Chapter 8
Case Study of Network-Based Supervised Learning: High-Level Data Classification

Abstract The power of computers to generalize to unseen data is intriguing. Computers have been used successfully to accurately predict prices of non-catalogued houses, trends in financial time series, or even to classify whether cancer tumors are benign or malign. One thing that all these tasks have in common is that computers are put forward to output answers to which they have not been explicitly programmed. A natural computational solution to estimate unseen data is to rely on the knowledge bases to which computers have been exposed, effectively mimicking the past behaviors. This chapter deals with supervised learning from a new learning perspective: a hybrid classification framework is presented that combines the decisions of low- and high-level classifiers. The low-level classifier realizes the classification task considering physical features of the input data, such as geometrical or statistical characteristics. In contrast, the high-level classification process checks the compliance of new test instances with the characteristic patterns formed by each of the classes that composes the training data. Test instances are declared members of those classes whose formed patterns are maintained with the introduction of those test instances. For this end, the high-level classifier extracts suitable organizational and topological descriptors of the network constructed from the input data. Using these network-based descriptors in a convenient collective manner, the high-level term is expected to promote the detection of data patterns with semantic and global meanings. The way we extract the patterns using these descriptors gives rise to several strategies to build up the high-level framework. In this book, we show two forms of pattern extraction strategies: using classical network measurements and employing dynamic information that is generated by several *tourist walk processes*. The ability of discovering high-level features formed by the data relationships is investigated using several artificial and real-world data sets. Here, we focus in situations in which the high-level term is able to identify intrinsic data patterns, but the low-level term alone fails to do so. This provides a clear motivation for the employment of a dual classification procedure (low + high). The obtained results reveal that the hybrid classification technique is able to improve the already optimized performances of traditional classification techniques. Finally, the hybrid classification approach is applied to recognize handwritten digits images.

T.C. Silva, L. Zhao, *Machine Learning in Complex Networks*,
DOI 10.1007/978-3-319-17290-3_8

8.1 A Quick Overview of the Chapter

This chapter treats the issue of supervised data classification by using not only physical features of data items, but also their high-level characteristics. As examples of low-level features, we may highlight: distance between data instances, conditional distribution of the data, and composition of the data neighborhood. In contrast, high-level characteristics can be defined under several different perspectives. One can understand them as semantic relationships that extrapolate the classical raw end-to-end relationships of the data (such as the edges in a network context). In this regard, subsets of these raw relationships may give rise to new concepts of the data organization that are ultimately not seen by low-level visions. For instance, data members of the same class (subset of relationships) may share homogeneous visions to each other, such as in a well-behaved distribution. Meanwhile, they may also indicate heterogeneous organizations for different classes. Hence, here we consider high-level features as descriptors that summarize the organization of the data relationships in a structural sense.

Despite being an interesting problem, most methods in the literature ignore the high-level relations among the data, such as the formation of clear patterns in the data relationships. In view of this gap, a hybrid classification technique has been proposed that takes into account both types of learning [24]. In essence, the low-level classification is guided by the labels and the physical features of the data items. In practical terms, it can therefore be implemented by any traditional techniques in the literature. In contrast, the high-level classification uses, besides the data labels, structural or pattern information of the data relationships. Here, new forms of extracting high-level features of the data relationships are discussed using a network-based approach. In this respect, the pattern extraction is conducted by exploring the complex topological properties of the underlying network constructed from the input data. In this framework, the low- and high-level classifiers are joined together via a suitable convex linear combination, which is calibrated by what is called the *compliance term*. Basically, the compliance term adjusts the importance that is given for the low- and high-level decisions.

In this chapter, two different implementations of the high-level term are discussed, both relying on a networked representation of the data, as follows:

- The first one comprises a weighted combination of three classical network measurements, namely the assortativity, the clustering coefficient, and the average degree;
- The second one is composed of two quantities that are directly derived from the dynamics of tourist walks, which are the cycle and the transient lengths.

The compliance term plays a crucial role in the classification process. As such, several analyses are going to be provided in order to show the impact of the compliance term in data sets with different distributions and particularities that range from completely well-posed classes to highly overlapping classes. As a quick glimpse of the final results, we show that one must raise the decision influence of the

high-level classifier as the joint distribution of the classes becomes more complex in the sense of the existence of overlapping regions.

Once the hybrid classification technique is properly presented, we explore the effectiveness of the model by delving into the real-world application of handwritten digits and letters recognition. Additionally, to illustrate the influence that the compliance term makes upon the final decision in this real-world problem, a small manuscript digits network sampled from the real data set is displayed. Such a network shows that the high-level term is really necessary in special occasions.

8.2 Motivation

Data items are not isolated points in the attribute space but instead tend to form certain patterns when looked in a collective manner. For example, in Fig. 8.1, the test instance represented by the "triangle" (purple) will probably be classified as a member of the "square" (blue) class if only distances among data instances are considered. In contrast, if we take into account the relationship and semantic meaning among the data items, we would intuitively classify the "triangle" item as a member of the "circle" (red) class, since a clear pattern of a "moon" contour is formed. The human (animal) brain has the ability to identify patterns according to the semantic meaning of the input data. But, this feature still stands as a hard task for computers. Supervised data classification that not only considers physical

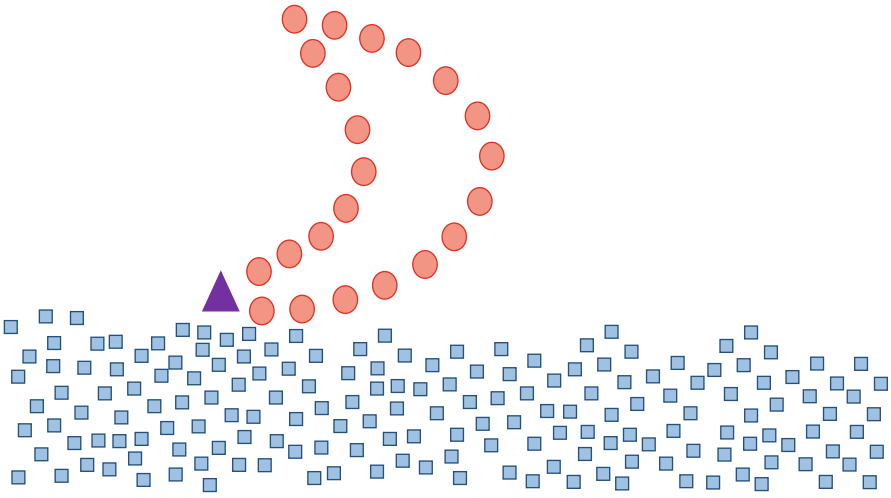

Fig. 8.1 A simple example of a supervised data classification task in which there exists a class with a clear pattern, the "*circle*" (*red*) class, and another without apparent structural organization, the "*square*" (*blue*) class. The goal is to classify the "*triangle*" (*purple*) data item. Traditional (low-level) classifiers would have trouble in classifying such item, since they derive their decisions merely based on physical measures

attributes of data items but also their pattern formations is here referred to as *high-level classification*.

The hybrid classification technique presents a way to classify data combining two semantically different views: the low-level view applies physical features of the data and the high-level view checks pattern formation of the data. In this sense, the co-training technique [4] is related to the hybrid classification technique. Co-training requires two independent views of the data. It first learns a separate classifier for each view using the labeled data items. Then, the most confident predictions of each classifier on the unlabeled data are then used to iteratively construct additional labeled training data. However, the "independent views" in co-training are generated by low-level classification techniques, i.e., the "independence" is at the physical feature level. On the other hand, the hybrid technique [24, 28] gives "independent views" from different levels ranging from physical features to semantic meaningful patterns. In the same sense, another related technique is the committee machine, which consists of an ensemble of classifiers [12]. In this case, each classifier makes a decision by itself and all these decisions are combined into a single response by a voting scheme. The combined response of the committee machine is supposed to be superior to those of its constituent experts. Again, all the involved techniques are low-level ones.

Another strong feature of the hybrid classification technique is that it is an across-network technique, i.e., it considers the network constructed from the input data as a whole and the global pattern of the network is taken into account. In the across-network approach, we take a set of network measures for each constructed data network, in such a way to characterize the global patterns formed by the underlying network via a measure vector. In this extraction process, we observe each network as if we were outside of it and hence each extracted measure represents a different view of the network.

In contrast, in the within-network approach, we look at the network inside it. In the within-network case, we basically have two objectives:

1. Making a probabilistic or deterministic inference to find out the best route from one vertex to another. For example, we may determine the class label of a test vertex using an inference process starting from an already labeled vertex.
2. Information transmission or diffusion. In this case, we propagate some kind of information to the entire or a portion of the network. For example, we may propagate labels from some vertices to the entire network in semi-supervised learning.

Figure 8.2 illustrates a schematic of the differences of the across- and within-network approaches.

The across-network feature of the hybrid framework classification contrasts with several other related works, such as the Semantic Web [1, 7, 23] and Statistical Relational Classification (SRC). Semantic Web uses ontologies to describe the semantics of the data. Even though it is a promising idea, it still presents several difficult challenges. A key challenge in creating Semantic Web is the semantic mapping among the ontologies, i.e., there are more than one ontology to describe

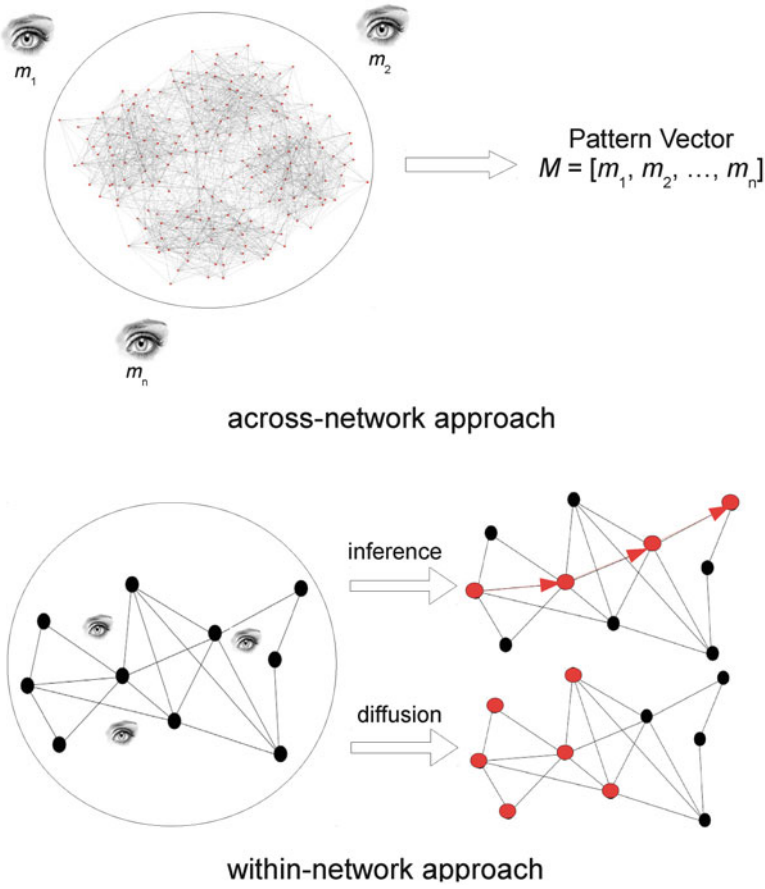

Fig. 8.2 Differences of across- and within-network approaches. In the across-network approach we look at the network as an "outsider" from different viewpoints. In the within-network, we either perform inference or label diffusion with processes that take place inside the network

the same data item. Another challenge is the one-to-many mapping of concepts in the Semantic Web, since most techniques are only able to induce one-to-one mapping, which does not correspond to real-world problems. In the case of statistical relational classification (SRC), it predicts the category of an object based on its attributes and its links and attributes of linked objects. Network-based SRC can be categorized into three main groups [9]: collective inference [9, 19, 29, 36–38, 40], network-based semi-supervised learning [5, 25–27, 39], and contextual classification techniques [2, 18, 20, 31, 32, 34, 35]. In all the cases, the labels are propagated from pre-labeled vertices to unlabeled vertices considering the local relationships or certain smoothness criteria. Therefore, those techniques are kind of within-network ones. On the other hand, the classification method introduced in high-level classification technique presents a different approach. It does not

consider the paths within the network, instead, it characterizes the pattern formation of the whole network by exploiting its topological properties using a set of network measurements. In this case, each of these measures views the network as an outsider in different perspectives.

8.3 Model Description

In this section, the high-level classification technique is presented. Specifically, Sect. 8.3.1 supplies the general idea of the model and Sect. 8.3.2 deals with the methodology for building the hybrid classification framework.

8.3.1 Fundamental Ideas Behind the Model

Suppose that it is given a training set $\mathscr{X}_{\text{training}} = \{(x_1, y_1), \ldots, (x_L, y_L)\}$ with L labeled items, where the first component of the i-th tuple $x_i = (x_{i1}, \ldots, x_{iP})$ denotes the attributes of the P-dimensional i-th training instance and the second component $y_i \in \mathscr{Y}$ is the class label of x_i. Denote $Y = |\mathscr{Y}|$ as the number of classes in the classification problem. Recall that we have a binary classification problem when $Y = 2$ and a multiclass classification problem when $Y > 2$.

As usual, the goal of supervised learning is to learn a mapping $x \mapsto y$. Normally, the generalization power of the constructed classifier is checked against a test set of U items $\mathscr{X}_{\text{test}} = \{x_{L+1}, \ldots, x_{L+U}\}$ without label information.

The classification process consists of two phases: the *training phase* and the *classification phase*. In the training phase, the classifier is induced or trained by using the training instances (labeled data) in $\mathscr{X}_{\text{training}}$. In network-based models, the classifier's model is represented by a network that is formed from the input data and the associated labels. We term this output network from the training phase as the training network. In the classification phase, the labels of the test instances in $\mathscr{X}_{\text{test}}$ are predicted using the induced classifier. That is, we start off from the training network and make some modifications to accommodate the unseen test instances. Using this slightly modified network, we predict the label of the test instance. This modified network is referred to as the classification network.

8.3.1.1 Training Phase

In this phase, the training data is transformed to a network $\mathscr{G}$ using a network formation technique $g : \mathscr{X}_{\text{training}} \mapsto \mathscr{G} = \langle \mathscr{V}, \mathscr{E} \rangle$. Hence, we have $V = |\mathscr{V}|$ vertices and $E = |\mathscr{E}|$ edges in the training network. Each vertex in $\mathscr{V}$ represents a training instance in $\mathscr{X}_{\text{training}}$, so that $V = L$ holds.

The network is constructed using a combination of the ϵ-radius and k-nearest neighbors (k-NN) techniques. As shown in the network construction chapter, both

approaches have their limitations,[1] i.e., these techniques, applied in the isolated form, may generate densely connected networks or may split the vertices into disconnected components.

For this reason, the combination of ϵ-radius and k-nearest neighbors techniques is used to construct the training network. The neighborhood of a training vertex x_i is given by:

$$\mathcal{N}_{\text{training}}(x_i) = \begin{cases} \epsilon\text{-radius}(x_i, y_i), & \text{if } |\epsilon\text{-radius}(x_i, y_i)| > k \\ k\text{-NN}(x_i, y_i), & \text{otherwise} \end{cases} \tag{8.1}$$

in which y_i denotes the class label of the training instance x_i, ϵ-radius(x_i, y_i) returns the set $\{x_j, j \in \mathcal{V} : d(x_i, x_j) <= \epsilon \wedge y_i = y_i\}$, and k-NN(x_i, y_i) returns, in principle, the set containing the k nearest vertices of the same class as x_i. There is a caveat, however, in this returned set of the k-NN technique. Suppose we rank all of data items in accordance with their similarities in relation to x_i. Let this sorted sequence be denoted by $\mathcal{S}(x_i) = \{x_i^{(1)}, \ldots, x_i^{(k-1)}, x_i^{(k)}, x_i^{(k+1)}, \ldots, x_i^{(Y(x_i)-1)}\}$, where $Y(x_i)$ is the number of data items of the same class as x_i. In this notation, $x_i^{(1)}$ and $x_i^{(Y(x_i)-1)}$ are the most and the least similar data items to x_i, respectively. As pointed out, we first try to connect x_i to its k most similar data items, i.e., $\{x^{(1)}, \ldots, x^{(k)}\}$. However, if we end up getting more than one graph component of the same class as x_i, we then drop the least similar data item among those k most similar data items, that is, we discard $x_i^{(k)}$, and attempt to connect x_i to the next most similar data item $x_i^{(k+1)}$. This process is recursively performed until we find connections of x_i to other data items in such a way to prevent the emergence of more than a network component of the same class.

Note that the ϵ-radius technique is used for dense regions ($|\epsilon\text{-radius}(x_i)| > k$), while the k-NN is employed for sparse regions. With this mechanism, it is expected that each class is represented by a unique and single component. Below, we present a simple contextual example showing the network formation technique.

Example 8.1. Consider the scatter plot in Fig. 8.3, where the task is to determine to which neighbors the central vertex of the red class (dark gray) connects. Consider that $k = 2$ and ϵ is the radius illustrated in the figure. As there are $3 > k$ vertices in the ϵ-neighborhood, the area depicted in the figure is considered as a dense region and the ϵ-radius technique is employed. In this way, the central red (dark gray) vertex is connected to the other three red (dark gray) vertices reached by this radius.

[1] Revisit Sect. 4.3 for a thorough analysis of the shortcomings and advantages of using k-NN and ϵ-radius network formation techniques.

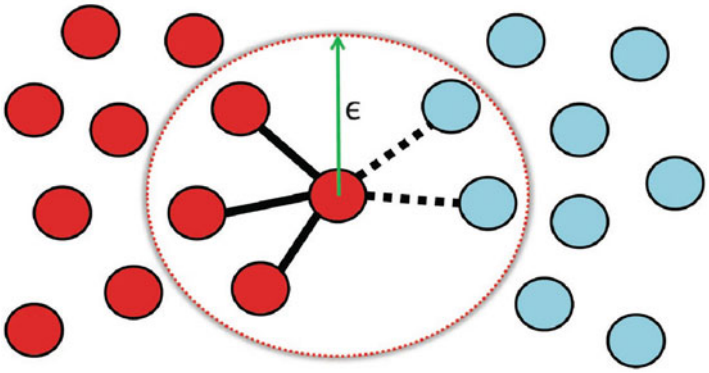

Fig. 8.3 Illustration of the network formation technique consisting in a combination of the k-nearest neighbor and the ϵ-radius techniques. In the depicted network, there are two classes: the *red* (*dark gray*) and the *blue* (*light gray*) classes. Since $k = 2$, the local region is considered as densely populated. Thus, the ϵ-radius technique is employed. As the centralized vertex belongs to the *red* (*dark gray*) class, it is only permitted to be linked to other *red* (*dark gray*) class vertices

For the sake of clarity, Fig. 8.4a shows a schematic of how the network looks like for a multiclass classification with $Y = 3$ when the training phase is completed. In this case, each class holds a representative component. In the figure, the surrounding circles denote these components: $\mathscr{G}_{C_1}$, $\mathscr{G}_{C_2}$, and $\mathscr{G}_{C_3}$.

8.3.1.2 Classification Phase

In the classification phase, the unlabeled data items (test instances) in the $\mathscr{X}_{\text{test}}$ are presented to the classifier one by one. The neighborhood of the test instance x_i is defined using the following rule:

$$\mathscr{N}_{\text{classification}}(x_i) = \begin{cases} \epsilon\text{-radius}(x_i), & \text{if } |\epsilon\text{-radius}(x_i)| > k. \\ k - \text{NN}(x_i), & \text{otherwise.} \end{cases} \tag{8.2}$$

Equation (8.2) means that the ϵ-radius connects every vertex within the radius ϵ, disregarding the class labels of the neighbor vertices. If the region is sparse enough, i.e., there are less than k vertices in this neighborhood, then the k-NN approach is employed. The modified network with the test instance is the classification network. In the high-level model, each class retains a network component. Once a test item is inserted, each component (class) calculates the changes that occur in its pattern formation with the insertion of this test instance by means of a set of complex network measures. If slight or no changes occur, then it is said that the test instance is in compliance with that class pattern. As a result, the high-level classifier yields a large membership value for that test instance on that class. Conversely, if these changes dramatically modify the class pattern, then the high-level classifier produces a small membership value on that class.

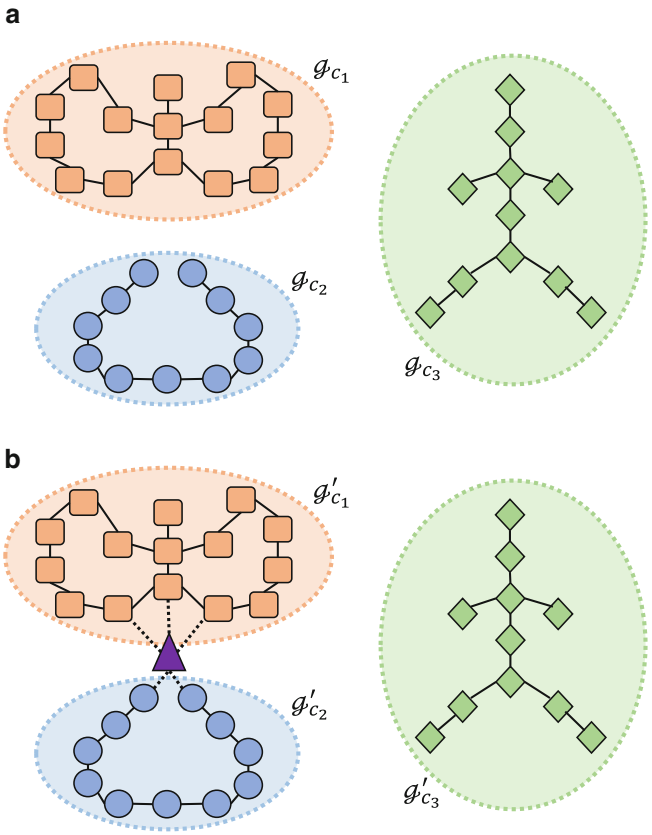

Fig. 8.4 Overview of the two phases of the supervised learning. (**a**) Schematic of the network in the training phase. (**b**) Schematic of how the inference in the classification phase is performed

Figure 8.4b shows a schematic of how the classification process is performed. The test instance (triangle-shaped) is inserted using the traditional ϵ-radius technique, resulting in the classification network. Due to its insertion, the class components become altered: $\mathcal{G}'_{C_1}$, $\mathcal{G}'_{C_2}$, and $\mathcal{G}'_{C_3}$, in which each of them is a component surrounded by a circle in Fig. 8.4b. It may occur that some class components do not share any links with this test instance. In the figure, this happens with $\mathcal{G}'_{C_3}$. In this case, the test instance does not comply with the pattern formation of the class component. For the components that share at least a link ($\mathcal{G}'_{C_1}$ and $\mathcal{G}'_{C_2}$), each of it calculates, in isolation, the impact on its pattern formation by virtue of the insertion of the test instance. For example, when we check the compliance of the test instance with the component $\mathcal{G}'_{C_1}$, the connections from the test instance to the component $\mathcal{G}'_{C_2}$ are ignored, and vice versa.

At the same time, a low-level classifier is also constructed to predict the membership of the test instance for each class. At the end, the predictions produced by both classifiers are combined to produce the final decision. Each of the low- and high-level technique provides a different view of the data set. The low-level techniques usually has good performance on well distributed and well separated data sets, while the high-level one has the ability to identify semantic meaning of the data. We may also understand the classification process in such a way that the low-level one guarantees the basic performance of the classification results and the high-level one explores complex and special patterns hidden in the data set.

8.3.2 Derivation of the Hybrid Classification Framework

The high-level classification complements the performance of the learning procedure by exactly capturing the formation of patterns in the data relationships. For this reason, a hybrid classification technique F is introduced. It consists of a convex combination of two terms:

1. A low-level classifier. It can be any traditional classification technique, for instance, a decision tree, Support Vector Machines (SVM), neural networks, Bayesian learning, or a k-NN classifier;
2. A high-level classifier, which is responsible for classifying test instances according to their pattern formation with the training data or network.

Specifically, the hybrid framework yields the decision $F_i^{(y)}$ representing the membership of the test instance $x_i \in \mathscr{X}_{\text{test}}$ with respect to the class $y \in \mathscr{Y}$ as follows:

$$F_i^{(y)} = (1 - \rho)L_i^{(y)} + \rho H_i^{(y)}, \tag{8.3}$$

in which $L_i^{(y)} \in [0, 1]$ and $H_i^{(y)} \in [0, 1]$ denote the low- and high-level predictions of test instance x_i's membership towards class y and $\rho \in [0, 1]$ is the *compliance term*, which plays the role of counterbalancing the classification decisions supplied by both classifiers. Whenever $L_i^{(y)} = 1$ and $H_i^{(y)} = 1$, x_i is very similar (low-level) and perfect complies (high-level) with class y. In contrast, whenever $L_i^{(y)} = 0$ and $H_i^{(y)} = 0$, x_i does not present any similarities (low-level) nor complies to the pattern formation (high-level) of class y. Values in-between these two extremes lead to natural uncertainty in the classification process and are found in the majority of times during a classification task. It is worth noting that (8.3) is a convex combination of two scalars, since the sum of the coefficients is unitary. Thus, as the domains of $L_i^{(y)}$ and $H_i^{(y)}$ range from 0 to 1, it is guaranteed that $F_i^{(y)} \in [0, 1]$. Therefore, (8.3) provides a fuzzy classification method. Moreover, it is valuable to mention that, when $\rho = 0$, (8.3) reduces to a common low-level classifier.

The test instance x_i receives the label from that class $y \in \mathscr{Y}$ that maximizes (8.3). Mathematically, the estimated label of the test instance x_i, $\hat{y}_i$, is computed as:

$$\hat{y}_i = \max_{y \in \mathscr{Y}} F_i^{(y)}. \tag{8.4}$$

Now, we proceed to a detailed analysis of the high-level classification term H. In the high-level classification, the inference of pattern formation within the data is processed using the generated training network. Due to the network formation process, the training network has the following structural constraints:

1. Each class $y \in \mathscr{Y}$ is an isolated network component; and
2. Each class $y \in \mathscr{Y}$ retains a representative and unique network component.

With these two network properties in mind, the pattern formation of the data is quantified via suitable combinations of network measurements. These measures are chosen in a way to cover relevant high-level aspects of the class component. In special, it is often desirable to capture strictly local, mixed, and global network characteristics, such as to cover structural network aspects in different perspectives.[2] The high-level classifier accepts an arbitrary number of network measurements. Suppose $m > 0$ network measures are selected to comprise the high-level classifier H. Mathematically, the high-level prediction on the membership of test instance $x_i \in \mathscr{X}_{\text{test}}$ with respect to class $y \in \mathscr{Y}$, here written as $H_i^{(y)}$, is given by:

$$H_i^{(y)} = \frac{\sum_{u=1}^{m} \alpha(u) \left[1 - f_i^{(y)}(u)\right]}{\sum_{g \in \mathscr{Y}} \sum_{u=1}^{m} \alpha(u) \left[1 - f_i^{(g)}(u)\right]}, \tag{8.5}$$

in which $\alpha(u) \in [0, 1], \forall u \in \{1, \ldots, m\}$, is a user-defined parameter that indicates the influence of each network measure in the classification process and $f_i^{(y)}(u)$ is a function that depends on the u-th network measure applied to the i-th data item with regard to the class y. This function is responsible for providing an answer whether or not the test instance x_i presents the same patterns or organizational features of the class y. The denominator in (8.5) has been introduced for normalization matters. Indeed, Eq. (8.5) is a valid convex combination of network measures if and only if:

$$\sum_{u=1}^{m} \alpha(u) = 1. \tag{8.6}$$

The functional form $f_i^{(y)}(u)$ is given by:

$$f_i^{(y)}(u) = \Delta G_i^{(y)}(u) p^{(y)}, \tag{8.7}$$

[2] Revisit Sect. 2.3.5 for definitions on the classification of network measurements.

in which $\Delta G_i^{(y)}(u) \in [0, 1]$ is the variation of the u-th network measure that occurs on the component representing class y if x_i joins it and $p^{(y)} \in [0, 1]$ is the proportion of data items pertaining to class y.

Remembering that each class has a single representative component, the strategy to check the pattern compliance of test instance x_i is to examine whether its insertion causes a great variation of the network measures in the class components. If for some class component the variation is small, then x_i is in compliance with all of the other training data items that comprise that class component, i.e., it follows the same pattern as the original members of that class. This case happens when the structural features of the network component are maintained with the introduction of x_i. Otherwise, if its insertion is responsible for a significant variation of the component's network measures, then x_i may not belong to that class in the structural sense. In this case, the structural properties of the class component are altered due to the insertion of x_i. These two behaviors are exactly captured by (8.5) and (8.7). To see that, note that a small variation of $f(u)$ causes a large membership value output by H; and vice versa. For didactic purposes, we show this concept in a simple example.

Example 8.2. Suppose a network in which there exists two equally sized classes, namely A and B. For simplicity, let us use a single network measure to quantify the pattern formation ($m = 1$). The goal is to classify a test instance x_i. Hypothetically, say that $\Delta G_i^{(\mathrm{A})}(1) = 0.7$ and $\Delta G_i^{(\mathrm{B})}(1) = 0.3$. In a pattern formation view, x_i has a bigger chance of belonging to class B, since its insertion causes a smaller impact on the pattern formation formed by class B than on the one formed by class A.

In general, a data set usually encompasses several classes of different sizes and many network measures are sensitive to the component size. In order to avoid the unbalanced problem, the term $p^{(y)}$ in Eq. (8.7) is introduced, which is the proportion of vertices that class y has. Formally, it is given by:

$$p^{(y)} = \frac{1}{V} \sum_{u=1}^{V} \mathbb{1}_{[y_u = j]}, \tag{8.8}$$

in which V is the number of vertices and $\mathbb{1}_{[.]}$ is the indicator function that yields 1 if the argument is logically true, or 0, otherwise.

The following simple example illustrates the effect of the term $p^{(y)}$.

Example 8.3. Consider a network in which there are two classes: A and B, but now A's size is ten times bigger than B's. From (8.8), $p^{(A)} = {}^{10}/_{11}$ and $p^{(B)} = {}^{1}/_{11}$. Without the term $p^{(y)}$, it is expected that variations of the network measures in the component A to be considerably smaller than those in component B, because of the size differences. This occurs even when the test instance x_i complies more with class B. By considering the term $p^{(y)}$, the larger value of $p^{(\mathrm{A})}$ over $p^{(\mathrm{B})}$ cancels out the effects of unbalanced classes in the calculation of the network measures. In this way, the component size modulates the variations of the topological descriptors when deciding on the compliance of new test instances.

8.4 Possible Ways of Composing High-Level Classifiers

Two network-based high-level classifications have been proposed [24, 28]. The first one makes use of a mixture of classical network measurements, namely the assortativity, the clustering coefficient, and the average degree measures. The second one uses the dynamical information generated by several tourist walks processes. In the following, the two techniques are discussed.

8.4.1 High-Level Classification Using a Mixture of Complex Network Measures

In this section, the first implementation of the high-level classification is introduced [24], which is composed of three complex network measures, which are: assortativity, clustering coefficient, and average degree. In spite of having chosen these measures, it is worth emphasizing that other network measures can be also plugged into the high-level classifier through Eq. (8.5). The reason these three measures have been chosen is as follows: the degree measure figures out strictly local or scalar information of each vertex in the network; the clustering coefficient of each vertex captures local structures by means of counting triangles formed by the current vertex and any of its two neighbors; the assortativity measure considers not only the current vertex and its neighbors, but also the second level of neighbors (neighbor of neighbor), the third level of neighbors, and so on. We perceive that the three measures characterize the network topological properties in a local to global fashion. In this way, the combination of these measures is expected to capture the

pattern formations of the underlying network in a systematic manner. Below, we revisit these three measures and show how to incorporate them into the high-level classification.

8.4.1.1 First Network Measure: Assortativity

Assortativity is the preference of vertices in a network to link to others that are similar in term of vertices' degrees. This measure has been discussed in the chapter dealing with the fundamentals of Complex Networks (cf. Definition 2.36). We now derive $\Delta G_i^{(y)}(1)$ using the assortativity measure. Consider that the membership of the test instance x_i with respect to the class y is going to be determined. The actual assortativity measure of the component representing class y (before the insertion of x_i) is given by $r^{(y)}$ (step performed in the training phase). Then, we temporarily insert x_i into the component representing class y using the explained network formation technique (classification phase) and quantify the new component's assortativity measure, here denoted as $r'^{(y)}$. This procedure is performed for all of the classes $y \in \mathscr{Y}$. It may occur that some classes $u \in \mathscr{Y}$ do not share any connections with the test instance x_i. Using this approach, $r^{(u)} = r'^{(u)}$, which is undesirable, since this configuration would state that x_i complies perfectly with class u. In order to overcome this problem, a simple postprocessing is necessary: for all components $u \in \mathscr{Y}$ that do not share at least one link with x_i, we deliberately set $r^{(u)} = -1$ and $r'^{(u)} = 1$, i.e., the maximum possible difference. One may interpret this postprocessing as a way to state that x_i does not share any pattern formation with class u, since it is not even connected with it.

In view of this, we are able to calculate $\Delta G_i^{(y)}(1)$ for all $y \in \mathscr{Y}$ as follows:

$$\Delta G_i^{(y)}(1) = \frac{|r'^{(y)} - r^{(y)}|}{\sum_{u \in \mathscr{Y}} |r'^{(u)} - r^{(u)}|}, \tag{8.9}$$

in which the denominator is introduced only for normalization matters. According to (8.9), for components in which the insertion of x_i result in a considerable variation of the assortativity measure, $\Delta G_i^{(y)}(1)$ is large, and, consequently, by (8.7), $f_i^{(y)}(1)$ is also large. In light of this, the high-level classifier H produces a small membership value, as (8.5) reveals. Conversely, for insertions that do not cause a considerable variation of the assortativity, $\Delta G_i^{(y)}(1)$ is small, resulting in a small $f_i^{(y)}(1)$. As a consequence, the high-level classifier H produces a large membership value. In this way, the high-level classifier favors test instances that do not impact much the organizational and pattern features of a class.

8.4.1.2 Second Network Measure: Clustering Coefficient

Clustering coefficient is an indicator of the degree to which nodes in a network tend to cluster together in a triangular manner. This measure has also been

investigated in the chapter dealing with the fundamentals of Complex Networks (cf. Definitions 2.46 and 2.47). We motivate the use of the clustering coefficient by the following facts: components with large clustering coefficient are found to have a modular structure with a high density of local connections, while components with small average clustering values tend to have many long-range connections, with the absence of local structures.

The derivation of $\Delta G_i^{(y)}(2)$ is rather analogous to the previous case, except for a simple detail: In this case, for all components $u \in \mathscr{Y}$ that do not share at least one link with the test instance x_i, we intentionally fix $CC^{(u)} = 0$ and $CC'^{(u)} = 1$, i.e., the maximum possible difference, since $CC^{(u)}$ ranges from $[0, 1]$. In this way, we are able to define $\Delta G_i^{(y)}(2)$ as:

$$\Delta G_i^{(y)}(2) = \frac{|CC'^{(y)} - CC^{(y)}|}{\sum_{u \in \mathscr{Y}} |CC'^{(u)} - CC^{(u)}|}. \tag{8.10}$$

where CC^u and CC'^u are clustering coefficients before and after the insertion of the test instance to class component u.

8.4.1.3 Third Network Measure: Average Degree or Connectivity

This measure has also been explored in the chapter related to the fundamentals of Complex Networks (cf. Definitions 2.10 and 2.12). The component connectivity is a relative simple measure, which statistically quantifies the average degree of the vertices of a component. This measure by itself is weak in terms of finding patterns in the network, since the mean value may not exactly quantify the degrees of the majority of vertices in a component. However, if it is jointly used with other measures, its recognition power significantly increases.

The derivation of $\Delta G_i^{(y)}(3)$ is similar to the previous case, except for a simple particularity: for all components $u \in \mathscr{Y}$ that do not share at least one link with test instance x_i, we purposefully assign:

$$\langle k'^{(u)} \rangle = \max \left(\langle k^{(u)} \rangle - \min_j \left(k_j^{(u)} \right), \max_j \left(k_j^{(u)} \right) - \langle k^{(u)} \rangle \right), \tag{8.11}$$

i.e., the maximum possible difference from the mean of the component. In this way, we are able to define $\Delta G_i^{(y)}(3)$ as:

$$\Delta G_i^{(y)}(3) = \frac{|\langle k'^{(y)} \rangle - \langle k^{(y)} \rangle|}{\sum_{u \in \mathscr{Y}} |\langle k'^{(y)} \rangle - \langle k^{(u)} \rangle|}. \tag{8.12}$$

Recall that $\langle k'^{(u)} \rangle$ and $\langle k^{(u)} \rangle$ represent the average degree of the component u before and after the test instance x_i is inserted into the class component u, respectively.

8.4.2 High-Level Classification Using Tourist Walks

Now we study the second approach of high-level classification [28]. Instead of using classical network measures, we use the dynamics generated by several tourist walk processes to extract high-level information from the network constructed from the input data. For the sake of clarity, we retrieve, in a synthetic manner, the main concepts that we utilize in this section.[3]

A tourist walk can be conceptualized as a walker (tourist) aiming at visiting sites (data items) in a P-dimensional map, representing the data set. At each step, the tourist follows a simple deterministic rule: it visits the nearest site that has not been visited in the previous μ steps. In other words, the walker performs partially self-avoiding deterministic walks over the data set, where the self-avoiding factor is limited to the memory window $\mu - 1$. This quantity can be understood as a repulsive force emanating from the sites in this memory window, which prevents the walker from visiting them in this interval (refractory time). Each tourist walk can be decomposed in two terms: (1) the initial *transient part* of length t and (2) a *cycle* (attractor) with period c. Since the tourist walker must respect the network topology, it may get in a dead end with no available neighboring vertex to go. In this case, we say that the cycle length is null. In spite of being a simple rule, it has been shown that this movement dynamic possesses complex behavior when $\mu > 1$ [15]. Moreover, the transient and cycle lengths are dependent on the choice of the memory length μ.

In the previous implementation of the high-level classifier, three different network measures have been employed: connectivity or average degree, clustering coefficient, and assortativity. An immediate question that arises is whether or not this set of selected measures is really sufficient to extract patterns from a network. In addition, in case they are sufficient, how one may come up with other sets of measures to construct new high-level classifiers? Therefore, a serious open problem of the previous approach is how one may choose other network measures in an intuitive way and also how one may define the learning weights that are associated with each of them. For instance, those three network measures have been chosen under a series of trial and error attempts against several well-known network measures. These issues are addressed when tourist walks are employed to construct high-level classifiers. Firstly, a unified measure to capture the patterns formed by the data is presented. In this way, one does not need to discover suitable and convenient sets of network measures to build up the high-level classifier, as occurs in the previous approach. In this new implementation, we show that the dynamical information generated by tourist walks processes is able to extract local-to-global organizational and complex features of the network by adjustments in the walker's memory length parameter. For example, when the memory window of the tourist is small, local structural features of the network are extracted. Conversely, as the

[3] Revisit Sect. 2.4.4 for a comprehensive review on tourist walks.

memory window grows larger, the walker is forced to venture far away from its starting point, permitting it to learn global features of the network. Secondly, the model selection procedure is simplified. As one can see in (8.5), several learning weights of the high-level classifier must be carefully fine-tuned. Because they are in large numbers, the model selection procedure may take a considerable amount of time to complete. In large-scale data sets, therefore, the application of the previous approach would be unfeasible. In this new approach, the learning weights are endogenously adjusted or fit from the training data. For this end, we utilize efficient statistical procedures to adjust the learning weights that run in linear time. As a result, the model selection effort is reduced to a large extent.

In addition to these advantages, the process of tourist walks presents some other interesting characteristics. One of them is the presence of class-dependent critical memory lengths. For a specific class $y \in \mathscr{Y}$, the critical memory length is defined as an emergent point in which larger memory length values make no changes in the behaviors of the transient and cycle lengths. This phenomenon is observed when the memory length assumes sufficient large values. We say that, when this happen, the walks have reached the "complexity saturation" of the class component. In this occasion, the global topological and organizational features of the network are said to be completely characterized in the sense of the tourist walks process. Moreover, this phenomenon can be related to phase transition in the context of complex networks.

Having in mind these concepts, the decision output of the high-level classifier based on tourist walks is given by:

$$H_i^{(y)} = K_H \sum_{\mu=0}^{\mu_c^{(y)}} w_{\text{inter}}^{(y)}(\mu) \left[w_{\text{intra}}^{(y)}(\mu) T_i^{(y)}(\mu) + (1 - w_{\text{intra}}^{(y)}(\mu)) C_i^{(y)}(\mu) \right], \tag{8.13}$$

in which:

- $\mu_c^{(y)}$ is a critical value that indicates the maximum memory length of the tourist walks performed in the training phase for class y;
- $T_i^{(y)}(\mu)$ and $C_i^{(y)}(\mu)$ are functions that depend on the transient and cycle lengths, respectively, of the tourist walk applied to the i-th data item with regard to class y. These functions are responsible for providing an estimate of whether or not the data item i under analysis possesses the same patterns of component y;
- $w_{\text{inter}}^{(y)}(\mu)$ is the weight or influence that is given for the tourist walk with memory length μ on class y. Observe that we have used the subscript *inter* to make clear that this coefficient deals with the regulation of tourist walks with different μ;
- $w_{\text{intra}}^{(y)}(\mu)$ is the weight or influence of the transient length of a particular tourist walk with memory length μ on class y. The complementary value, i.e., $(1 - w_{\text{intra}}^{(y)}(\mu))$, records the same information but for the cycle length. Note that we have used the subscript *intra* to denote that such coefficient is modulating the dynamic generated within the same tourist walk;
- K_H is a normalization constant which ensures the fuzziness of the high-level classifier H.

8.4.2.1 Calculating the Variational Descriptors $T_i^{(y)}(\mu)$ and $C_i^{(y)}(\mu)$

The descriptors that characterize the structural variations on the component representing class $y \in \mathscr{Y}$ due to insertion of the test instance x_i are defined as:

$$\begin{aligned} T_i^{(y)}(\mu) &= 1 - \Delta t_i^{(y)}(\mu) p^{(y)}, \\ C_i^{(y)}(\mu) &= 1 - \Delta c_i^{(y)}(\mu) p^{(y)}, \end{aligned} \tag{8.14}$$

in which $\Delta t_i^{(y)}(u), \Delta c_i^{(y)}(u) \in [0, 1]$ are the variations of the transient and cycle lengths on the component representing class y if test instance i joins it and $p^{(y)} \in [0, 1]$ is the proportion of training data items pertaining to class y.

In order to compute $\Delta t_i^{(y)}(\mu)$ and $\Delta c_i^{(y)}(\mu)$ that appear in (8.14), for a fixed μ, we perform tourist walks initiating from each of the vertices of component $y \in \mathscr{Y}$. In this way, we get the average transient and cycle lengths of component that represents class y, $\langle t^{(y)}(\mu) \rangle$ and $\langle c^{(y)}(\mu) \rangle$, respectively. For the purpose of estimating the variation of the component's network measures, consider that $x_i \in \mathscr{X}_{\text{test}}$ is a test instance. After x_i is inserted into an arbitrary class $y \in \mathscr{Y}$, we recalculate the average transient and cycle lengths of this component, denoted as $\langle t'^{(y)}_i(\mu) \rangle$ and $\langle c'^{(y)}_i(\mu) \rangle$, respectively. This procedure is performed for all classes $y \in \mathscr{Y}$. Again, if some classes $u \in \mathscr{Y}$ do not share any connections with the test instance x_i, we set a high value for $\langle t'^{(y)}_i(\mu) \rangle$ and $\langle c'^{(y)}_i(\mu) \rangle$.

Then, we can calculate $\Delta t_i^{(y)}(\mu)$ and $\Delta c_i^{(y)}(\mu)$, $\forall y \in \mathscr{Y}$, as follows:

$$\begin{aligned} \Delta t_i^{(y)}(\mu) &= \frac{|\langle t'^{(y)}_i(\mu) \rangle - \langle t^{(y)}(\mu) \rangle|}{\sum_{u \in \mathscr{Y}} |\langle t'^{(u)}_i(\mu) \rangle - \langle t^{(u)}(\mu) \rangle|}, \\ \Delta c_i^{(y)}(\mu) &= \frac{|\langle c'^{(y)}_i(\mu) \rangle - \langle c^{(y)}(\mu) \rangle|}{\sum_{u \in \mathscr{Y}} |\langle c'^{(u)}_i(\mu) \rangle - \langle c^{(u)}(\mu) \rangle|}, \end{aligned} \tag{8.15}$$

in which the denominator is just for normalization matters. According to (8.15), large variations of a component's transient and cycle lengths, $\Delta t_i^{(y)}(\mu)$ and $\Delta c_i^{(y)}(\mu)$, yield small membership values $T_i^{(y)}(\mu)$ and $C_i^{(y)}(\mu)$ and small variations produce large membership values.

The memory length μ has a high influence on the classification result. According to (8.13), the above described procedure is performed by varying the memory length μ, ranging from 0 (memoryless) to a critical value μ_c. In this way, the descriptors can capture complex patterns of each of the representative class components in a local to global fashion. When μ is small, the walks tend to possess a small transient and cycle parts, so that the walker does not wander far away from the starting vertex. In this way, the walking mechanism is responsible for capturing the local structures of the class component. On the other hand, when μ increases, the walker

is compelled to venture deep into the component, possibly very far away from its starting vertex. In this case, the walking process is responsible for capturing the global features of the component.

8.4.2.2 Determining the Intra-Modulation Parameters $w^{(y)}_{\text{intra}}(\mu)$

The idea to estimate the parameters $w^{(y)}_{\text{intra}}(\mu)$ is simple. Intuitively, the transient length and cycle length of each tourist walk can be considered as a peculiar or unique vision of the class component. For a fixed μ, if the variation of transient lengths (cycle lengths) is small, all the walks have a homogenous vision on a class component; otherwise, they have heterogeneous visions. In the event of an insertion of a new test instance, it is reliable to state that its impact on the organizational formation of the component is much stronger if its insertion causes a break in the homogenous vision rather than in the heterogeneous vision. That is, a greater influence should be intuitively given for the changes in homogeneous visions in detriment to changes in heterogeneous visions. With this idea in mind, we propose a calibration of $w^{(y)}_{intra}(\mu)$ using the variances of the transient and cycle lengths generated by the training set.

In the following, we formalize this idea. For a fixed μ and $y \in \mathscr{Y}$, let the variances of the tourist walk's transient and cycle lengths be $\sigma_c^{(y)}(\mu)$ and $\sigma_t^{(y)}(\mu)$, respectively. Then, $w^{(y)}_{\text{intra}}(\mu)$, $\forall \mu \in \{0, \ldots, \mu_c\}, y \in \mathscr{Y}$, is given as follows:

$$w^{(y)}_{\text{intra}}(\mu) = \frac{\sigma_c^{(y)}(\mu) + 1}{\sigma_t^{(y)}(\mu) + \sigma_c^{(y)}(\mu) + 2}, \tag{8.16}$$

$$1 - w^{(y)}_{\text{intra}}(\mu) = \frac{\sigma_t^{(y)}(\mu) + 1}{\sigma_t^{(y)}(\mu) + \sigma_c^{(y)}(\mu) + 2}, \tag{8.17}$$

in which Eq. (8.16) refers to the influence of the transient length and (8.17) provides the influence of the cycle length when tourist walks with memory length μ are performed. Note that we have employed the Laplace Smoothing technique into the determination of such parameters, which adds 1 for the variances of each variable. This allows Eqs. (8.16) and (8.17) to be always defined for every $\sigma_c^{(y)}(\mu) \times \sigma_t^{(y)}(\mu) \in \mathbb{R}^2$.

8.4.2.3 Determining the Inter-Modulation Parameters $w^{(y)}_{\text{inter}}(\mu)$

In order to estimate the influence weights of tourist walks with different μ, the same strategy based on variances given in the previous subsection can be applied. i.e., a higher weight value is given to the walks that have less total variance (transient + cycle). This strategy is the same as to declare that we are favoring homogenous visions against heterogeneous visions.

The idea is formalized as follows. For a fixed μ and $y \in \mathscr{Y}$, say that the variances of the transient and cycle lengths of tourist walks with $\mu \in \{0, \ldots, \mu_c\}$ are given by $\sigma_c^{(y)}(\mu)$ and $\sigma_t^{(y)}(\mu)$, respectively. Then, $w_{\text{inter}}^{(y)}(\mu), \mu \in \{0, \ldots, \mu_c\}$, is given as follows:

$$w_{\text{inter}}^{(y)}(\mu) = \frac{\sum_{\Delta=0, \Delta \neq \mu}^{\mu_c^{(y)}} \sigma_t^{(y)}(\Delta) + \sigma_c^{(y)}(\Delta)}{\sum_{\rho=0}^{\mu_c^{(y)}} \sum_{\Delta=0, \Delta \neq \rho}^{\mu_c^{(y)}} \sigma_t^{(y)}(\Delta) + \sigma_c^{(y)}(\Delta)}$$
$$= \frac{k_{\text{inter}}^{(y)} - (\sigma_t^{(y)}(\mu) + \sigma_c^{(y)}(\mu))}{\mu_c^{(y)} k_{\text{inter}}^{(y)}}, \tag{8.18}$$

in which:

$$k_{\text{inter}}^{(y)} = \sum_{\Delta=0}^{\mu_c^{(y)}} \sigma_t^{(y)}(\Delta) + \sigma_c^{(y)}(\Delta). \tag{8.19}$$

Note that, for a fixed μ, if $\sigma_t^{(y)}(\mu) + \sigma_c^{(y)}(\mu)$ is large, then the corresponding influence of that tourist walk, $w_{\text{inter}}^{(y)}(\mu)$, is small in relation to the others $w_{\text{inter}}^{(y)}(\Delta), \Delta \neq \mu$. Conversely, if the sum of the descriptors of the tourist walk with fixed μ is small, we get a large $w_{\text{inter}}^{(y)}(\Delta)$, showing the relative importance of those tourist walks in the learning process.

8.5 Numerical Analysis of the High-Level Classification

In this section, we assess the performance of the high-level classifier composed by the dynamical information generated from the tourist walks. Section 8.5.1 reviews a problematic situation where the high-level of learning is welcomed and Sect. 8.5.2 supplies a parameter sensitivity analysis of the model.

8.5.1 An Illustrative Example

Figure 8.5 shows a segment of line representing the red or circular-shaped class (9 vertices) and also a condensed hollow circular class (torus) depicted by the blue or square-shaped class (1000 vertices). The network formation parameters are fixed as $k = 1$ and $\epsilon = 0.07$[4]. The fuzzy SVM technique [16] with RBF kernel

[4](This radius covers, for any vertex in the straight line, two adjacent vertices, except for the vertices in each end).

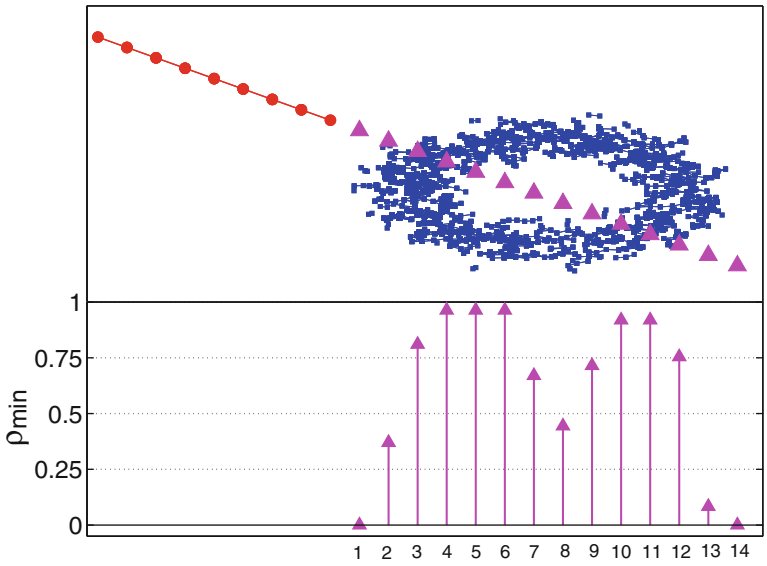

Fig. 8.5 Minimum value of the compliance term, $\rho_{\min}$, that results in the classification of the *purple or triangle*-shaped test instances as members of the *red or circular*-shaped class. Reproduced from [28] with permission from Elsevier

($C = 2^2$ and $\gamma = 2^{-1}$) is employed as the traditional low-level classifier. The task is to classify the 14 test instances represented by the big triangle-shaped items from left to right. After a test instance is classified, it is incorporated to the training set with the corresponding predicted label (self-learning). The low part of the figure shows the minimum required value of $\rho_{\min}$ for which the triangle-shaped items are classified as members of the red or circular-shaped class. From this figure, we see that the big triangle-shaped items can be identified to form a line pattern with the red or circular elements, even if the straight line crosses the condensed region of the blue or square-shaped class. Another feature is that the required compliance term takes small values when the test instances stay long away from the blue or square-shaped class (simple cases of classification) but it takes large values when the straight line crosses the "torus" class (complex cases of classification). This means that the high-order of learning is very useful in complex situations of classification.

8.5.2 Parameter Sensitivity Analysis

In this section, several simulations are performed to show the influence of the model's parameters. We only provide a parameter sensitivity analysis of the high-level classifier that is based on tourist walk processes, because this approach self-adjusts the learning of the network descriptors. Therefore, model selection

procedures become viable. In contrast, the model based on complex network measures has as many learning weights as there are complex network measures.

8.5.2.1 Influence of the Parameters Related to the Network Formation

The network formation step plays a crucial role in the learning process of the high-level classification. It is regulated by the parameters k and ϵ. For every data item (vertex), the algorithm determines if it is located in a sparse or dense region by counting the number of data items within a hyper-sphere with radius ϵ. If this number is smaller than k, the vertex is declared to be in a sparse region and the k-NN is used. Otherwise, the ϵ-radius is employed. If k is bounded by the number of training data items V, some mathematical consequences of this procedure are given below:

- If $k \to V \Rightarrow$ all vertices are declared to be in sparse regions $\Rightarrow$ always use the k-NN technique, regardless of ϵ;
- If $k \to 0 \Rightarrow$ all vertices are declared to be in dense regions $\Rightarrow$ always use the ϵ-radius technique;
- If $\epsilon \to \infty \Rightarrow$ all vertices are declared to be in dense regions $\Rightarrow$ always use the ϵ-radius technique, regardless of k;
- If $\epsilon \to 0 \Rightarrow$ all vertices are declared to be in sparse regions $\Rightarrow$ always use the k-NN technique.

The aforementioned behaviors imply that only for intermediate values of k and ϵ, both the k-NN and ϵ-radius techniques are enabled. This is because, if k is too large, the ϵ-radius technique is disabled. Conversely, if ϵ is too large, the k-NN technique is turned off.

8.5.2.2 Influence of the Critical Memory Length

In Ref. [28], the authors uncovered a phenomenon regarding the critical memory length $\mu_c^{(y)}$, called *complexity saturation*. In order to facilitate the understanding of such property, let us first investigate some simulation results on synthetic and real-world data sets.

For the experiments, we consider the Wine data set (unbalanced classes), which is a well-known data set from the UCI Machine Learning Repository [8]. Figure 8.6a and b portrays the transient and cycle lengths computed for each class reported in the Wine data set. With respect to the transient length behavior, we can see that the transient length increases as μ increases. However, when μ is sufficiently large, the components' transient lengths settle down in a flat region. In contrast, the cycle length behavior shows an interesting behavior, which can be roughly divided in three different regions: (1) for a small μ, the cycle length is directly proportional to μ; (2) for intermediate values of μ, the cycle length is inversely proportional to μ; and

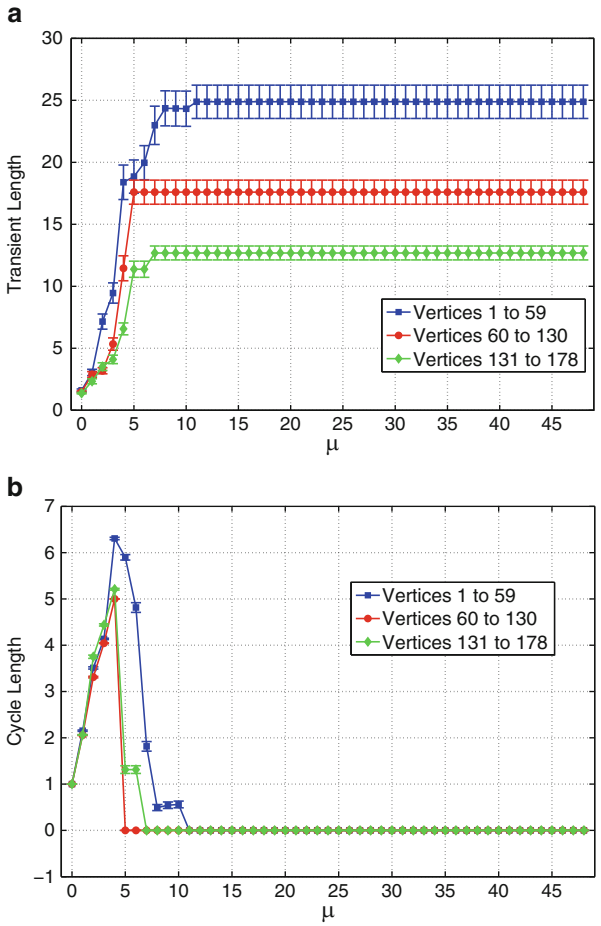

Fig. 8.6 Behavior of the transient and cycle lengths of the Wine data set. Network formation parameters: $k = 3$ and $\epsilon = 0.04$. Reproduced from [28] with permission from Elsevier. (**a**) Transient length vs. μ_c and (**b**) cycle length vs. μ_c

(3) for sufficiently large values of μ, the cycle length also settles down in a steady region . One can interpret these results as follows:

- When μ is small, it is very likely that the transient and cycle parts are also small, because the memory of the tourist is very limited. We can conceive this as a walk with almost no restrictions;
- When μ assumes an intermediate value, the transient length keeps increasing but the cycle length reaches a peak and starts to decrease afterwards. This peak characterizes the topological complexity of the component and varies from one to another. Hence, this is the most important region for capturing pattern formation of the class component by using the network topological structure;

- When μ is large, the tourist has a greater chance of getting trapped in a vertex of the graph. This happens when the entire neighborhood of the visited vertex is contained within the memory window μ. In this scenario, the transient length is expected to be very high and the cycle length, null. This phenomenon explains the steady regions in Fig. 8.6a and b. In this region, the tourist walks have already covered all the global aspects of the class component, and increasing the memory length μ will not capture any new topological features or pattern formation of the class components. In this scenario, it is said that the tourist walks have completely described the topological complexity of the class component (saturation). In view of this, the calculation of tourist walks by further increasing μ is redundant.

This analysis suggests that the accuracy of the high-level classification may not change given that we choose suitable $\mu_c^{(y)}, y \in \mathscr{Y}$, residing near these steady regions for each class in the problem. This means that larger values for $\mu_c^{(y)}$ only cause redundant computations as the accuracy rate does not enhance nor reduce.

This phenomenon of complexity saturation observed in these data sets can also be related to phase transition in networks. For example, when $\mu < \mu_c$, we can conceive the tourist walks in the network to be in an exploratory phase, where the dynamics of the transient and cycle lengths change as the parameter μ is modified. Therefore, in this initial exploratory phase, parameter μ is sensitive to the outcome of the tourist walks' dynamics. However, when $\mu = \mu_c$, the walk changes from the exploratory to the stationary phase, in which the lengths of the transient and cycle parts of the tourist walks performed on networks are not sensitive (independent) anymore. This holds true for all $\mu \geq \mu_c$. In this phase, the graph topology restrains the tourist walk such as to not change its dynamical information anymore. As the network becomes more dense, more different walks are probabilistic possible, and μ_c is expected to take on larger values. In this regard, in a complete graph, the μ_c of a networked topology would be exactly equal to a networkless (lattice) approach.

Based on these experiments, a heuristic for estimating the critical memory length $\mu_c^{(y)}$ is provided as follows. For a particular class, the dynamics of the tourist walks are calculated starting from $\mu = 0$. Once finished, μ is incremented and the same calculations are performed for the new μ. Say that $t^{(y)}(\mu)$ and $c^{(y)}(\mu)$ are the curves drawn from these calculations for the transient and cycle lengths, respectively. Once the derivatives of $t^{(y)}(\mu)$ and $c^{(y)}(\mu)$, i.e., $t'^{(y)}(\mu)$ and $c'^{(y)}(\mu)$ are zero, we store the $\mu_c^{(y)}$ in which this happened and start out a counter, which monitors how many iterations of μ the derivatives of these measures have not changed. This counter is incremented as μ increases. If $t'^{(y)}(\mu)$ and $c'^{(y)}(\mu)$ remain zero-valued in few iterations, the learning process is stopped and all calculations for which $\mu > \mu_c^{(y)}$ are discarded.

8.6 Application: Handwritten Digits Recognition

In this section, we show the appealing feature of high-level classification through a real pattern recognition application—handwritten digits recognition. We focus on the performance of the high-level classification that relies on a linear combination of tourist walks because the model selection procedure is simpler.

Section 8.6.1 motivates the importance of handwritten recognition in real-world applications and the challenges involved in this process. Section 8.6.2 describes the data set composed of handwritten digits that is employed in the automated recognition task. Section 8.6.3 presents a suitable image-based similarity measure that we use when constructing the training network. Section 8.6.4 lists a small set of low-level classification techniques that are plugged into the hybrid classification framework to test its robustness. Section 8.6.5 reports the results of the hybrid classifier that comprises a suitable combination of the low-level and the high-level classification. Section 8.6.6 illustrates how the training network of handwritten digits is and also shows how the high-level classifier can really help in classifying digits of a real-world data set.

8.6.1 Motivation

Handwritten recognition is the ability of computers to receive and interpret intelligible handwritten input from sources such as paper documents, photographs, touchscreens, data sets, and other devices [17, 30]. Ideally, the handwriting recognition systems should be able to read and understand any handwriting [3]. Handwriting recognition has been one of the most fascinating and challenging research areas in the field of image processing and pattern recognition in the recent years [22]. It contributes immensely to the advancement of an automation process and can improve the interface between man and machine in numerous applications [3, 30]. In general, handwritten recognition is classified into off-line or on-line. In the first case, the writing is obtained by an electronic device and the captured writing is completely available as an image to the handwritten recognition method. In the second case, the coordinates of successive points are available by means of a function dependent on time, i.e., the complete image is not given [22, 30]. In summary, several research works have been proposed [11, 21, 22] in an attempt to reduce the processing time of both off-line and on-line methods, while, at the same time, providing higher recognition accuracy. Due to the high complexity that this topic offers, it still has a wide range of problems to be addressed, such as the efficient recognition of images that are distorted or suffered a nonlinear transformation [3, 6, 30]. In view of these complexities, we attempt to utilize complex networks to help in the task of handwritten digits and letters recognition by taking advantage of the topological characteristics of the constructed network of patterns.

8.6.2 Description of the MNIST Data Set

Handwriting digits recognition is a well accepted benchmark for comparing pattern recognition methods. Here, the Modified NIST (National Institute of Standards and Technology) data set [14], MNIST for short, is used. It was created by "re-mixing" the samples from NIST's original data sets. While the NIST's training data set was taken from American Census Bureau employees, the test data set was taken from American high school students. In view of the different data distributions of the training and test sets, the NIST's complete data set was considered too hard.

The database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training data set, while the other half of the training set and the other half of the test set were taken from NIST's testing data set. This data set is almost balanced with regard to the size of the ten existing classes, each of which representing a digit. Similarly to [14], a pre-processing step is conducted. In this respect, the gray-level images (samples) are reduced to fit in a 20×20 pixel box, while preserving their aspect ratio.

8.6.3 A Suitable Similarity Measure for Images

In a network-based data representation, the images (data items) are represented by the vertices, while the relationships between them are given by the links. A link connecting two vertices (images) holds a weight that numerically translates the similarity between them. Each image can be represented by a "square" matrix $\eta \times \eta$. For rectangle images, a pre-processing is required to transform it into a square image. We conventionally set the pixels' values range to lie within the interval $[0, 1]$ by normalization. Thus, an arbitrary data item (image) x_i can be seen as a matrix with dimensions $\eta \times \eta$, where each pixel $x_i^{(u,j)} \in [0, 1], \forall (u, j) \in \{1, \ldots, \eta\} \times \{1, \ldots, \eta\}$.

In order to construct the network, we are required to establish a similarity measure. The traditional pixel-per-pixel distance is rather insufficient in terms of reliably representing data, since such measure is very sensitive to rotations and scale modifications. With the purpose of overcoming this difficulty, we propose a measure based on the eigenvalues that each image inherently carries with it. First of all, we remove the mean associated to each data item (image), so that we have a common basis of comparison. After that, we calculate the ϕ greatest eigenvalues of the image. Efficient methods have been developed for finding the leading eigenvalues of real-valued asymmetric matrices [10, 33]. The magnitudes of the eigenvalues are related to the variations that the image possesses; hence, it is a natural carrier of information [13]. The greater its value, more information about the image it conveys. By virtue of that, a good choice is to only extract the greatest $\phi < \eta$ eigenvalues and drop the smaller values, since these do not transport too much information about the

image. Also, in order to give more emphasis to the largest eigenvalues, a weight is associated to each one so that the larger an eigenvalue is, the larger is its associated weight.

Consider that we are to compare the similarity between two images, say x_i and x_j, in relation to the ϕ largest eigenvalues. We firstly sort the ϕ eigenvalues of each image as: $|\lambda_i^{(1)}| \geq |\lambda_i^{(2)}| \geq \ldots \geq |\lambda_i^{(\phi)}|$ and $|\lambda_j^{(1)}| \geq |\lambda_j^{(2)}| \geq \ldots \geq |\lambda_j^{(\phi)}|$, where $|\lambda_i^{(k)}|$ marks the k-th eigenvalue of the i-th data item. In this case, the dissimilarity $d(i,j)$ (or, equivalently, the similarity $s(i,j) = 1 - d(i,j)$) between image i and j is given by:

$$d(i,j) = \frac{1}{\rho_{\max}} \sum_{k=1}^{\phi} \beta(k) \left[|\lambda_i^{(k)}| - |\lambda_j^{(k)}| \right]^2, \tag{8.20}$$

in which $\rho \in [0,1]$, $\rho_{\max} > 0$ is a normalization constant, $\beta : \mathbb{N}^* \rightarrow (0,\infty)$ indicates a monotonically decreasing function that can be arbitrarily chosen by the user.

8.6.4 Configurations of the Low-Level Classification Techniques

Three low-level classification techniques are going to be used in the following computer simulations. For more details about the setup and architectural characteristics of the first two techniques, one can refer to [14].

- *A Perceptron neural network*: each input pixel value contributes to a weighted sum for each output neuron. The output neuron with the highest sum (including the contribution by virtue of the bias applied to that neuron) marks the class of the input character.
- *A k-nearest neighbors classifier*: we set the similarity as the reciprocal of the Euclidean distance and $k = 3$.
- *A network-based ϵ-radius classifier*: the ϵ-radius network formation technique is employed with a dissimilarity measure given by a weighted sum of the $\phi = 4$ greatest eigenvalues, as described in the previous section.

8.6.5 Experimental Results

Figure 8.7 shows the performance of the three low-level techniques acting together with the high-level classification in a networked environment. For example, the Perceptron alone reaches 88 % of accuracy rate, while a small increase in the compliance term is able to increase the overall model's accuracy rate to 91 %

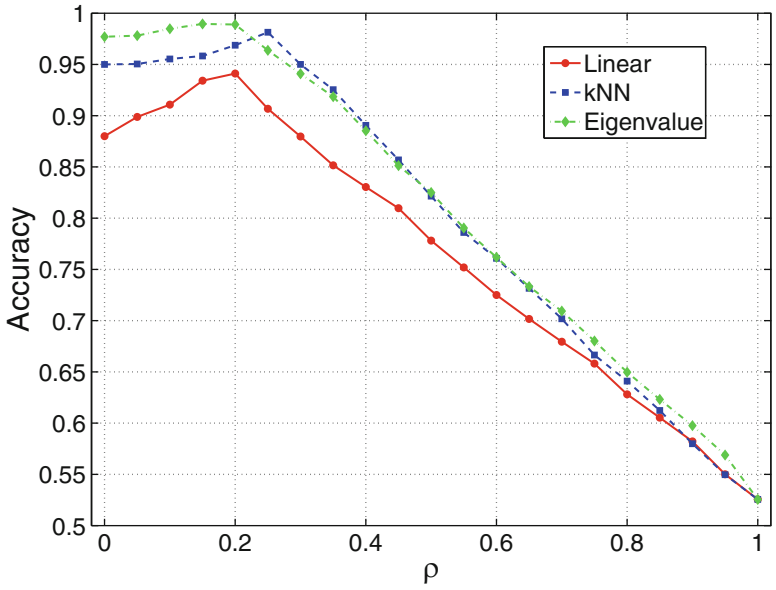

Fig. 8.7 A detailed analysis of the impact of the compliance term ρ on different traditional low-level techniques applied to the MNIST database. One can see that a mixture of traditional and high-level techniques does give a boost in the accuracy rate in this real-world data set. Reproduced from [28] with permission from Elsevier

($\rho = 0.2$). Regarding the k-nearest neighbor algorithm, for a pure traditional classifier, we obtain 95 % of accuracy rate, against 97.6 % when $\rho = 0.25$. For the weighted eigenvalue measure, we obtain 98 % of accuracy rate when $\rho = 0$, against 99.1 % when $\rho = 0.2$. It is worth noting that these enhancements are significant. Even in the third case, the improvement is quite welcomed, because it is a hard task to increase an already very high accuracy rate.

8.6.6 Illustrative Examples: High-Level Classification vs. Low-Level Classification

In this section, we provide illustrative examples to show the situations where the high-level classification works but the low-level term fails.

For simplification matters, we consider only two classes: digits "5" and "6". Figure 8.8a and b illustrates how the digit classification is carried out by using simple networks containing these small samples of digits.

Firstly, let us consider Fig. 8.8a, where the digits "5" and "6" surrounded by brown and blue boxes, respectively, represent the training set. The task is to classify the test instance represented by the digit in the red box. If only the low-level

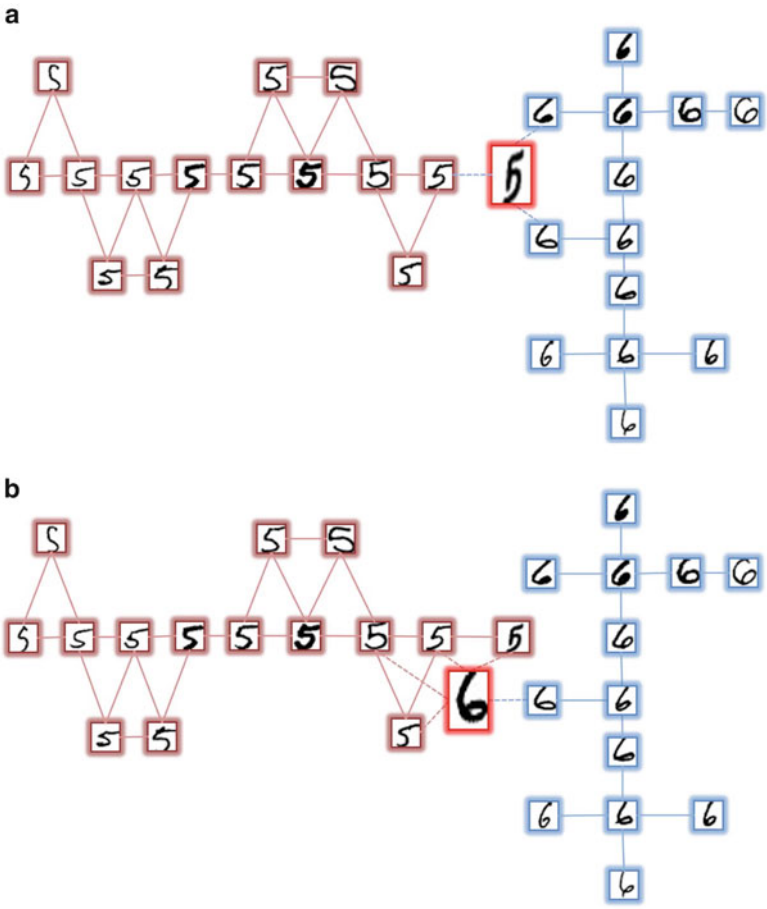

Fig. 8.8 Illustration of the pattern formation impact in a subset of samples extracted from the MNIST data set. The training instances are displayed by the *brown* (digit 5) and *blue* (digit 6) colors. The test instances are indicated by a *red color* (bigger sizes). Reproduced from [28] with permission from Elsevier. (**a**) Insertion of a digit 5 test instance and (**b**) insertion of a digit 6 test instance

classification is applied, the test digit will probably be classified as a digit "6", because there are more neighbors of digit "6" than that of "5" in the vicinity. On the other hand, if we also consider the class' geometrical disposition (high-level classification), it is more suggestive that the referred test instance is a member of the digit "5" class, because it complies more to the pattern formed by training digits of the class "5" than to the one formed by the digits of the class "6". In organizational terms, if the test digit is inserted into the class "5" as displayed, it will just extend the somewhat formed horizontal "line" pattern. As a consequence, the inclusion of the test digit in this class will disturb (change) the class organization in a small

extent, i.e., its representative descriptors (transient and cycle length) will not vary significantly. However, if the test digit is inserted into the class "6", larger variations of the component measures will occur, since cycles are formed in the component. Taking into consideration that, before the insertion of the test instance, there were no cycles in the components, it is clear that the representative descriptors of the component representing the class "6" will vary considerably by virtue of this abrupt change.

Figure 8.9a and b exhibits the transient and cycle lengths, as well as their corresponding variations, as a function of μ, when the digit "5" test instance is inserted into the component represent the digit "5" cluster. As we expected, we see that the variations are very small in the class representing the digit "5", indicating and suggesting the strong compliance of the digit "5" test instance with the pattern

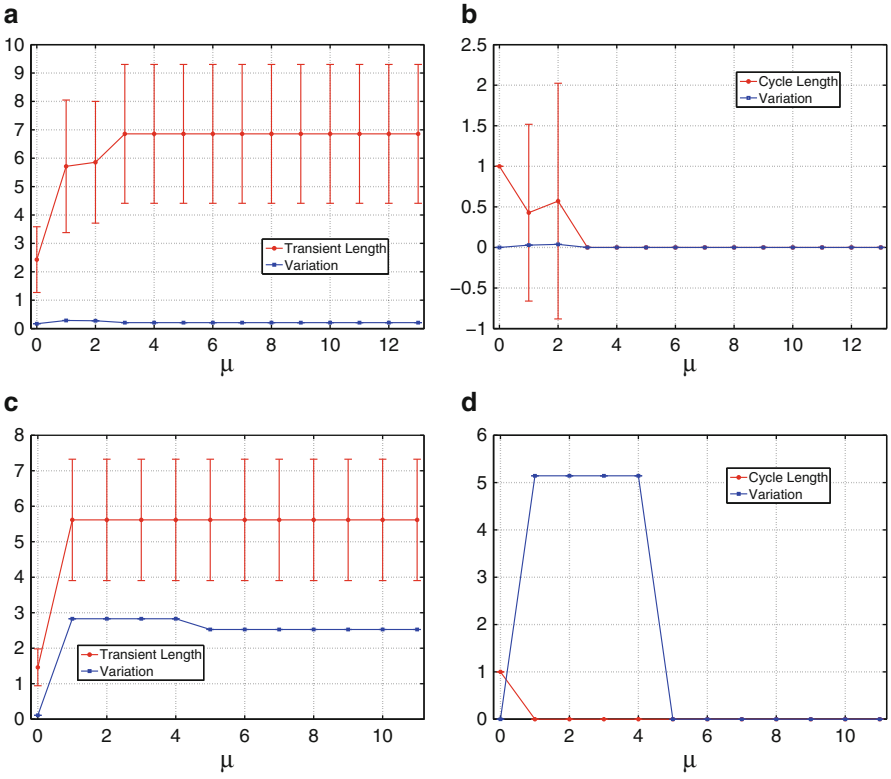

Fig. 8.9 Transient and cycle lengths of the graph components representing the training instances of the digits 5 (*brown class*) and 6 (*blue class*), as shown in Fig. 8.8a. In addition, the variation on these two measurements are reported due to the insertion of the digit 5 test instance (*red instance*). Reproduced from [28] with permission from Elsevier. (**a**) *Brown class* transient length, (**b**) *brown class* cycle length, (**c**) *blue class* transient length and (**d**) *blue class* cycle length

already formed by the representative graph component of the digit "5". On the other hand, Fig. 8.9c and d show the variations of the transient and cycle lengths of the component representing the digit "6" cluster when the digit "5" test instance is inserted. Here, we see that larger variations occur, which means that the digit "5" test instance does not comply with the pattern formed by the representative component of the digit "6" cluster.

Putting together these two observations, we conclude that the high-level classification will correctly classify the test instance as a digit "5". The same reasoning can be applied to the digit network shown in Fig. 8.8b. In this case, the transient and cycle lengths as well as the corresponding variations are shown in Fig. 8.10a–d, when the digit "6" test instance is inserted into the component of digit "5" or "6" cluster, respectively. Using the same arguments and aforementioned plots, in this situation, we can verify that the digit "6" test instance is correctly classified as a digit "6".

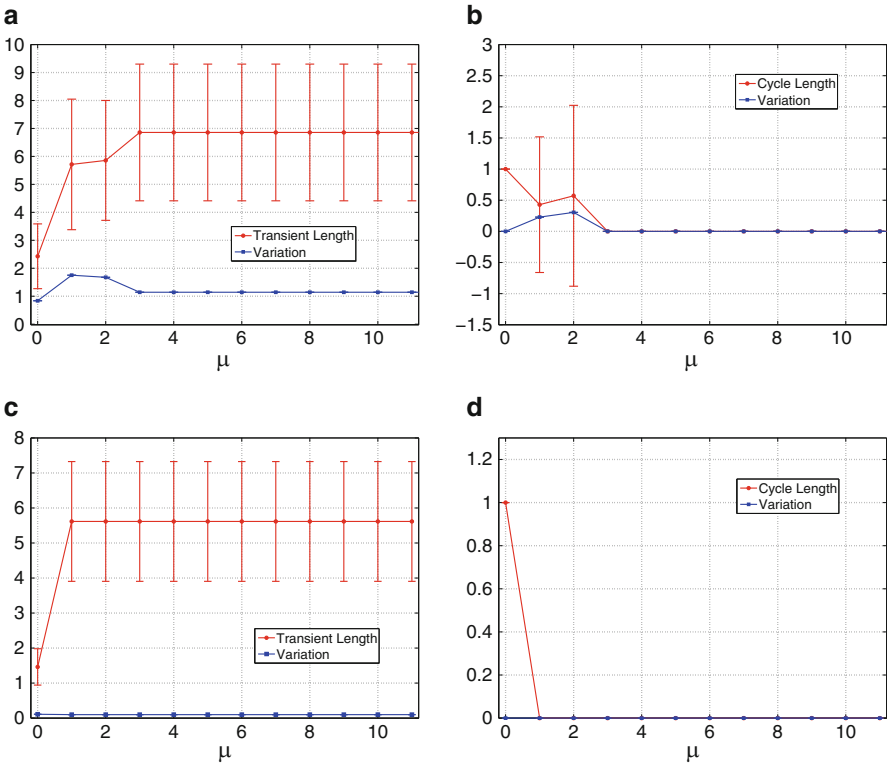

Fig. 8.10 Transient and cycle lengths of the graph components representing the training instances of the digits 5 (*brown class*) and 6 (*blue class*), as shown in Fig. 8.8b. In addition, the variation on these two measurements are reported due to the insertion of the digit 6 test instance (*red instance*). Reproduced from [28] with permission from Elsevier. (**a**) *Brown class* transient length, (**b**) *brown class* cycle length, (**c**) *blue class* transient length and (**d**) *blue class* cycle length

8.7 Chapter Remarks

In this chapter, we have studied a general classification framework that is composed of a novel combination of low- and high-level classifiers. The low-level term classifies test instances according to their physical features, while the second term measures how well new test instances comply with the existing patterns formed by the data relationships. This is performed by exploiting the complex topological properties of the network built from the training data.

In addition to the novel definition of this hybrid classification, two implementations for the high-level term are reviewed, both running in a networked environment. In the first one, the high-level classification is composed of three complex network measures: the average degree, clustering coefficient, and assortativity. In the second implementation, the complex dynamics generated by several tourist walks processes are employed. Specifically, the model is characterized by linear weighted combinations of transient and cycle lengths of different tourist walks. It stores tourist walks with varying values (up to a critical value) for the memory parameter. The motivation behind taking combinations of tourist walks with different memory values is that they can capture local (small memory values) to global (large memory values) aspects of the network.

Several experiments are conducted on synthetic and real-world data sets, so that we can better assess the performance of the hybrid classification framework. A quite interesting feature of this technique is that the influence of the high-level term has to be increased as the complexity of the classes increases. This suggests that the high-level term is specially useful in complex situations of classification.

The high-level classification techniques have been applied to handwritten digits recognition and we have seen that the hybrid model can really improve the accuracy rates of traditional classification techniques in certain conditions. It is worth noting that the employment of the high-level term in isolation generally does not perform very well. However, when utilized together with a suitable low-level classification technique, it can really boost the performance of the overall classification procedure.

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Sci. Am. **284**(5), 34–43 (2001)
2. Binaghi, E., Gallo, I., Pepe, M.: A cognitive pyramid for contextual classification of remote sensing images. IEEE Trans. Geosci. Remote Sens. **41**(12), 2906–2922 (2003)
3. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer, New York (2007)
4. Blum, A., Mitchell, T.: Combining labeled and unlabeled data with co-training. In: Proceedings of the 11th Annual Conference on Computational Learning Theory, pp. 92–100 (1998)
5. Chapelle, O., Schölkopf, B., Zien, A. (eds.): Semi-Supervised Learning. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA (2006)
6. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley, New York, NY (2001)

7. Feigenbaum, L., Herman, I., Hongsermeier, T., Neumann, E., Stephens, S.: The semantic web in action. Sci. Am. **297**(6), 90–97 (2007)
8. Lichman, M., UCI machine learning repository, University of California, Irvine, School of Information and Computer Sciences (2013)
9. Gallagher, B., Tong, H., Eliassi-rad, T., Faloutsos, C.: Using ghost edges for classification in sparsely labeled networks. In: Knowledge Discovery and Data Mining, pp. 256–264 (2008)
10. Goldhirsch, I., Orszag, S.A., Maulik, B.K.: An efficient method for computing leading eigenvalues and eigenvectors of large asymmetric matrices. J. Sci. Comput. **2**, 33–58 (1987)
11. Govindan, V.K., Shivaprasad, A.P.: Character recognition: a review. Pattern Recogn. **23**, 671–683 (1990)
12. Haykin, S.S.: Neural Networks and Learning Machines. Prentice Hall, Englewood Cliffs, NJ (2008)
13. Jolliffe, I.T.: Principal Component Analysis. Springer Series in Statistics, New York (2002)
14. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
15. Lima, G.F., Martinez, A.S., Kinouchi, O.: Deterministic walks in random media. Phys. Rev. Lett. **87**(1), 010603 (2001)
16. Lin, C.F., Wang, S.D.: Fuzzy support vector machines. IEEE Trans. Neural Netw. **13**, 464–471 (2002)
17. Liu, C.L., Sako, H., Fujisawa, H.: Performance evaluation of pattern classifiers for handwritten character recognition. IJDAR **4**, 191–204 (2002)
18. Lu, D., Weng, Q.: Survey of image classification methods and techniques for improving classification performance. Int. J. Remote Sens. **28**(5), 823–870 (2007)
19. Macskassy, S.A., Provost, F.: Classification in networked data: a toolkit and a univariate case study. J. Mach. Learn. Res. **8**, 935–983 (2007)
20. Micheli, A.: Neural network for graphs: a contextual constructive approach. IEEE Trans. Neural Netw. **20**, 498–511(3) (2009)
21. Mori, S., Suen, C.Y., Yamamoto, K.: Historical review of OCR research and development. Proc. IEEE **80**, 1029–1058 (1992)
22. Pradeep, J., Srinivasan, E., Himavathi, S.: Diagonal based feature extraction for handwritten alphabets recognition system using neural network. Int. J. Comput. Sci. Inf. Technol. **3**, 27–38 (2011)
23. Shadbolt, N., Berners-Lee, T., Hall, W.: The semantic web revisited. IEEE Intell. Syst. **6**, 96–101 (2006)
24. Silva, T.C., Zhao, L.: Network-based high level data classification. IEEE Trans. Neural Netw. Learn. Syst. **23**(6), 954–970 (2012)
25. Silva, T.C., Zhao, L.: Network-based stochastic semisupervised learning. IEEE Trans. Neural Netw. Learn. Syst. **23**(3), 451–466 (2012)
26. Silva, T.C., Zhao, L.: Semi-supervised learning guided by the modularity measure in complex networks. Neurocomputing **78**(1), 30–37 (2012)
27. Silva, T.C., Zhao, L.: Stochastic competitive learning in complex networks. IEEE Trans. Neural Netw. Learn. Syst. **23**(3), 385–398 (2012)
28. Silva, T.C., Zhao, L.: High-level pattern-based classification via tourist walks in networks. Inform. Sci. **294**(0), 109–126 (2015). Innovative Applications of Artificial Neural Networks in Engineering
29. Skolidis, G., Sanguinetti, G.: Bayesian multitask classification with gaussian process priors. IEEE Trans. Neural Netw. **22**(12), 2011–2021 (2011)
30. Theodoridis, S., Koutroumbas, K.: Pattern Recognition. Academic, London (2008)
31. Tian, B., Azimi-Sadjadi, M.R., Haar, T.H.V., Reinke, D.: Temporal updating scheme for probabilistic neural network with application to satellite cloud classification. IEEE Trans. Neural Netw. **11**(4), 903–920 (2000)

32. Tian, Y., Yang, Q., Huang, T., Ling, C.X., Gao, W.: Learning contextual dependency network models for link-based classification. IEEE Trans. Data Knowl. Eng. **18**(11), 1482–1496 (2006)
33. Tsai, S.H., Lee, C.Y., Wu, Y.K.: Efficient calculation of critical eigenvalues in large power systems using the real variant of the Jacobi-Davidson QR method. IET Gener. Transm. Distrib. **4**, 467–478 (2010)
34. Tuia, D., Camps-Valls, G., Matasci, G., Kanevski, M.: Learning relevant image features with multiple-kernel classification. IEEE Trans. Geosci. Remote Sens. **48**(10), 3780–3791 (2010)
35. Williams, D., Liao, X., Xue, Y., Carin, L.: On classification with incomplete data. IEEE Trans. Pattern Anal. Mach. Intell. **29**(3), 427–436 (2007)
36. Zhang, D., Mao, R.: Classifying networked entities with modularity kernels. In: International Conference on Information and Knowledge Management, pp. 113–122 (2008)
37. Zhang, H., Liu, J., Ma, D., Wang, Z.: Data-core-based fuzzy min-max neural network for pattern classification. IEEE Trans. Neural Netw. **22**(12), 2339–2352 (2011)
38. Zhang, T., Popescul, A., Dom, B.: Linear prediction models with graph regularization for web-page categorization. In: Conference on Knowledge Discovery and Data Mining, pp. 821–826. Association for Computing Machinery, New York (2006)
39. Zhu, X.: Semi-supervised learning literature survey. Tech. Rep. 1530, Computer Sciences, University of Wisconsin-Madison (2005)
40. Zhu, S., Yu, K., Chi, Y., Gong, Y.: Combining content and link for classification using matrix factorization. In: Special Interest Group on Information Retrieval, pp. 487–494. Association for Computing Machinery, New York (2007)