MACHINE LEARNING

UNIT-2

Beyond Binary Classification

Topics

- **Handling more than Two Classes**
- **Regression**
- **Unsupervised and Descriptive Learning**

Multi-Class Classification

- Classification tasks with more than two classes are very common.

- If we have k classes, performance of a classifier can be assessed using a k-by-k contingency table.

- Assessing performance is easy if we are interested in the classifier's accuracy, which is still the sum of the descending diagonal of the contingency table, divided by the number of test instances.

Example for K x K Contingency Table

Consider the following three-class confusion matrix (plus marginals):

	Predicted			
	15	**2**	**3**	20
Actual	**7**	**15**	**8**	30
	2	**3**	**45**	50
	24	20	56	100

- The accuracy of this classifier is $(15+15+45)/100 = 0.75$.
- We can calculate per-class precision and recall: for the first class this is $15/24 = 0.63$ and $15/20 = 0.75$ respectively, for the second class $15/20 = 0.75$ and $15/30 = 0.50$, and for the third class $45/56 = 0.80$ and $45/50 = 0.90$.

Example for K x K Contingency Table

- We could average these numbers to obtain single precision and recall numbers for the whole classifier.
  - For instance, the average precision is (0.63+0.75+0.80)/3 = 0.72.
- we could take a weighted average taking the proportion of each class into account.
  - For instance, the weighted average precision is 0.20*0.63+0.30*0.75+0.50*0.80 = 0.75.
- Another possibility is to perform a more detailed analysis by looking at precision and recall numbers for each pair of classes:
  - when distinguishing the first class from the third precision is 15/17 = 0.88 and recall is 15/18 = 0.83,
  - while distinguishing the third class from the first these numbers are 45/48 = 0.94 and 45/47 = 0.96

		Predicted			
		15	2	3	20
Actual		7	15	8	30
		2	3	45	50
		24	20	56	100

Construction of multi-class classifiers

- Imagine now that we want to construct a multi-class classifier, but we only have the ability to train two-class models
- There are several ways to combine several of them into a single k-class classifier.
- ***one-versus-rest* scheme:**
  - train k binary classifiers, the first of which separates class $C1$ from $C2, \ldots, Cn$, the second of which separates $C2$ from all other classes, and so on.
  - When training the i -th classifier we treat all instances of class Ci as positive examples, and the remaining instances as negative examples.
  - we learn k models, the i -th one separating Ci from $Ci+1, \ldots, Cn$ with $1 \leq i < n$.
- ***one-versus-one Scheme:***
  - we train $k(k-1)/2$ binary classifiers, one for each pair of different classes.

Construction of multi-class classifiers

- A convenient way to describe all these schemes to decompose a *k-class* task into *'l' binary classification tasks is by means of a* ***output code matrix.***
- This is a ***k-by-l matrix*** *whose entries are +1, 0 or –1.*
- *Ex:*

$$\begin{pmatrix} +1 & +1 & 0 \\ -1 & 0 & +1 \\ 0 & -1 & -1 \end{pmatrix} \qquad \begin{pmatrix} +1 & -1 & +1 & -1 & 0 & 0 \\ -1 & +1 & 0 & 0 & +1 & -1 \\ 0 & 0 & -1 & +1 & -1 & +1 \end{pmatrix} \qquad \begin{pmatrix} +1 & -1 & -1 \\ -1 & +1 & -1 \\ -1 & -1 & +1 \end{pmatrix}$$

Construction of multi-class classifiers

one-versus-one schemes can be described by means of **output code** matrix:

$$\begin{pmatrix} +1 & +1 & 0 \\ -1 & 0 & +1 \\ 0 & -1 & -1 \end{pmatrix}$$

where each column describes a binary classification task, using the class in the row with $+1$ entry as $\oplus$ and the class in the row with -1 entry as $\ominus$.

$$\begin{pmatrix} +1 & -1 & +1 & -1 & 0 & 0 \\ -1 & +1 & 0 & 0 & +1 & -1 \\ 0 & 0 & -1 & +1 & -1 & +1 \end{pmatrix}$$

The asymmetric scheme learns three more classifiers with the roles of positives and negatives swapped.

Prediction in Multi-Class Classifier

- In order to decide the class for a new test instance for the scheme one-versus-rest:
  - Collect predictions from all binary classifiers which can again be +1 for positive, −1 for negative
  - Together, these predictions form a 'word' that can be looked up in the code matrix, a process also known as *decoding*.
  - Suppose the word is −1 +1 −1 then the decision should be class common in -1 prediction.

Prediction in Multi-Class Classifier

- **If the scheme is symmetric one-versus-one:**
  - Check the nearest code word.
- we define the distance between a word w and a code word c as

$$d(w,c) = \sum_i (1 - w_i c_i)/2,$$

  where i ranges over the columns in the code matrix.
- That is, bits where the two words agree do not contribute to the distance;
- bits where one word has +1 and the other −1 contributes 1;
- and if one of the bits is 0 the contribution is 1/2,
- The predicted class for word w is then $\text{argmin}_j\, d(w, c_j)$, where c_j is the j -th row of the code matrix.

Prediction in Multi-Class Classifier

- suppose the word is (0, +1, 0), and the scheme is symmetric one-versus-one:

$$d(w,c) = \sum_i (1 - w_i c_i)/2,$$

$$\begin{pmatrix} +1 & +1 & 0 \\ -1 & 0 & +1 \\ 0 & -1 & -1 \end{pmatrix}$$

- So, if w = (0, +1, 0) then
- d(w,c1) = 1 and
- d(w,c2) = 1.5
- d(w,c3) = 1.5,
- which means that we predict C1.

Prediction in Multi-Class Classifier

A one-versus-one code matrix for $k = 4$ classes is as follows:

$$\begin{pmatrix} +1 & +1 & +1 & 0 & 0 & 0 \\ -1 & 0 & 0 & +1 & +1 & 0 \\ 0 & -1 & 0 & -1 & 0 & +1 \\ 0 & 0 & -1 & 0 & -1 & -1 \end{pmatrix}$$

Suppose our six pairwise classifiers predict $w = +1 -1 +1 -1 +1 +1$. We can interpret this as votes for $C_1 - C_3 - C_1 - C_3 - C_2 - C_3$;

multi-class scores and probabilities

- If we want to calculate multi-class scores and probabilities from binary classifiers, we have a number of different options.
  - We can use the distances obtained by loss-based decoding and turn them into scores by means of some appropriate transformation.
  - we can use the output of each binary classifier as features (real valued if we use the scores, binary if we only use the predicted class) and train a model that can produce multi-class scores, such as naive Bayes or tree models. This requires additional training.
  - A simple alternative that is also generally applicable and often produces satisfactory results is to derive scores from *coverage counts: the number of examples* of each class that are classified as positive by the binary classifer.

Margins and Loss Function of scoring classifier

- If we take the true class $c(x)$ as +1 for positive examples and -1 for negative examples, then the quantity $z(x) = c(x) * \hat{s}(x)$ is positive for correct predictions and negative for incorrect predictions.

- The quantity $z(x)$ is called the margin assigned by the scoring classifier.

We would like to reward large positive margins, and penalise large negative values. This is achieved by means of a so-called *loss function* $L : \mathbb{R} \mapsto [0, \infty)$ which maps each example's margin $z(x)$ to an associated loss $L(z(x))$.

Multi Class Classification with scoring Binary Classifier

- If our binary classifiers output scores :
- we assume that the sign of the scores '*si' indicates the class.*
- *We can then use* the appropriate entry in the code matrix *'c j i' to calculate a margin zi = si *c j i , which we* feed into a loss function *L.*
- We thus define the distance between a vector of scores *s and the j -th code word c j as*
  - $d(s,c_j) = \sum L(s_i * c_{ji})$,

  and we assign the class which minimizes this distance.
- *This way* of arriving at a multi-class decision from binary scores is called *loss-based decoding.*

Ex for Multi Class Classification with scoring Binary Classifier

Continuing the previous example, suppose the scores of the six pairwise classifiers are $(+5, -0.5, +4, -0.5, +4, +0.5)$. This leads to the following margins, in matrix form:

$$\begin{pmatrix} +5 & -0.5 & +4 & 0 & 0 & 0 \\ -5 & 0 & 0 & -0.5 & +4 & 0 \\ 0 & +0.5 & 0 & +0.5 & 0 & +0.5 \\ 0 & 0 & -4 & 0 & -4 & -0.5 \end{pmatrix}$$

$L_{01}(z) = 1$ if $z \leq 0$, and $L_{01}(z) = 0$ if $z > 0$;

Using 0–1 loss we ignore the magnitude of the margins and thus predict C_3 as in the voting-based scheme of Example 3.2. Using exponential loss $L(z) = \exp(-z)$, we obtain the distances $(4.67, 153.08, 4.82, 113.85)$.

Coverage Counts

- Suppose we have three classes and three binary classifiers which either predict positive or negative.
- The first classifier classifies 8 examples of the first class as positive, no examples of the second class, and 2 examples of the third class.
- For the second classifier these counts are 2, 17 and 1, and for the third they are 4, 2 and 8.
- Suppose a test instance is predicted as positive by the first and third classifiers.

$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 8 & 0 & 2 \\ 2 & 17 & 1 \\ 4 & 2 & 8 \end{pmatrix} = \begin{pmatrix} 12 & 2 & 10 \\ 14 & 19 & 11 \end{pmatrix}$$

multi-class probabilities

- In previous Example, we can divide the class counts by the total number of positive predictions. This results in the following class distributions: (0.80,0, 0.20) for the first classifier, (0.10,0.85,0.05) for the second classifier, and (0.29, 0.14, 0.57) for the third.
- The probability distribution associated with the combination of the first and third classifiers is

$$\frac{10}{24}(0.80, 0, 0.20) + \frac{14}{24}(0.29, 0.14, 0.57) = (0.50, 0.08, 0.42)$$

Multi-class probabilities

Similarly, the distribution associated with all three classifiers is

$$\frac{10}{44}(0.80, 0, 0.20) + \frac{20}{44}(0.10, 0.85, 0.05) + \frac{14}{44}(0.29, 0.14, 0.57) = (0.32, 0.43, 0.25)$$

Topics

- **Handling more than Two Classes**
- **Regression**
- **Unsupervised and Descriptive Learning**

Regression

- In all the tasks considered so far such as classification, scoring, and probability estimation---the label space was a discrete set of classes.

A *function estimator*, also called a *regressor*, is a mapping $\hat{f} : \mathcal{X} \to \mathbb{R}$. The regression learning problem is to learn a function estimator from examples $(x_i, f(x_i))$.

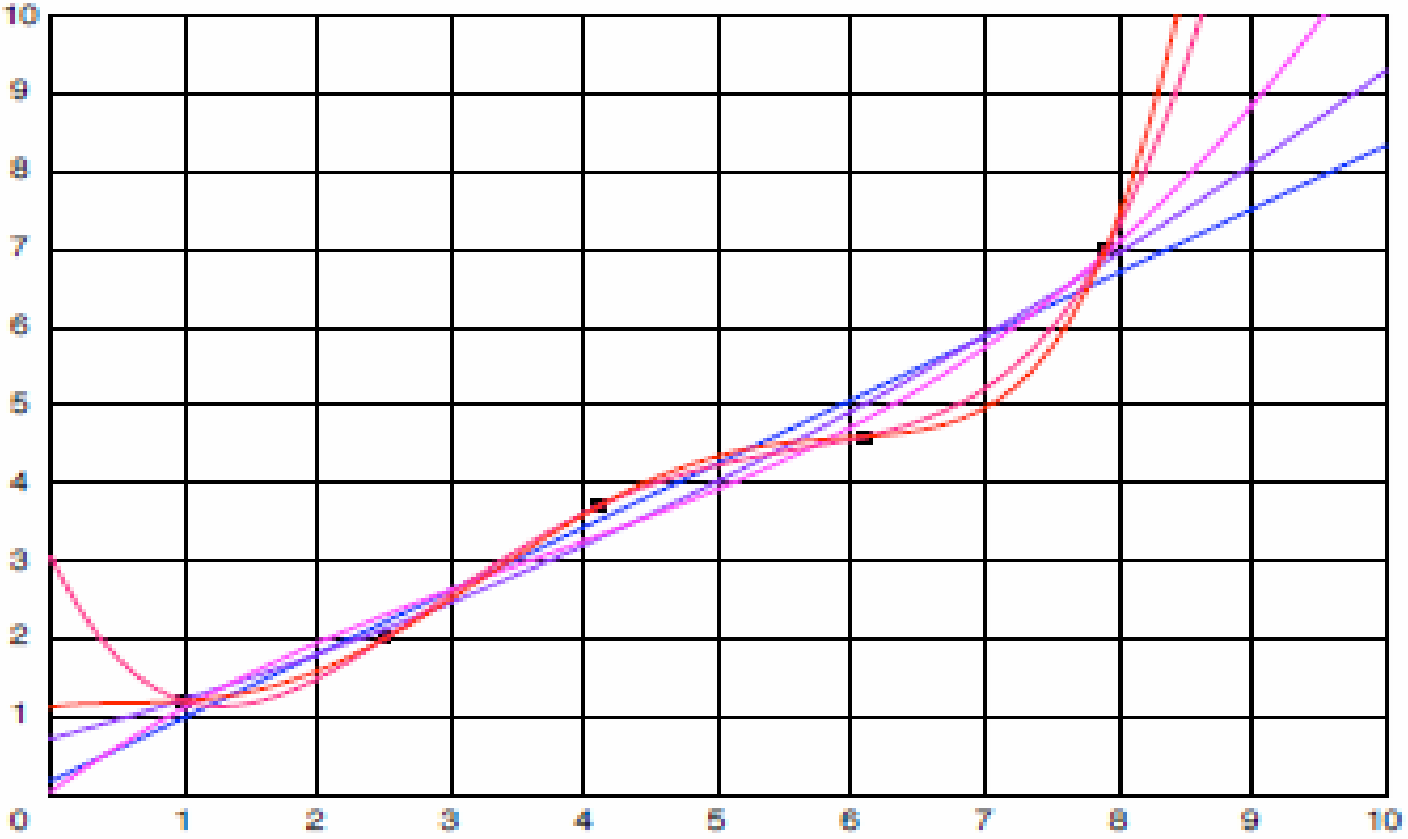
Example for Regression

Consider the following set of five points:

x	y
1.0	1.2
2.5	2.0
4.1	3.7
6.1	4.6
7.9	7.0

We want to estimate *y* by means of a polynomial in *x*

Example for Regression



Overfitting in Regression

An n-degree polynomial has $n+1$ parameters: e.g., a straight line $y = a \cdot x + b$ has two parameters, and the polynomial of degree 4 that fits the five points exactly has five parameters.

So the models that are able to fit the points exactly are the models with more parameters.

A rule of thumb is that, to avoid overfitting, the number of parameters estimated from the data must be considerably less than the number of data points.

Evaluating the Regression Model

- We have seen that classification models can be evaluated by applying a loss function to the margins, penalizing negative margins (misclassifications) and rewarding positive margins (correct classifications).

- Regression models are evaluated by applying a loss function to the *residuals* $f(x) - \hat{f}(x)$.

- The most common choice here is to take the squared residual as the loss function.

Errors in Machine Learning

- In machine learning, an error is a measure of how accurately an algorithm can make predictions for the previously unknown dataset.
- On the basis of these errors, the machine learning model is selected that can perform best on the particular dataset.
- There are mainly two types of errors in machine learning, which are:
  - **Reducible errors:** These errors can be reduced to improve the model accuracy. Such errors can further be classified into bias and Variance.
  - **Irreducible errors:** These errors will always be present in the model

Bias–Variance dilemma

- If we underestimate the number of parameters of the model, we will not be able to decrease the loss to zero, regardless of how much training data we have.

- On the other hand, with a larger number of parameters the model will be more dependent on the training sample, and small variations in the training sample can result in a considerably different model.

- This is called the ***bias–variance dilemma***

Bias

- Bias is the **difference between the average prediction of our model and the correct value** which we are trying to predict.
  - **High Bias** indicates more assumptions in the learning algorithm about the relationships between the variables.
  - **Less Bias** indicates fewer assumptions in the learning algorithm.
  - Examples of **Low-bias** machine learning algorithms: Decision Trees, k-Nearest Neighbors and Support Vector Machines.
  - Examples of **High-bias** machine learning algorithms: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

Variance

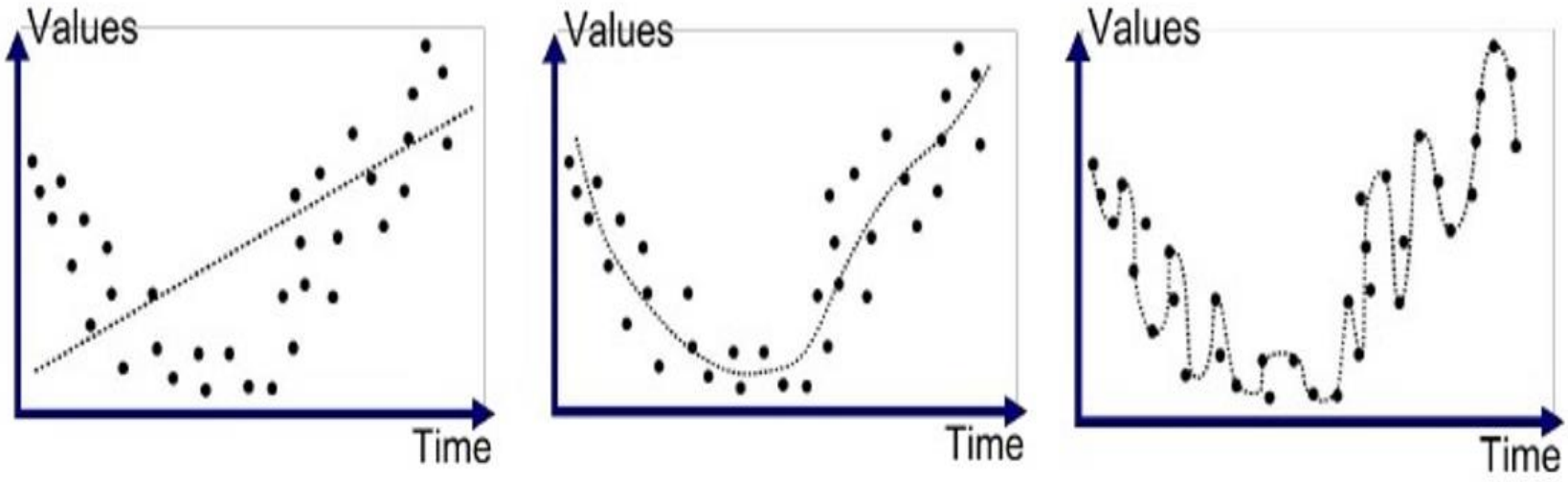
- The variance would specify the amount of variation in the prediction if different data was used.
- Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before.
- As a result, such models perform very well on training data but has high error rates on test data.
  - **Low Variance**: Suggests small changes to the estimate of the target function with changes to the training dataset.
  - **High Variance**: Suggests large changes to the estimate of the target function with changes to the training dataset

Bias-Variance

- when there is **a high bias**, it results in a very simplistic model that does not consider the variations very well. Since it does not learn the training data very well, it is called **Underfitting.**

- When there is **high variance** (for small variation in the input data the model prediction changes a lot), the model learns too much from the training data, it is called **overfitting.**

Bias and Variance

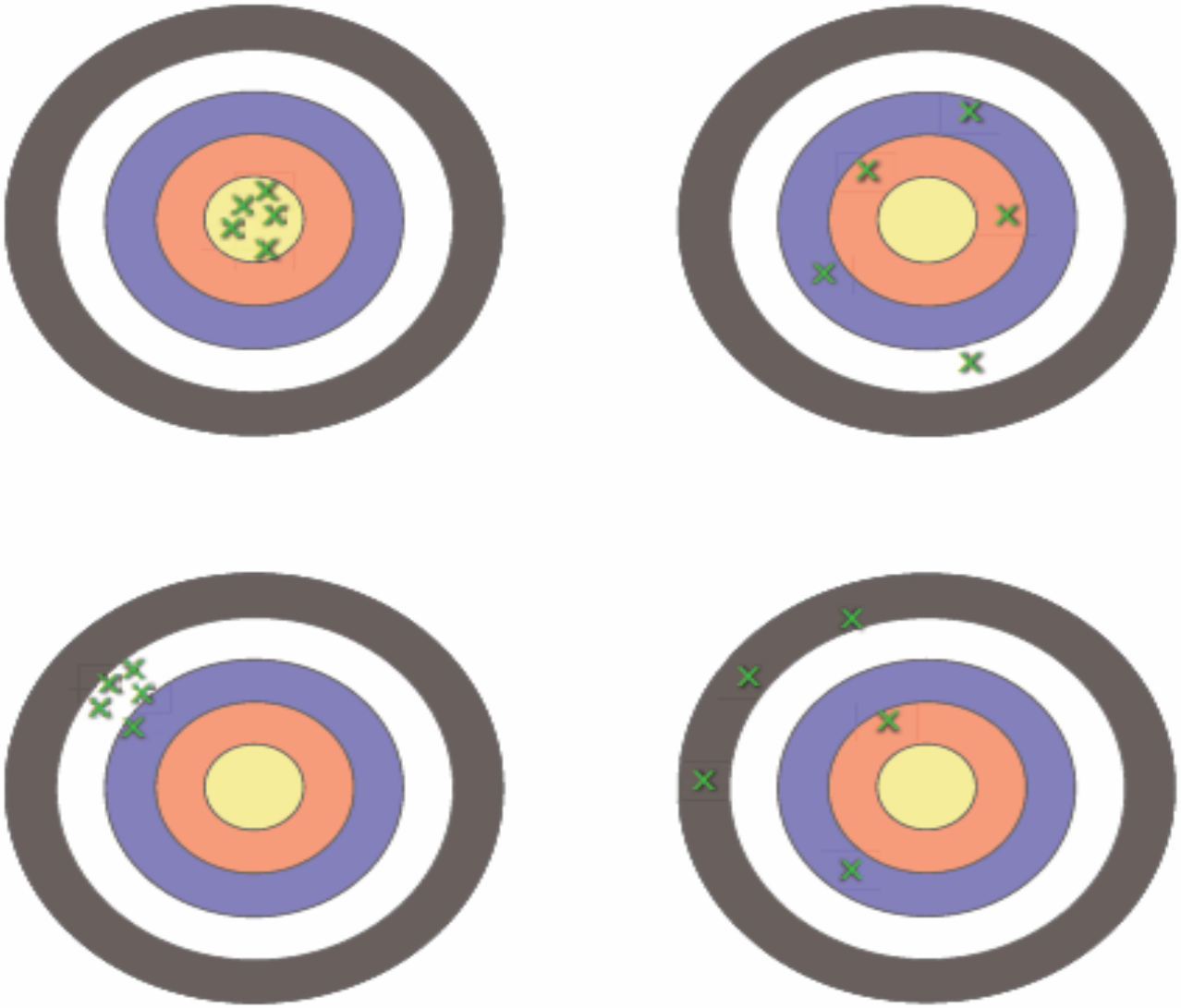


Underfitted

Good Fit/Robust

Overfitted

Bias- Variance with bulls eye diagram



Bias- Variance Diagram

- **Low-Bias, Low-Variance:** The combination of low bias and low variance shows an ideal machine learning model. However, it is not possible practically.

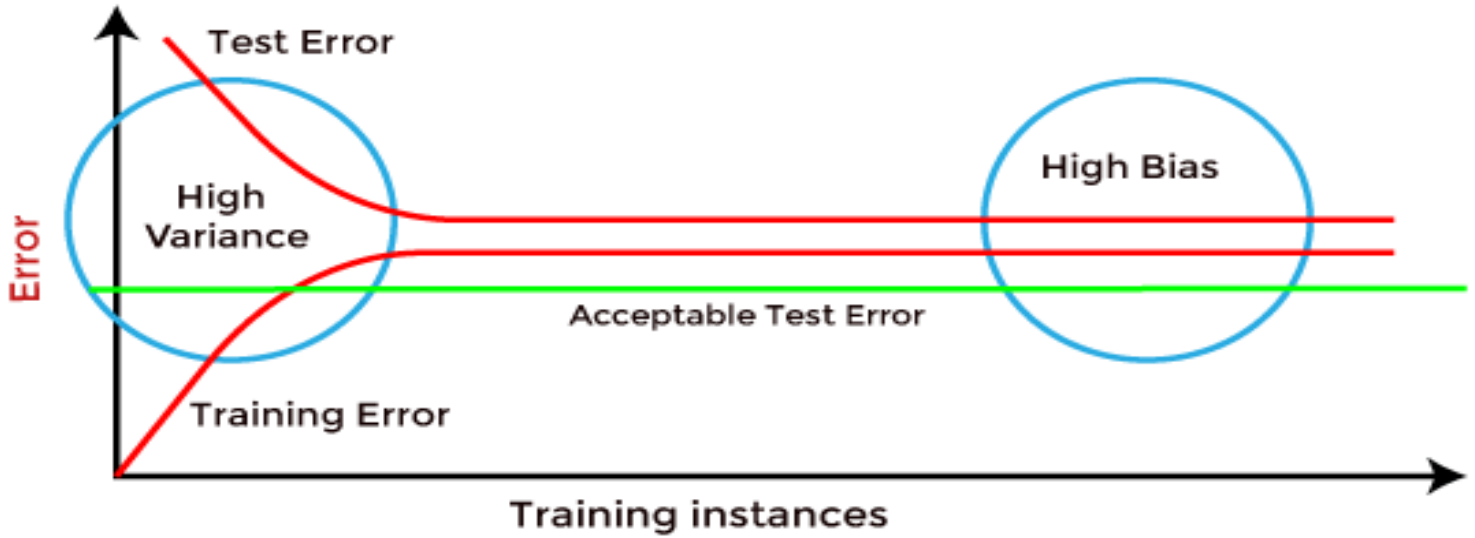
- **Low-Bias, High-Variance:** With low bias and high variance, model predictions are inconsistent and accurate on average. This case occurs when the model learns with a large number of parameters and hence leads to an **overfitting**

- **High-Bias, Low-Variance:** With High bias and low variance, predictions are consistent but inaccurate on average. This case occurs when a model does not learn well with the training dataset or uses few numbers of the parameter. It leads to **underfitting** problems in the model.

- **High-Bias, High-Variance:**
  With high bias and high variance, predictions are inconsistent and also inaccurate on average.

How to identify High variance or High Bias?

High variance can be identified if the model has:



- Low training error and high test error.

High Bias can be identified if the model has:

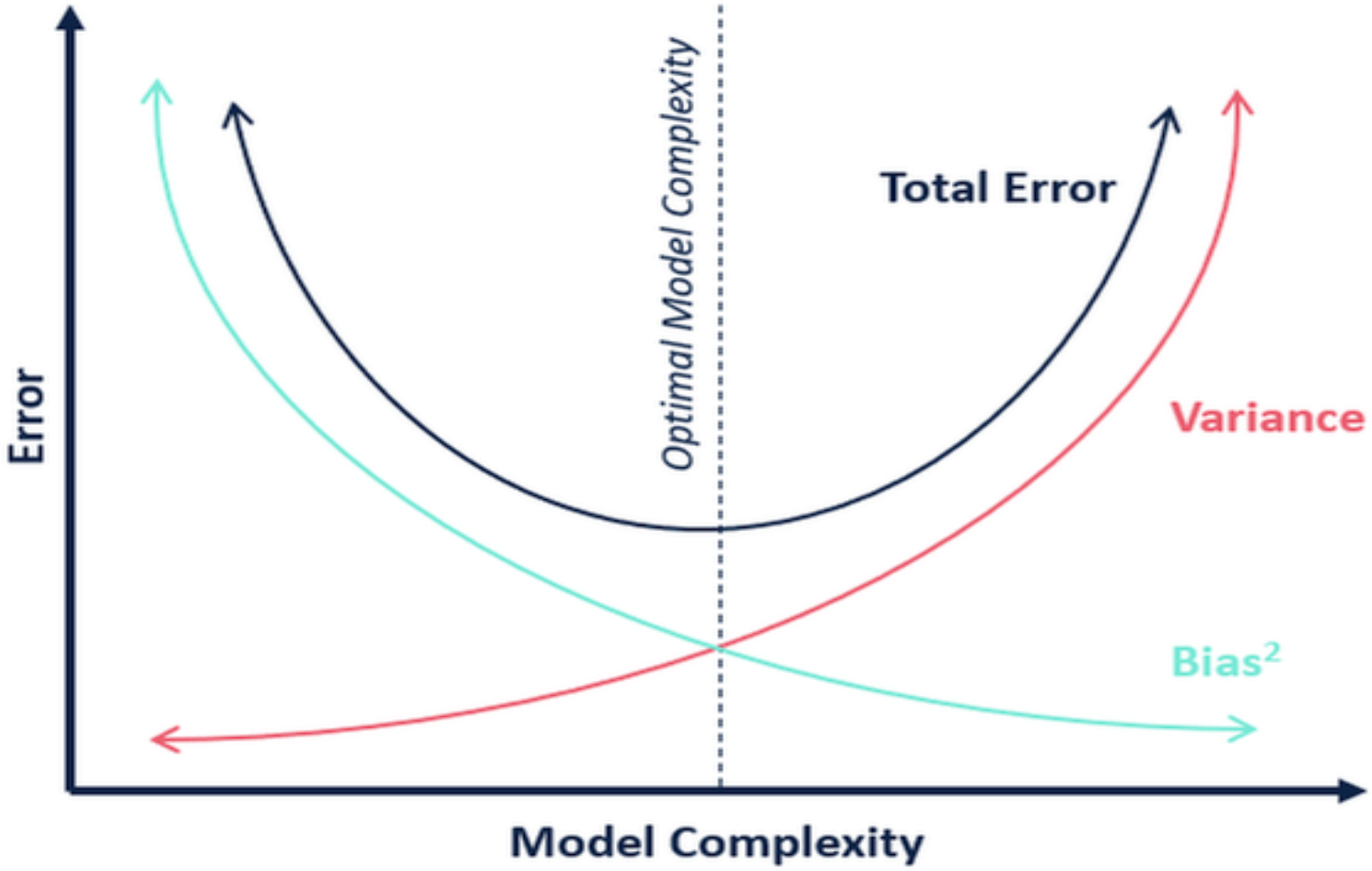
- High training error and the test error is almost similar to training error.

Ways to handle High variance or High Bias

- **Ways to reduce High Bias:**
  1. Increase the input features as the model is under-fitted.
  2. Decrease the regularization term.
  3. Use more complex models, such as including some polynomial features.

- **Ways to Reduce High Variance:**
  1. Reduce the input features or number of parameters as a model is over-fitted.
  2. Do not use a much complex model.
  3. Increase the training data.
  4. Increase the Regularization term.

Bias- Variance Tradeoff



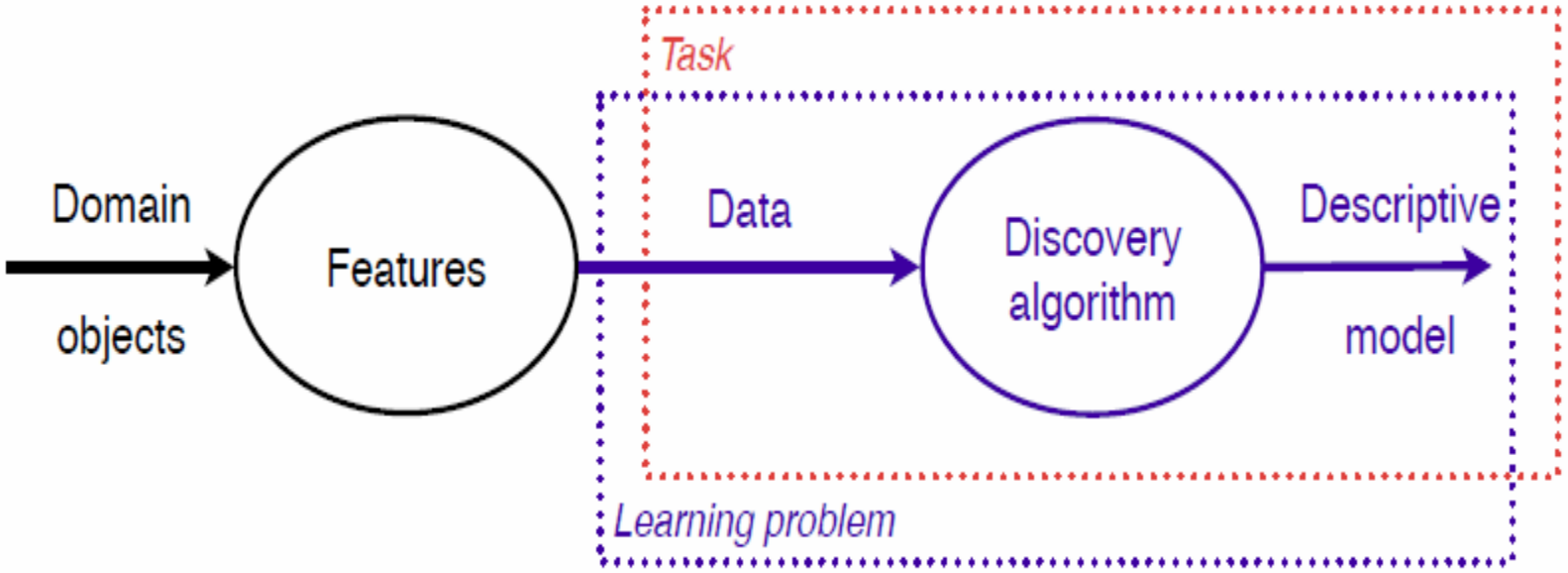
Topics

- **Handling more than Two Classes**
- **Regression**
- **Unsupervised and Descriptive Learning**

Unsupervised and Descriptive Learning

	Predictive model	*Descriptive model*
Supervised learning	classification, regression	subgroup discovery
Unsupervised learning	predictive clustering	descriptive clustering, association rule discovery

Descriptive Learning



In descriptive learning the task and learning problem coincide: we do not have a separate training set, and the task is to produce a descriptive model of the data.

Predictive and Descriptive Clustering

One way to understand clustering is as learning a new labelling function from unlabelled data. So we could define a 'clusterer' in the same way as a classifier, namely as a mapping $\hat{q}: \mathcal{X} \rightarrow \mathcal{C}$, where $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ is a set of new labels. This corresponds to a *predictive* view of clustering, as the domain of the mapping is the entire instance space, and hence it generalises to unseen instances.

A ***descriptive*** clustering model learned from given data *D* ⊆ *X would be a mapping ˆ q :D →C whose* domain is *D rather than X.*

Distance based Clustering

Most distance-based clustering methods depend on the possibility of defining a 'centre of mass' or *exemplar* for an arbitrary set of instances, such that the exemplar minimises some distance-related quantity over all instances in the set, called its *scatter*. A good clustering is then one where the scatter summed over each cluster – the *within-cluster scatter* – is much smaller than the scatter of the entire data set.

This analysis suggests a definition of the clustering problem as finding a partition $D = D_1 \uplus \ldots \uplus D_K$ that minimises the within-cluster scatter. However, there are a few issues with this definition:

- the problem as stated has a trivial solution: set $K = |D|$ so that each 'cluster' contains a single instance from D and thus has zero scatter;
- if we fix the number of clusters K in advance, the problem cannot be solved efficiently for large data sets (it is NP-hard).

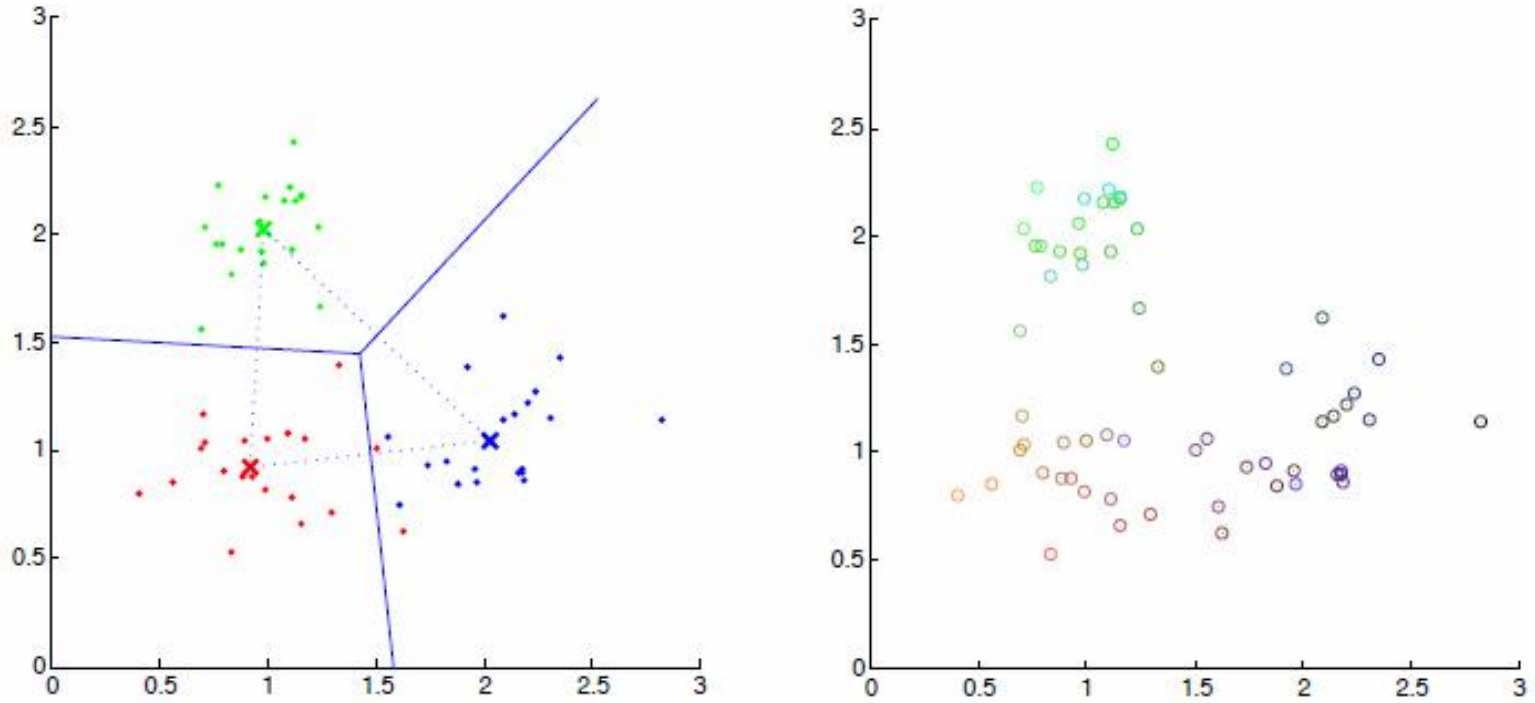
Distance based Clustering

The first problem is the clustering equivalent of overfitting the training data. It could be dealt with by penalising large K. Most approaches, however, assume that an educated guess of K can be made. This leaves the second problem, which is that finding a globally optimal solution is intractable for larger problems. This is a well-known situation in computer science and can be dealt with in two ways:

- by applying a heuristic approach, which finds a 'good enough' solution rather than the best possible one;
- by relaxing the problem into a 'soft' clustering problem, by allowing instances a degree of membership in more than one cluster.

Notice that a soft clustering generalises the notion of a partition, in the same way that a probability estimator generalises a classifier.

Distance based Clustering



(left) An example of a predictive clustering. The coloured dots were sampled from three bivariate Gaussians centred at $(1,1)$, $(1,2)$ and $(2,1)$. The crosses and solid lines are the cluster exemplars and cluster boundaries found by 3-means. **(right)** A soft clustering of the same data found by matrix decomposition.

Evaluating the Clustering

Suppose we have five test instances that we think should be clustered as $\{e1, e2\}, \{e3, e4, e5\}$. So out of the $5 \cdot 4 = 20$ possible pairs, 4 are considered 'must-link' pairs and the other 16 as 'must-not-link' pairs. The clustering to be evaluated clusters these as $\{e1, e2, e3\}, \{e4, e5\}$ – so two of the must-link pairs are indeed clustered together ($e1$–$e2$, $e4$–$e5$), the other two are not ($e3$–$e4$, $e3$–$e5$), and so on.

We can tabulate this as follows:

	Are together	*Are not together*	
Should be together	**2**	**2**	4
Should not be together	**2**	**14**	16
	4	16	20

We can now treat this as a two-by-two contingency table, and evaluate it accordingly. For instance, we can take the proportion of pairs on the 'good' diagonal, which is $16/20 = 0.8$. In classification we would call this accuracy, but in the clustering context this is known as the *Rand index*.

Other Descriptive Models

- **Association Rule Mining**
- **Subgroup Discovery**

Topics

- **Handling more than Two Classes**
- **Regression**
- **Unsupervised and Descriptive Learning**