish to minimize the information loss to be caused by dimensionality reduction, we try to eliminate the least distinctive/informative features. For instance, a feature that is highly correlated with another one can be considered as redundant.

There are two main methods for reducing dimensionality: *feature selection* and *feature extraction*. In feature selection, we find k of the d dimensions (features) that give us the most information and we eliminate the other dimensions. Feature selection methods can be roughly divided into two categories: filters and wrappers. Filters extract feature relevancies via various scoring techniques without using a learning model and select a subset of features using these scores. Filters are computationally simple and fast. Some of the popular filter approaches are mutual information [14], chi-square [15], and information gain [16].

Wrappers, on the other hand, conduct a search for good features using the learning algorithm itself as part of the function [17]. Wrapper techniques provide interaction between feature subset and learning model, but are computationally expensive when compared to filters. The two approaches here are forward selection and backward elimination. Forward selection refers to a search that begins an empty set of features. At each step, for all features, we add a feature in the feature subset and we train our model on the training set and calculate error on the validation set. Then, we choose the feature which causes the greatest decrease in error and permanently put it in the feature subset. This continues until no further improvement occurs. In backward elimination, we start with the full set of features and we remove one feature at a time. We eliminate the one, removing of which causes the least error increase [11].

In feature extraction, we transform the original d dimensions to a new set of dimensions and select k of these new dimensions. A popular technique to do the latter is principal component analysis (PCA), where we analyze the data and come up with the most informative components (dimensions).

Normally the reduction is performed for much higher dimensionalities but to visualize the process let us consider an example where three dimensional data is reduced to two dimensions. As mentioned above, the main idea is to capture the most informative dimensions. Figure 9a shows a set of data points in three dimensions (features). The data points roughly constitute a plane and

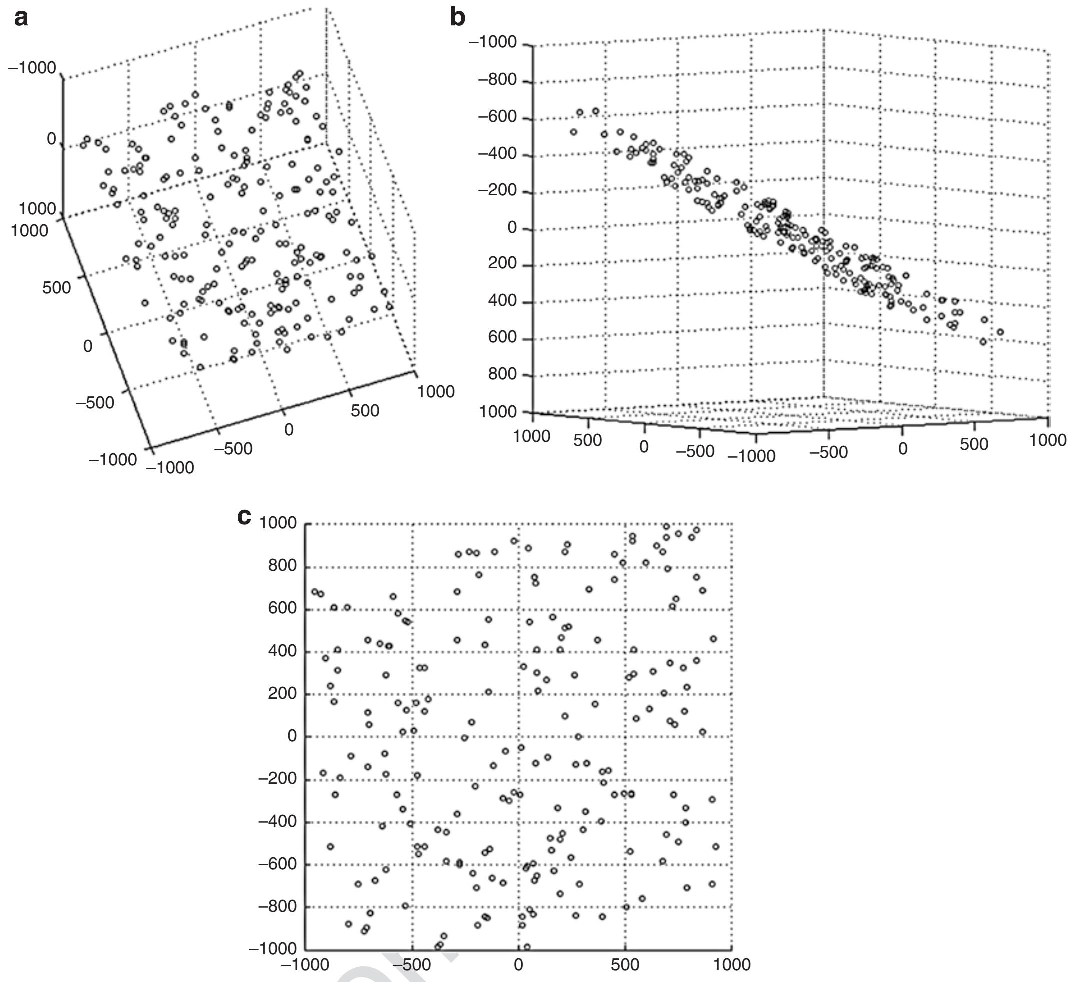

Fig. 9 3D to 2D dimensionality reduction example. (**a**) A set of data points in three dimensions (features). (**b**) The same data points viewed from another angle to emphasize that the points roughly constitute a plane. (**c**) 2D data points after the redundant dimension is removed with principal component analysis (PCA)

this fact is shown in Fig. 9b where the same data points are viewed from another angle. This means that the distinction between the data points can be represented in 2D and a third dimension does not add much information to this distinction because all the data points have approximately the same value on that dimension. The reader should note that selecting two features of the original data does not accomplish the desired reduction because the plane that the data is on, i.e., the redundant dimension is not one of the original *x*, *y* and *z* dimensions (axes) but a combination of these. PCA helps us to transform our data to a new set of three dimensions and in that space we can omit the redundant dimension to obtain a new 2D dataset (Fig. 9c).

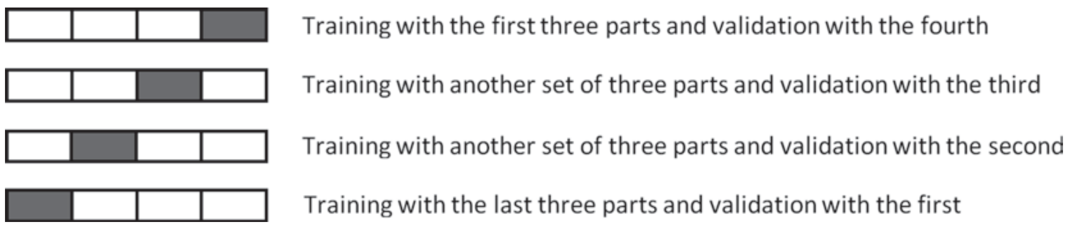
Training with the first three parts and validation with the fourth
Training with another set of three parts and validation with the third
Training with another set of three parts and validation with the second
Training with the last three parts and validation with the first

Fig. 10 Illustration of the partition of the dataset for K-fold Cross-Validation with $K=4$

Table 1
The so-called *confusion matrix* shows the four possible situations that can occur according to the truth values of the actual and the predicted class

	Predicted class label	Actual positive	Actual negative
Actual class label		+1	−1
Predicted positive	+1	True positive	False positive
Predicted negative	−1	False negative	True negative

2.4 Randomization and Cross-validation

Randomization is required to ensure that the result of the learning process is independent of the selection of training data [11]. This is a typical problem in real-world experiments. For instance, a part of some measurement data may have been taken when the device was in a certain state (slightly different tuning etc.).

As mentioned earlier, we need to divide our training data to obtain the training and validation sets (after sparing some part as the test set). We would like to ensure the random sampling of these sets from the data we have. If the dataset is large enough, we can randomly divide it into K parts, and then randomly divide each part into two as the training and validation sets. This means we repeat the experiment K times. Unfortunately, datasets are rarely large enough to do this. So randomization is accomplished by repeated use of the same data split differently; this is called *cross-validation*.

2.4.1 K-Fold Cross-validation

Illustrated in Fig. 10 for $K=4$, the dataset is divided randomly into K equal-sized parts. Then, $K-1$ parts are used to train a set of models and the remaining part is used as a validation set to evaluate those models. This procedure is repeated for all K possible choices [12]. As K increases, the percentage of the training set increases and we get more robust estimators, but the validation set becomes smaller. Therefore, a K value that ensures randomization is a good choice and larger values should be avoided.

2.5 Robust Performance Metrics

2.5.1 Precision–Recall

Let us first introduce the so-called *confusion matrix* (Table 1). As shown at the bottom-right portion of the table, there are four possible cases. For a positive example, if the prediction is also positive, it is a true positive; if our prediction is negative for an actually

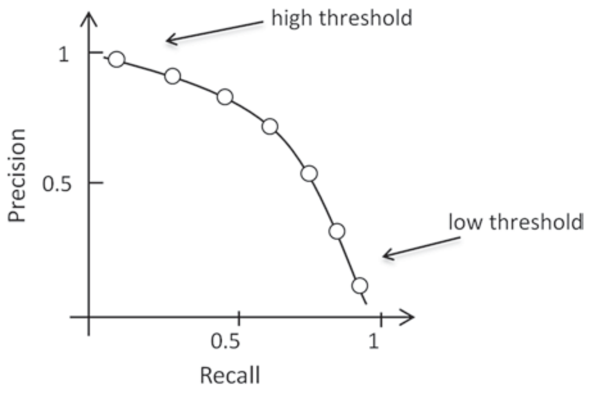

Fig. 11 Precision–Recall curve representation. *Circles* denote results of trials with different thresholds

positive example, it is a false negative. For a negative example, if the prediction is also negative, we have a true negative, and we have a false positive if we predict a negative example as positive.

For the cancer diagnosis example, where "positive" denotes having cancer, a false positive is wrongly making a cancer diagnosis for a healthy patient and a false negative is missing a patient actually having cancer. Precision tells us what fraction actually has cancer of all patients where we predicted "positive." Recall tells us what fraction we correctly detected as having cancer of all patients that actually have cancer.

With the definitions above, we can write

$$Precision = \frac{\#\,True\ Positives}{\#\,Predicted\ Positives} \qquad Recall = \frac{\#\,True\ Positives}{\#\,Actual\ Positives}.$$

The values in the confusion matrix, as well as precision and recall, change as we modify our detection algorithm's threshold, which defines at which probability a sample is labeled as "positive." Different threshold probability values can be chosen for different tasks or for preferred behavior regarding the same task.

Precision–Recall (PR) curves are generally used in the community for performance evaluation. A typical PR curve is shown in Fig. 11, where the circles denote results of trials with different thresholds. Changing the threshold of the algorithm moves us on the curve.

With a low threshold, we tend to predict "positive" for the data samples and our recall gets closer to 1 since we miss few actual positives (patients with cancer), however our precision is quite low since there are many false positives. On the other hand, if we choose a high threshold and we only predict "positive" for the most probable samples, our precision is high since the number of true and predicted positives become close to each other. But this time, recall is low. Ideally we want to keep both precision and recall high, this corresponds to the area under the curve.

Table 2
The F_1 scores of three different algorithms used for a detection problem

	Precision (P)	Recall (R)	F_1 Score
Algorithm 1	0.5	0.4	0.444
Algorithm 2	0.7	0.1	0.175
Algorithm 3	0.02	1.0	0.0392

The algorithm with precision and recall values close to each other has a higher F_1 score

A measure that was proposed to compare different precision–recall pairs (different thresholds) is the F score:

$$F_\beta = \left(1+\beta^2\right)\frac{Precision \times Recall}{\left(\beta^2 \times Precision\right) + Recall}$$

Most commonly, F_1 score is used (where $\beta = 1$). That is the F score corresponding to the harmonic mean of precision and recall:

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall}.$$

To concretize its effectiveness, the F_1 score for different algorithms is tabulated in Table 2, where the algorithm having precision and recall values close to each other has a distinctively higher F_1 score.

2.5.2 Specificity and Sensitivity

From another perspective but with the same aim, there are the two measures of *sensitivity* and *specificity*. Sensitivity is the same as recall. Specificity measures the proportion of negatives which are correctly identified, i.e., true negatives divided by the total number of negatives. One can also draw sensitivity vs. specificity curve using different thresholds.

2.5.3 ROC Curve

Receiver Operating Characteristics (ROC) curve is another graphical plot to illustrate the performance comparison of different methods. It is created by plotting, at various thresholds, the fraction of true positives out of the positives (TPR = true positive rate) vs. the fraction of false positives out of the negatives (FPR = false positive rate). TPR is the same as recall and sensitivity; FPR is 1 – specificity.

3 Supervised Machine Learning Methods

3.1 Probabilistic Classification Methods

A popular classification approach is to model the relationship between features and the class of the data points using probabilities. Let us denote the features as $x_i (i = 1, \ldots, M)$ and the feature vector for a data point then becomes:

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_M].$$

We can write, for a data point, the probability of belonging to each of the N classes $(c_1, c_2, \ldots, c_N)$ as

$$P(C = c_1 \mid \boldsymbol{x}), P(C = c_2 \mid \boldsymbol{x}), \ldots, P(C = c_N \mid \boldsymbol{x}).$$

Given a new data point, it is classified as the class with the maximum probability,

$$c^* = \arg\max_{c_j} P\left(C = c_j \mid \boldsymbol{x}\right) \quad \textit{and} \quad j = 1, \ldots, N.$$

The probability for each class can be computed using the Bayes' rule

$$P(C \mid \boldsymbol{x}) = \frac{P(\boldsymbol{x} \mid C) P(C)}{P(\boldsymbol{x})} = \frac{P(\boldsymbol{x} \mid C) P(C)}{\sum_{c_i \in C} P(\boldsymbol{x} \mid C = c_i) P(C = c_i)}.$$

As a toy example for the miRNA identification problem, assume that we have received a collection of nucleotide sequences. Of these 1,000 are miRNAs and 1,500 are not miRNAs that we call negative samples. The nucleotide motif "AGCACU" exists in 900 of the miRNA sequences and only 50 of the negative samples. We will have a single feature x that is equal to 1 if the candidate sequence contains "AGCACU" and 0 otherwise. We can compute the relevant probabilities as follows:

$$P(x = 0 \mid C = miRNA) = \frac{1{,}000 - 900}{1{,}000} = 0.1, \quad P(x = 1 \mid C = miRNA) = \frac{900}{1{,}000} = 0.9,$$

$$P(x = 0 \mid C = negative) = \frac{1{,}500 - 50}{1{,}500} = 0.967, \quad P(x = 1 \mid C = negative) = \frac{50}{1{,}500} = 0.033,$$

$$P(C = miRNA) = \frac{1{,}000}{2{,}500} = 0.40, \quad P(C = negative) = \frac{1{,}500}{2{,}500} = 0.60,$$

$$P(x = 0) = \sum_{c \in \left\{\substack{miRNA, \\ negative}\right\}} P(x = 0 \mid C = c) P(C = c) = 0.1 \times 0.4 + 0.967 \times 0.6 = 0.62,$$

$$P(x = 1) = \sum_{c \in \left\{\substack{miRNA, \\ negative}\right\}} P(x = 1 \mid C = c) P(C = c) = 0.9 \times 0.4 + 0.033 \times 0.60 = 0.38.$$

Given these probabilities, it is possible to compute the probability of each class $P(C|x)$ for a novel candidate sequence as follows:

$$P(C = miRNA \mid x = 0) = \frac{P(x = 0 \mid C = miRNA)P(C = miRNA)}{P(x = 0)} = \frac{0.1 \times 0.4}{0.62} = 0.06,$$

$$P(C = negative \mid x = 0) = \frac{P(x = 0 \mid C = negative)P(C = negative)}{P(x = 0)} = \frac{0.967 \times 0.6}{0.62} = 0.94,$$

$$P(C = miRNA \mid x = 1) = \frac{P(x = 1 \mid C = miRNA)P(C = miRNA)}{P(x = 1)} = \frac{0.9 \times 0.4}{0.38} = 0.95.$$

$$P(C = negative \mid x = 1) = \frac{P(x = 1 \mid C = negative)P(C = negative)}{P(x = 1)} = \frac{0.033 \times 0.6}{0.38} = 0.05.$$

With the calculated probabilities, if a given nucleotide sequence contains the word "AGCACU," it will be classified as miRNA because $P(C = \text{miRNA}|x = 1) > P(C = \text{negative}|x = 1)$.

Since we are taking the maximum over the classes, the $P(x)$ term in the denominator does not affect the predicted class and we can simplify the classification rule. It can be directly written as:

$$c^* = \arg\max_{c_j} P(\boldsymbol{x} \mid C = c_j)P(C = c_j) \quad and \quad j = 1, \ldots, N.$$

During training $P(x|C)$ and $P(C)$ are estimated from the training data points for each class.

When the features are continuous, there are many ways to model the probability $P(x|C)$. Parametric models assume a particular form for the probability that is controlled by several parameters as shown in Fig. 12a. A commonly used model is the Gaussian distribution $\mathcal{N}(\mu, \Sigma)$ controlled by its mean $\boldsymbol{\mu}$ and covariance matrix Σ. The control parameters can be estimated from the training data. If a single Gaussian distribution is too simple to model the feature distribution, a mixture of Gaussians can be estimated by the Expectation Maximization (EM) technique [18]. The mixture is formed by a weighted sum of Gaussian distributions as $\sum_{i=1}^{K} \alpha_k \mathcal{N}(\mu_k, \Sigma_k)$, where α_k are the mixing coefficients that weigh the contribution from each Gaussian component. Unlike a single Gaussian, a mixture model can have multiple modes and therefore it is more general.

Another alternative is to model the probability $P(x|C)$ with nonparametric methods that do not have control parameters. A histogram over the feature space can estimate the data density in various regions of the partitioned feature space. As illustrated in Fig. 12b, by weighting the contribution from each sample over a range of histogram bins, the computed histogram can be smoothed to reduce errors in the sparsely populated regions of the feature space.

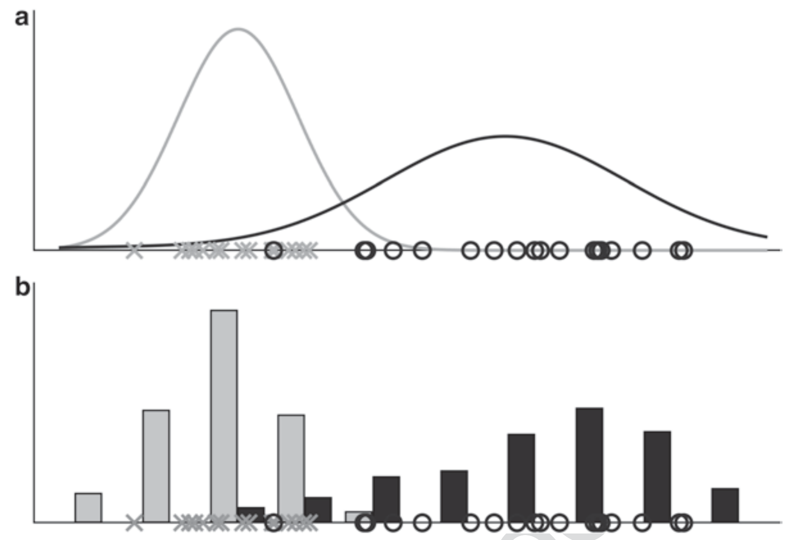

Fig. 12 Modeling $P(x|C)$ from the training data (the *crosses* and the *circles*). (**a**) Parametric models (such as the Gaussian curves depicted above) are functions that are controlled by a set of control parameters. The values of these parameters are estimated from the training data points. (**b**) Nonparametric models are histograms that are computed by separating the input feature space into distinct bins. Each training data point contributes to several bins around itself according to a local weighting function

3.1.1 Naïve Bayes

One common way to simplify the modeling of the joint feature probability $P(x|C)$ is to assume independence between features. In exact form, we can write

$$P(\boldsymbol{x}\,|\,C) = P(x_1\,|\,x_2,\ldots,x_M,C)\cdot P(x_2\,|\,x_3,\ldots,x_M,C),\ldots,P(x_M\,|\,C).$$

If we assume independence between features $x_1,\ldots,x_M$ then the joint probability reduces to

$$P(\boldsymbol{x}\,|\,C) \cong P(x_1\,|\,C)\cdot P(x_2\,|\,C),\ldots,P(x_M\,|\,C) = \prod_{j=1}^{M} P(x_j\,|\,C).$$

This simplification is called the naïve Bayes assumption and a classifier using such a model is called a naïve Bayes classifier. Although the independence assumption is quite strong and does not hold in general, naïve Bayes classifiers perform remarkably well for a wide range of problems. Moreover, the training can be very efficiently performed since each feature probability can be computed independently.

The independence assumption can improve training accuracy by reducing the number of model parameters that needs to be estimated. Consider a learning problem with F features that are real numbers. If the full joint probability is modeled as a multidimensional Gaussian, then $(F^2+3F)/2$ parameters are required to represent the mean and the covariance matrix. With the naïve Bayes assumption, this reduces to $2F$ parameters since each feature probability can be modeled by a one-dimensional Gaussian distribution. Hence, the model parameters can be more reliably estimated with limited training data.

Still, it is necessary to exercise caution when estimating the feature probabilities with a small number of data points. Since the feature probabilities are multiplied, a single zero probability for a feature can overcome strong evidence from several other features. As a precaution, virtual training samples that are uniformly distributed across the feature space and the classes can be used to make sure that no feature probability is ever exactly zero.

Naïve Bayes classifiers are also used in the miRNA identification task [6]. For a chosen dictionary of M nucleotide words that are likely to be present in miRNAs, a binary feature w_i represents whether word i exists in the nucleotide sequence, i.e., $w_i = 1$ if the word i is part of the sequence, $w_i = 0$ otherwise. For each such feature, the probabilities $P(w_i = 0|C = \text{miRNA})$, $P(w_i = 1|C = \text{miRNA})$, $P(w_i = 0|C = \text{negative})$, and $P(w_i = 1|C = \text{negative})$ are all estimated from a large sample of miRNA positive and negative examples. Given the words in a novel candidate sequence, a feature vector $w = [w_1, w_2, \ldots, w_M]$ can be extracted. The joint probabilities for both the "miRNA" and "negative" classes can be calculated as

$$P(\boldsymbol{w} \mid C = miRNA) = \prod_{i=1}^{M} P(w_i \mid C = miRNA) \text{ and}$$
$$P(\boldsymbol{w} \mid C = negative) = \prod_{i=1}^{M} P(w_i \mid C = negative).$$

The larger of these probabilities determines the label of the candidate nucleotide sequence. Of course, in an actual miRNA system, more complex features determined by experts are included in the statistical model (*see* Chapters 9, 10, and 12 and [6]).

3.2 Linear Discriminant Functions

The probabilistic approach outlined above first models the class conditional probabilities $P(x|C)$ then bases its classification estimates on the class with the maximum probability. In the case of a binary classification problem this amounts to first computing the ratio

$$\gamma = \frac{P(\boldsymbol{x} \mid C = c_1)}{P(\boldsymbol{x} \mid C = c_2)}.$$

And then, choosing c_1 if $\gamma > 1$, and c_2 otherwise. The region of the feature space where $\gamma = 1$ forms the decision boundary.

Depending on the form of $P(\boldsymbol{x}|C)$, the decision boundary can be very complex or just a simple hyper-plane. A hyper-plane in a d dimensional space is a $d - 1$ dimensional flat region with the equation $a_1x_1 + a_2x_2 + \cdots + a_dx_d = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{c}$. A hyper-plane in two dimensions is a line and a hyper-plane in three dimensions is a plane. If $P(\boldsymbol{x}|C)$ is a Gaussian distribution with the same covariance matrix Σ for both classes then the decision boundary is exactly a hyper-plane [12]. This linear decision boundary can be written as

$$y = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} + c, \text{ with } \boldsymbol{a} = \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \text{ and } c = -\tfrac{1}{2}\mu_1^{\mathrm{T}}\Sigma^{-1}\mu_1 + \tfrac{1}{2}\mu_2^{\mathrm{T}}\Sigma^{-1}\mu_2 .$$

3.2.1 Fisher's Linear Discriminant

In case the form of $P(x|C)$ is not Gaussian, we can still exploit the formulation above to find a linear separation boundary that distinguishes between the classes in an optimal way. Fisher's linear discriminant method achieves this by projecting the input data points onto a hyper-plane such that their data distributions are as far apart from each other as possible. The projection that maximizes the separation between distributions is given as $y = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}$ with $\boldsymbol{a} = (\Sigma_1 + \Sigma_2)^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$. Note that the data distributions do not need to have the same covariance matrix. If they do, then the Fisher's linear discriminant is equivalent to the Gaussian formulation above. Once the data is projected into one dimension y, a threshold is computed so that the prediction error is minimized on the training data. Both the Fisher's linear discriminant and the Gaussian formulation above can be easily generalized for multi-class classification problems.

3.3 Support Vector Machines

For two linearly separable classes with unknown distributions, there is no unique decision boundary. As long as the selected hyper-plane separates the training samples of the two classes, it can be chosen as the decision boundary. However, it is prudent to select a decision boundary that does not pass too close to the training samples to account for the limited training data and errors in data collection.

Support Vector Machines (SVMs) rely on maximizing the margin of error to select the best hyper-plane. The margin is determined by a set of hyper-planes parallel to the decision boundary on the positive and negative sides of the discriminant function each at the same distance to the boundary as depicted by Fig. 13. When the margin is maximized, the training data points that are closest to the decision boundary are on the margin hyper-planes. These training data points are called the "support vectors."

Since the margins and the decision boundary are only determined by the support vectors, the SVM classification rule can be written as a function of these points. If we have a two class problem and we label the classes with $\{-1, +1\}$, the ith training data point can be written as $X^i = \{\boldsymbol{x}^i, y^i\}$, where x^i is the feature vector and $y^i \in \{-1, +1\}$ is the supervised class label. The support vector decision boundary corresponds to the hyper-plane equation $\gamma = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + c$ and the weight vector is given by $\boldsymbol{w} = \sum_i \alpha^i y^i \boldsymbol{x}^i$ with $\alpha^i = 0$ for the training samples that are not support vectors.

In practice, it is usually not possible to completely separate all training samples by a hyper-plane and some training samples can end up on the wrong side of the decision boundary or within the margin.

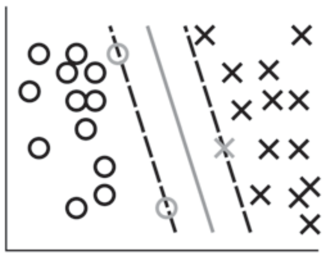

Fig. 13 The decision boundary of an SVM (the *solid line*) is determined by the support vectors (the *lighter circles* and *crosses*) that lie on the margin hyperplanes (the *dashed lines*). The support vectors and the decision boundary are selected to maximize the margin

The SVM formulation is softened to allow such data points. A complexity parameter, usually denoted by *C*, controls how much these points are penalized. A higher penalty means a more complex model with potentially more support vectors. The value of this parameter should be set using a validation dataset as discussed before.

3.3.1 Kernels

Most real-world problems involve data distributions that are not linearly separable. One possible approach around this problem is to perform a nonlinear transformation of the input features into a higher dimensional space as $\boldsymbol{x} \to \Phi(\boldsymbol{x})$. This transformation can make the class data distributions linearly separable. Then the linear SVM can be trained in the new space since the linear decision boundary in this space corresponds to a nonlinear curve in the original feature space as illustrated by Fig. 14.

The SVM formulation is particularly suitable to this kind of transformation since the feature vectors always appear in the form of dot products of two data points as $K\left(\boldsymbol{x},\hat{\boldsymbol{x}}\right) = \boldsymbol{x}^{\mathrm{T}}\hat{\boldsymbol{x}}$. This dot product is called a "kernel" and it is a measure of similarity between the two data points $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$. So even if the transformation $\Phi(\boldsymbol{x})$ is complex and very high dimensional, after the transformation the dot product $K\left(\boldsymbol{x},\hat{\boldsymbol{x}}\right) = \Phi\left(\boldsymbol{x}\right)^{\mathrm{T}}\Phi\left(\hat{\boldsymbol{x}}\right)$ may have a simple form. Indeed, the form of $K\left(\boldsymbol{x},\hat{\boldsymbol{x}}\right)$ can be set directly without ever computing the nonlinear mapping $\Phi(\boldsymbol{x})$, provided that the kernel form satisfies some mathematical constraints [19].

For example, the Gaussian kernel can be written as $K\left(\boldsymbol{x},\hat{\boldsymbol{x}}\right) = e^{-\frac{\|\boldsymbol{x}-\hat{\boldsymbol{x}}\|^2}{2\sigma^2}}$. Other kernels that are commonly used are the polynomial, the sigmoid and the radial basis function (RBF) kernels. Since each kernel corresponds to a different nonlinear transformation of the input space, it is not possible to know which one will be the best choice for a particular machine learning problem. Also kernels usually have parameters that define the shape and the complexity of the nonlinear transformation such as $\boldsymbol{\sigma}$ for the Gaussian kernel. Both the type and the parameters of the kernel should be selected using a validation set as described before in Subheading 2.3.

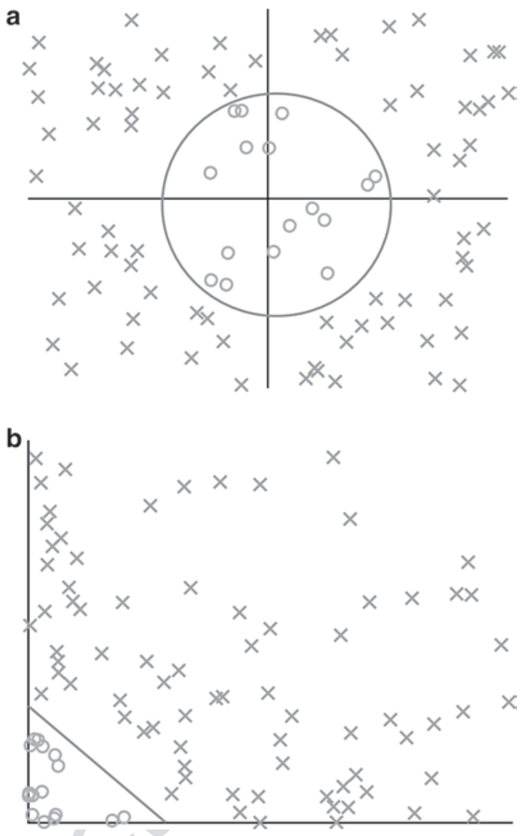

Fig. 14 Nonlinear transformations can linearize the decision boundary between classes. (**a**) In the original feature space, the classes are not linearly separable. (**b**) Each feature value is squared to perform a nonlinear mapping of the data points. In the transformed feature space, the decision boundary is a *simple line*

4 Conclusion

Machine learning techniques provide exciting new ways to exploit the available computational power and data in a variety of scientific domains. They can analyze huge amounts of data in a relatively short time that is not possible by manual labor. This provides opportunities for scientists to develop new experimental procedures and to channel their efforts on the most promising questions of their problem domain.

However, automated solutions are not a replacement for good scientific judgment. Like any other tool, a machine learning technique needs to be utilized in a careful manner to make the most out of its use. It is better to start with the simpler methods to judge problem difficulty and to gain more insight about algorithm behavior. It is also important to try a few different algorithms and compare their performances. As discussed in Subheading 2, experiments to test the generalization ability of a model should be designed properly considering the important aspects such as choosing the training samples and randomization.

References

1. RapidMiner -- Data mining, ETL, OLAP, BI, http://sourceforge.net/projects/rapidminer/
2. scikit-learn: machine learning in Python, http://scikit-learn.org/stable/
3. The SHOGUN machine learning toolbox, http://www.shogun-toolbox.org/
4. Weka 3 - Data mining with open source machine learning software in Java, http://www.cs.waikato.ac.nz/ml/weka/
5. Guyon I, Elisseeff A (2003) An introduction to variable and feature selection. J Mach Learn Res 3:1157–1182
6. Yousef M, Nebozhyn M, Shatkay H et al (2006) Combining multi-species genomic data for microRNA identification using a Naïve Bayes classifier. Bioinformatics 22: 1325–1334
7. MacQueen J (1967) Some methods for classification and analysis of multivariate observations. Proceedings of the fifth Berkeley symposium on mathematical statistics and probability. University of California Press, Los Angeles, CA, pp 281–297
8. Hastie T, Tibshirani R, Friedman JH (2003) The elements of statistical learning. Springer, New York, NY
9. Ng AY, Jordan MI, Weiss Y et al (2002) On spectral clustering: analysis and an algorithm. Adv Neural Inform Process Syst 2:849–856
10. Chapelle O, Schölkopf B, Zien A (eds) (2010) Semi-supervised learning. The MIT Press, Cambridge, MA
11. Alpaydın E (2010) Introduction to machine learning. The MIT Press, Cambridge, MA
12. Bishop C (2006) Pattern recognition and machine learning. Springer, New York, NY
13. Bellman RE (1961) Adaptive control processes: a guided tour. Princeton University Press, Princeton, NJ
14. Liu H, Sun J, Liu L et al (2009) Feature selection with dynamic mutual information. Pattern Recogn 42:1330–1339
15. Chen Y-T, Chen MC (2011) Using chi-square statistics to measure similarities for text categorization. Expert Syst Appl 38:3085–3090
16. Lee C, Lee GG (2006) Information gain and divergence-based feature selection for machine learning-based text categorization. Inform Process Manag 42:155–165
17. Kohavi R, John GH (1997) Wrappers for feature subset selection. Artif Intell 97:273–324
18. Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J Roy Stat Soc B 39:1–38
19. Schlkopf B, Smola AJ (2001) Learning with kernels: support vector machines, regularization, optimization, and beyond. The MIT Press, Cambridge, MA