o guarantee that nearby locations are predicted into similar classes, tree node tests nust be extended to add spatial constraints.

Future work can be pursued to improve the method. First, we can test the sensiivity of other parameters and conduct accuracy evaluation on multiple datasets. We nay also compare our spatial decision tree learning algorithm with other relevant echniques in geographical classification, e.g., spatial predictive clustering tree [15], GEOBIA (geographical object-based image analysis) [16], preprocessing, and post-)rocessing.

References

1. M.A. Friedl, C.E. Brodley, Decision tree classification of land cover from remotely sensed data. Remote Sens. Environ. **61**(3), 399–409 (1997)
2. A. Akselrod-Ballin, M. Galun, R. Basri, A. Brandt, M. Gomori, M. Filippi, P. Valsasina, An integrated segmentation and classification approach applied to multiple sclerosis analysis, in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1 (IEEE, 2006), pp. 1122–1129
3. M. Celebi, H. Kingravi, Y. Aslandogan, W. Stoecker, Detection of blue-white veil areas in dermoscopy images using machine learning techniques, in *Proceedings of SPIE Vol*, vol. 6144 (Citeseer, 2006), pp. 61445T–1
4. R. Brooks, D. Wardrop, C. Cole, Inventorying and monitoring wetland condition and restoration potential on a watershed basis with examples from spring creek watershed, pennsylvania, USA. Environ. Manag. **38**(4), 673–687 (2006)
5. A. Deschamps, D. Greenlee, T. Pultz, R. Saper, Geospatial data integration for applications in flood prediction and management in the red river basin, in *International Geoscience and Remote Sensing Symposium, Toronto, Canada. Symposium, Geomatics in the Era of RADARSAT (GER'97)* (Ottawa, Canada, 2002)
6. R. Hearne, Evolving water management institutions in the red river basin. Environ. Manag. **40**(6), 842–852 (2007) (Springer)
7. B. Walsh, How wetlands worsen climate change, http://www.time.com/time/health/article/0,8599,1953751,00.html (2010)
8. J. Quinlan, Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986) (Springer)
9. J.R. Quinlan, *C4.5: Programs for Machine Learning* (Morgan Kaufmann, San Mateo, 1993)
0. B. Ripley, Classification and regression trees, in *R Package Version* (2005), p. 1
1. Z. Jiang, S. Shekhar, P. Mohan, J. Knight, J. Corcoran, Learning spatial decision tree for geographical classification: a summary of results, in *SIGSPATIAL/GIS* (ACM, 2012), pp. 390–393
2. L. Anselin, Local indicators of spatial association–lisa. Geograph. Anal. **27**(2), 93–115 (1995)
3. X. Li, C. Claramunt, A spatial Entropy-Based decision tree for classification of geographical information, in *Transactions in GIS*, vol. 10(3) (Blackwell Publishing Ltd, 2006), pp. 451–467
4. O. Schabenberger, C. Gotway, in *Statistical Methods for Spatial Data Analysis*, vol. 64 (CRC Press, 2005)
5. D. Stojanova, M. Ceci, A. Appice, D. Malerba, S. Džeroski, Global and local spatial autocorrelation in predictive clustering trees, in *Discovery Science* (Springer, Berlin, 2011), pp. 307–322
6. G. Hay, G. Castilla, Geographic object-based image analysis (GEOBIA): a new name for a new discipline, in *Object-Based Image Analysis* (Springer, Berlin, 2008), pp. 75–89

Chapter 5
Focal-Test-Based Spatial Decision Tree

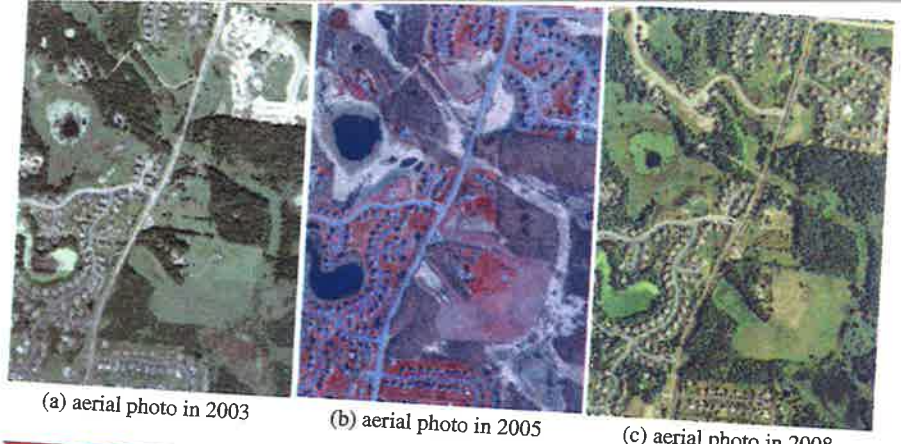
Abstract This chapter introduces another spatial classification technique called focal-test-based spatial decision tree (FTSDT), in which the tree traversal direction of a sample is based on both local information and focal (neighborhood) information. We also provide comparisons of FTSDT with existing decision trees and spatial decision trees on real-world wetland mapping data.

5.1 Introduction

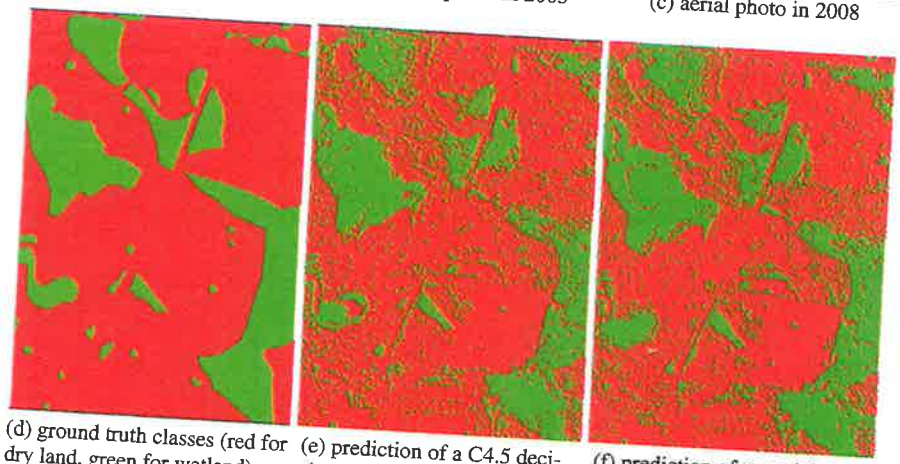
Given a spatial raster framework, as well as training and test sets, the spatial decision tree learning (SDTL) problem aims to find a decision tree model that minimizes classification errors as well as salt-and-pepper noise. Figure 5.1 is a motivation example from a real-world wetland mapping application. Input features are bands of three aerial photographs (Fig. 5.1a–c). Classification results by two existing decision tree classifiers [1, 2] are shown in Fig. 5.1e, f, respectively. Both predicted maps exhibit poor appearance accuracy with high levels of salt-and-pepper noise, when compared with ground truth classes (Fig. 5.1d).

Societal Applications: The SDTL problem has many applications. In the field of remote sensing, a large amount of images of the earth surface are collected (e.g., NASA collects about 5TB data per day). SDTL can be used to classify remote sensing images into different land cover types [3]. For example, in wetland mapping [4, 5], explanatory features, including spectral bands (e.g., red, green, blue, near-infrared) from remote sensors, are used to map land surface into wetland areas and dryland areas. Land cover classification is important for climate change research [6], natural resource management [7, 8], and disaster management [9]. In medical image processing, SDTL can help in lesion classification and brain tissue segmentation [10, 11] on MRI images. It can also be used for galaxy classification [12] in astronomy and semiconductor inspection [13] in materials science.

Challenges: A key challenge in the SDTL problem is that learning samples show spatial autocorrelation in class labels. For example, the ground truth class labels in Fig. 5.1d show strong spatial autocorrelation due to the phenomenon of "patches" (i.e., regions of the same class tend to be contiguous). Testing only local feature information in decision nodes results in salt-and-pepper noise, i.e., locations or pix-



(a) aerial photo in 2003 (b) aerial photo in 2005 (c) aerial photo in 2008



(d) ground truth classes (red for dry land, green for wetland) (e) prediction of a C4.5 decision tree (f) prediction of a spatial-information-gain-based decision tree

Fig. 5.1 Real-world problem example

els whose class labels are different from those of their neighbors, as illustrated in Fig. 5.1e. However, incorporating focal (i.e., neighborhood) information increases both the number and the complexity of candidate tree node tests. Instead of simple linear scanning and thresholding on one-dimensional feature values, tree node tests must incorporate the spatial relationships of various neighborhood sizes. Thus, SDTL problem is also computationally challenging.

Related work and limitations: Figure 5.2 presents a classification of related work. Traditional decision tree algorithms include ID3 [14], C4.5 [1], and CART [15]). These classifiers follow the classic assumption that learning samples are independently and identically distributed. This assumption does not hold for spatial data and leads to salt-and-pepper noise in predictions. A second category is the spatial entropy or information gain-based decision tree classifiers [2, 16–18]. These newer methods

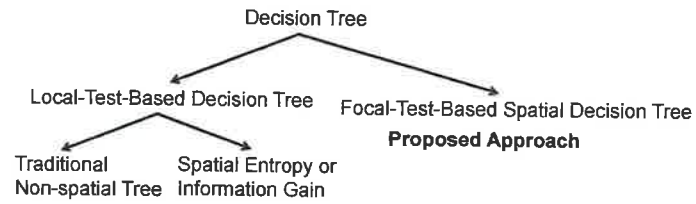


Fig. 5.2 Related work classification

use spatial autocorrelation level as well as information gain to select candidate tree node tests. While they do a better job if there exists some feature that favors spatial autocorrelation but does not provide the largest information gain in one tree node test, they still relies on local testing of information by tree nodes. Thus, if all the candidate tests have poor spatial autocorrelation, this type of decision tree will still select one of them, resulting in salt-and-pepper noise. This means neither approach adequately accounts for spatial autocorrelation in the prediction phase.

To address this limitation, we recently defined a focal-test-based spatial decision tree (FTSDT) model [19, 20], whereby the tree traversal direction of a learning sample is based on not only local but also focal (neighborhood) properties of features. We proposed FTSDT learning algorithms and evaluated the classification performance of the proposed approach on real-world remote sensing datasets. We also extended the basic FTSDT algorithm in a journal paper [21] with the following additional contributions:

1. We add a new design decision in the FTSDT model to allow focal function computation with adaptive neighborhoods (i.e., FTSDT-adaptive). Compared with previous FTSDT with fixed neighborhoods (i.e., FTSDT-fixed), the new design decision can adjust the neighborhood shape to avoid over-smoothing in wedge-shaped areas.
2. We characterize the computational structure of the FTSDT learning algorithm and confirm that the computational bottleneck is a vast number of focal function computations. We design a refined algorithm (FTSDT-Refined) that reuses focal values across candidate thresholds and prove its correctness.
3. We also provide cost models of our previous baseline algorithm and our refined algorithm and show that the refined algorithm improves computational scalability.
4. We compare the classification performance of FTSDT-adaptive with FTSDT-fixed as well as LTDT on real-world datasets. Results show that FTSDT-adaptive improves classification accuracy of FTSDT-fixed and LTDT.
5. We also conduct experimental evaluations of computational performance on real-world datasets with various parameter settings. Experiment results show that our refined algorithm significantly reduces computational time cost.

Scope: This work focuses on incorporating focal tests inside a decision tree for raster data classification. Other classification algorithms such as Markov random field [22], spatial autoregression (SAR) model [23], logistic regression, and neural network are beyond the scope. In addition, for simplicity, this work only considers learning samples with continuous features. The case of discrete features is not addressed.

Outline: The chapter is organized as follows: Sect. 5.1 introduces basic concepts and formalizes the SDTL problem; Sect. 5.2 presents our FTSDT learning algorithm, especially a new design decision to allow focal function with adaptive neighborhoods. Section 5.3 describes computational optimization and the refined algorithm design with theoretical analysis. Computational and classification performances of the proposed algorithms are evaluated in Sect. 5.4. Section 5.6 discusses some other relevant techniques in the literature. Section 5.7 concludes the chapter with future work.

5.2 Basic Concepts and Problem Formulation

This section introduces basic concepts and formally defines the spatial decision tree learning problem.

5.2.1 *Basic Concepts*

Spatial raster framework: A spatial raster framework F is a tessellation of a 2-D plane into a regular grid. On a spatial raster framework, there may exist a set of explanatory feature maps, as well as a class label map. For example, Fig. 5.3 shows a spatial raster framework with explanatory features $f^1, f^2, ..., f^m$ and a class label map c. Each grid cell on the raster framework is a *spatial data sample* (e.g., location i in Fig. 5.3). For simplicity, we use the words "sample," "pixel," "location," and "spatial data sample" interchangeably in the remainder of the chapter.

Neighborhood relationship: A spatial neighborhood relationship describes the range of dependency between spatial locations. It is commonly represented as a W-matrix, whose element $W_{i,j}$ has a nonzero value when locations i and j are *neighbors* and a zero value otherwise. For example, in Fig. 5.3, the pixel in dark gray has eight neighbors indicated in light gray in a 3-by-3 neighborhood.

Salt-and-pepper noise: Salt-and-pepper noise is defined as a kind of fat-tail impulse noise whose values are often extreme (e.g., minimum or maximum) [24]. In a predicted class label map, salt-and-pepper noise can be considered as a single

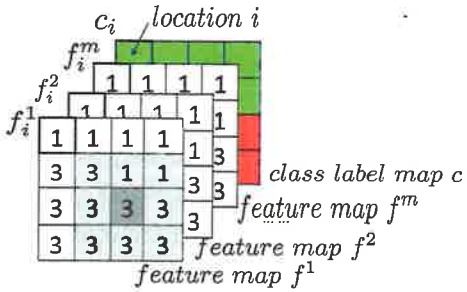


Fig. 5.3 Example of a spatial raster framework and a neighborhood relationship

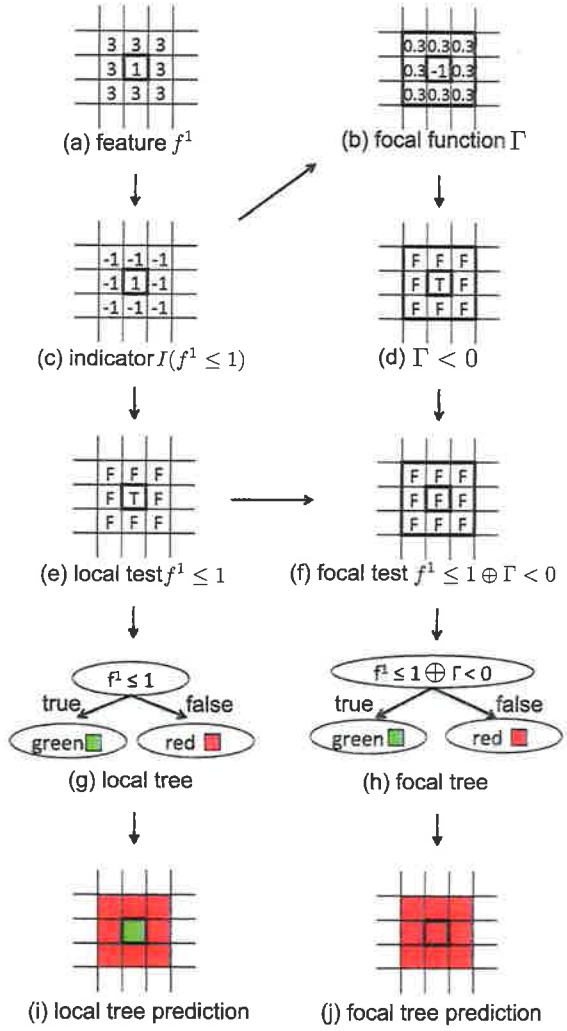


Fig. 5.4 Comparison of a local test versus a focal test, a local-test-based decision tree versus a focal-test-based spatial decision tree. ("T" is "true"; "F" is "false")

pixel (or a small group of contiguous pixels) that is distinct from its (or their) spatial neighborhood. For example, in Fig. 5.4i, the central pixel is salt-and-pepper noise.

Local test and *indicators*: A *local test* $f^m \le \delta$ checks the value of feature f^m at a sample's location against a threshold δ. The local test results can be represented as *indicator variable* $I(f^m \le \delta)$ or simply I, whose value is 1 when $f^m \le \delta$ is true and -1 otherwise. A decision tree whose tree nodes conduct local tests is called a *<>local-test-based decision tree (LTDT)*. For example, given the feature f^1 shown in Fig. 5.4a, the local test results of $f^1 \le 1$ and corresponding indicator variables are shown in Fig. 5.4c, e, respectively. The corresponding LTDT and its class predictions with salt-and-pepper noise are shown in Fig. 5.4g, i.

Table 5.1 List of symbols and descriptions

Symbols	Descriptions
δ	A local test threshold
$\oplus$	Logic operator "xor," i.e., $0 \oplus 1 = 1, 1 \oplus 0 = 1, 0 \oplus 0 = 0, 1 \oplus 1 = 0$
$W_{i,j}$	Neighborhood relationship between location i and location j
f^m, f_i^m	Value of feature m, the feature value at location i
$I(f^m \leq \delta), I_i$	Indicator variable of local test $f^m \leq \delta$, the indicator variable at location i
Γ, Γ_i	Focal Gamma autocorrelation statistic, the focal Gamma at location i

Focal function and *spatial autocorrelation statistic*: A focal function is an aggregate of non-spatial attribute values in the neighborhood of a location. One important kind of focal function is *focal autocorrelation statistic*, which measures the dependency between attribute values of a location and the values of its neighbors. For example, the focal Gamma index [25] on local test indicators is defined as (Table 5.1)

$$\Gamma_i = \frac{\sum_j W_{i,j} I_i I_j}{\sum_j W_{i,j}},$$

where i and j are locations, $W_{i,j}$ is a W-matrix element, and I_i and I_j are indicator variables of a local test. A negative focal Gamma value (i.e., $\Gamma < 0$) indicates that the current location is potentially salt-and-pepper noise. Figure 5.4b shows an example of focal Gamma values computed on indicator variables in Fig. 5.4c with a 3-by-3 neighborhood. The central location has a negative Gamma because its local test result is different from its neighbors'.

Focal test: A focal test is a test or a combination of tests on attribute values in a neighborhood of a location. For example, $f \leq \delta \oplus \Gamma < 0$, where $\oplus$ is an "xor" logical operator, is a focal test that combines a local test $f \leq \delta$ and the test $\Gamma < 0$. This combined focal test is less prone to salt-and-pepper noise, compared with the local test $f \leq \delta$ only. The reason is that salt-and-pepper noise pixels often have a negative focal Gamma index (i.e., $\Gamma < 0$ is true), and their local test results ($f \leq \delta$) are flipped by logical operator $\oplus$ (i.e., "false" xor true becomes "true," and "true" xor true becomes "false"). For instance, the local test result of the central pixel in Fig. 5.4e is true, but false for its neighborhood; while the focal test result of the same pixel in Fig. 5.4f is false, the same as for its neighborhood.

Focal-test-based spatial decision tree (FTSDT): An FTSDT is a tree whose nodes conduct focal tests. An example of FTSDT is in Fig. 5.4h, and its class predictions are in Fig. 5.4j. In our approach, both local tests and focal tests are defined on a single feature. When multiple features exist, the local test or focal test on each feature is considered as a candidate tree node test and the best candidate test is selected for a tree node, similar to the situation of a traditional decision tree.

5.2.2 *Problem Definition*

Based on the concepts above, the spatial decision tree learning problem is formally defined as follows:

Given:
- A spatial raster framework F
- A spatial neighborhood definition and its maximum size S_{max}
- Training and test samples drawn from F

Find:
- A decision tree model based on training samples.

Objective:
- Minimize classification errors as well as salt-and-pepper noise

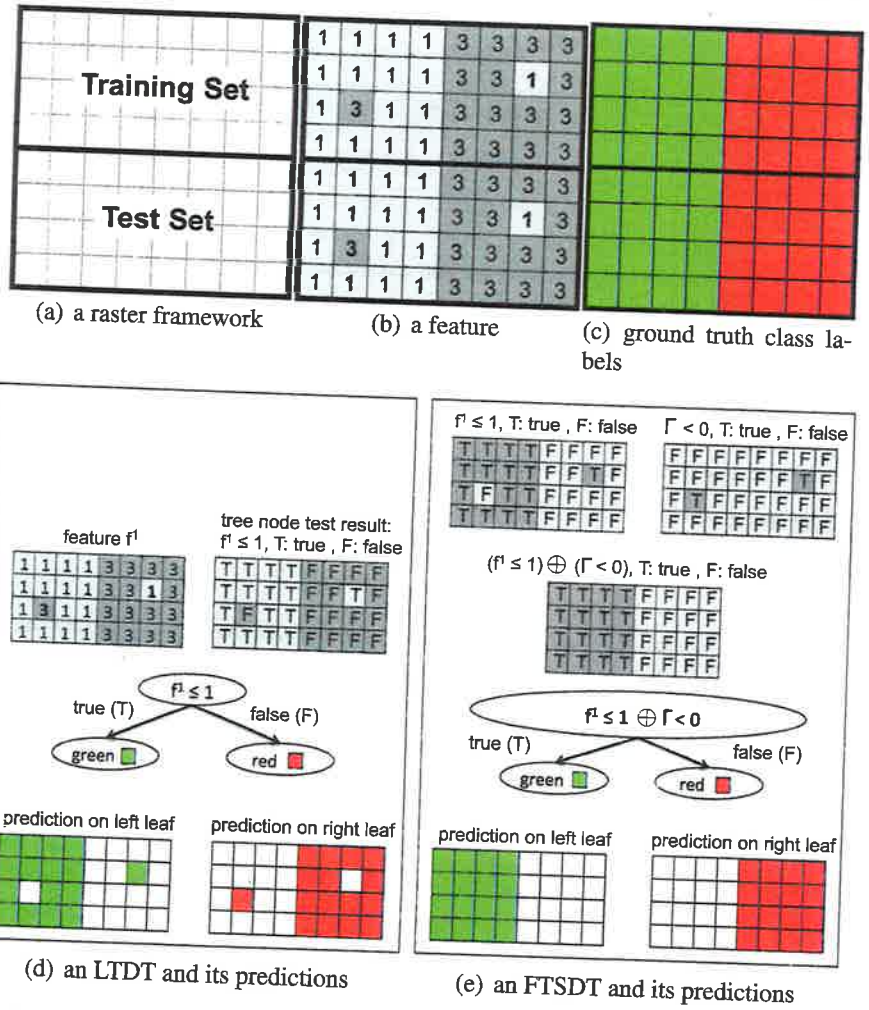
Constraints:
- Training samples form contiguous patches of locations in F
- Spatial autocorrelation exists in class labels

Problem description: The output decision tree model can be a local-test-based decision tree or a focaltest-based spatial decision tree, depending on the selected approach. Parameters to be learned from the training set are the tree structure, as well as which feature f, test thresholds δ, and proper neighborhood size s (in the case of FTSDT) to use in each tree node.

Example: Consider the example of problem inputs and outputs in Fig. 5.5. The raster spatial framework F, shown in Fig. 5.5a, consists of training pixels on the upper half and test pixels on the lower half. Neighborhood relationship is defined as a 3-by-3 square window. The minimum tree node size is four. Figure 5.5b shows a candidate feature f^1. Figure 5.5c shows the ground truth class labels. The output local-test-based traditional decision tree learned from the training set and its predictions with salt-and-pepper noise are shown in Fig. 5.5d. In contrast, the output focal-test-based spatial decision tree and its predictions without salt-and-pepper noise are shown in Fig. 5.5e.

5.3 FTSDT Learning Algorithms

This section describes the baseline FTSDT learning algorithm (i.e., without computational optimization) of the focal-test-based spatial decision tree. The learning algorithm has two phases: a training phase, FTSDT-Train; and a prediction phase, FTSDT-Predict. FTSDT-Train here extends the previous one we proposed in [19] by allowing focal function tests with adaptive neighborhoods to avoid over-smoothing in wedge-shaped areas.



(a) a raster framework (b) a feature (c) ground truth class labels

(d) an LTDT and its predictions (e) an FTSDT and its predictions

Fig. 5.5 Illustrative problem example (best viewed in color)

5.3.1 *Training Phase*

FTSDT-Train (Algorithm 1) learns an FTSDT classifier from training samples. It includes two subroutines (Node-Split and Focal Function). Similar to traditional C4.5 [1], it is a divide and conquer method with a greedy strategy (i.e., maximize information gain).

Steps 1–3 check the stopping criteria. If the training samples are less than the minimum tree node size, or all the class labels are identical, a leaf labeled with the majority class will be returned.

Steps 4–13 enumerate through every candidate feature f, every neighborhood size s, and every candidate threshold δ to select the best setting for a model tree node. Candidate thresholds δ are generated from distinct values of feature f in the training samples (steps 8–9). Step 10 calls a Node-Split subroutine to split training samples. Step 11 evaluates the corresponding information gain on the column of class labels. Steps 12–13 update the current best candidate test.

Steps 14–18 create an internal node with the best test, split the training samples into two subsets accordingly, recursively call FTSDT-Train on each subset, and return the internal node.

Algorithm 2 FTSDT-Train(T, S_{max}, N_0, *neighType*)

Input:
- T: rows are samples, columns are features (last column as class)
- S_{max}: maximum neighborhood size
- N_0: minimum decision tree node size
- *neighType*: neighborhood type, 0 for fixed neighborhood, 1 for adaptive neighborhood

Output:
- root of an FTSDT model

1: let N be number of samples, F be number of features, c be column index of classes; $IG_0 = -\infty$
2: **if** $N < N_0$ or T same class **then**
3: **return** a leaf node;
4: **for each** $f \in \{1...F\}$ **do**
5: sort rows of T //in ascending order of f^{th} column
6: **for each** $s \in \{0...S_{max}\}$ **do**
7: **for each** $i \in \{N_0...(N - N_0)\}$ **do**
8: **if** $T[i][f] < T[i+1][f]$ **then**
9: $\delta = T[i][f]$
10: $\{T_1, T_2\}$ = **Node-Split**(T, f, δ, s, *neighType*);
11: IG = **InformationGain**($T[\ ][c], T_1[\ ][c], T_2[\ ][c]$)
12: **if** $IG > IG_0$ **then**
13: $IG_0 = IG$; $s_0 = s$; $f_0 = f$; $\delta_0 = \delta = T[i][f]$;
14: I = CreateInternalNode(f_0, δ_0, s_0);
15: $\{T_1, T_2\}$ = **Node-Split**(T, f_0, δ_0, s_0, *neighType*);
16: $I.left$ = **FTSDT-Train**(T_1, S_{max}, N_0, *neighType*)
17: $I.right$ = **FTSDT-Train**(T_2, S_{max}, N_0, *neighType*)
18: **return** I

Node-Split: The Node-Split subroutine (Algorithm 3) splits the training samples into two subsets based on their focal test results and proceeds as follows:

Step 1 initializes the two subsets as empty sets. Samples with node test results TRUE will be assigned to one subset, and samples with test results FALSE will be assigned to the other.

Steps 2–11 compute the focal tree node test result of each training sample and add the sample to its appropriate subset accordingly. The algorithm begins by computing local test indicators (I) of all samples. It then computes the focal function value ($focalFun[i]$) via a *FocalFunction* subroutine and computes the focal test result

(*focalTest*[*i*]) on each sample location. For example, we may specify the focal function as Γ, and the focal test as "$f \leq \delta \oplus \Gamma < 0$".

Algorithm 3 Node-Split($T, f, \delta, S, neighType$)

Input:
- T: rows as samples, columns as features (last column as class)
- f: a feature index
- δ: threshold of feature test
- S: neighborhood size
- *neighType*: neighborhood type, 0 for fixed neighborhood, 1 for adaptive neighborhood

Output:
- $\{T_1, T_2\}$: sample subsets with test results true and false respectively

1: $T_1 = T_2 = \emptyset$
2: **for each** $i \in \{1...N\}$ **do**
3: indicators $I[i] = I(T[i][f] \leq \delta)$
4: **for each** $i \in \{1...N\}$ **do**
5: *focalFun*[*i*] = **FocalFunction**($I[], i, s, neighType$)
6: *focalTest*[*i*] = *FocalTest*($I[i], focalFun[i]$)
7: **if** *focalTest*[*i*] == *true* **then**
8: $T_1 = T_1 \cup \{T[i]\}$
9: **else**
10: $T_2 = T_2 \cup \{T[i]\}$
11: **return** $\{T_1, T_2\}$

Algorithm 4 FocalFunction($I, i, s, neighType$)

Input:
- I: vector of indicator variable values
- i: current location
- s: neighborhood window size
- *neighType*: neighborhood type, 0 for fixed neighborhood, 1 for adaptive neighborhood

Output:
- *FocalFun*[*i*]: focal function value at current location i

1: identify the 2s+1 by 2s+1 window centered on location i
2: **if** *adaptNeigh* == 0 **then**
3: $W_{i,j} = 1$ for all j in the window, $W_{i,j} = 0$ otherwise.
4: **else**
5: get connected components of same I values in the window
6: identify the topologically outermost component cc_0 that contains or surrounds location i
7: $W_{i,j} = 1$ for all j in cc_0, $W_{i,j} = 0$ otherwise.
8: compute focal function value *foc* at location i based on $W_{i,j}$
9: **return** *foc*

FocalFunction: The FocalFunction subroutine (Algorithm 4) computes the focal function values of local test indicators in the neighborhood of a location. It has an important parameter *neighType*, whose value is 0 for a fixed neighborhood, and is 1 for an adaptive neighborhood. The intuition behind an adaptive neighborhood is to

utilize spatial topological relationships to select proper neighbors of the central pixel in a fixed window.

Step 1 identifies all locations within the window of size 2s + 1 by 2s + 1 centered on the current location. These locations are potential neighbors of i.

Steps 2 and 3 determine that all the locations in the window are neighbors of i if a fixed neighborhood is used, similar to our previous work in [19].

Steps 4–7 determine which locations in the window are neighbors if an adaptive neighborhood is used. The window is first segmented into different connected components, each of which has the same indicator value. Then the component that is the outermost, and that surrounds or contains the current location i, is considered as the actual set of neighbors.

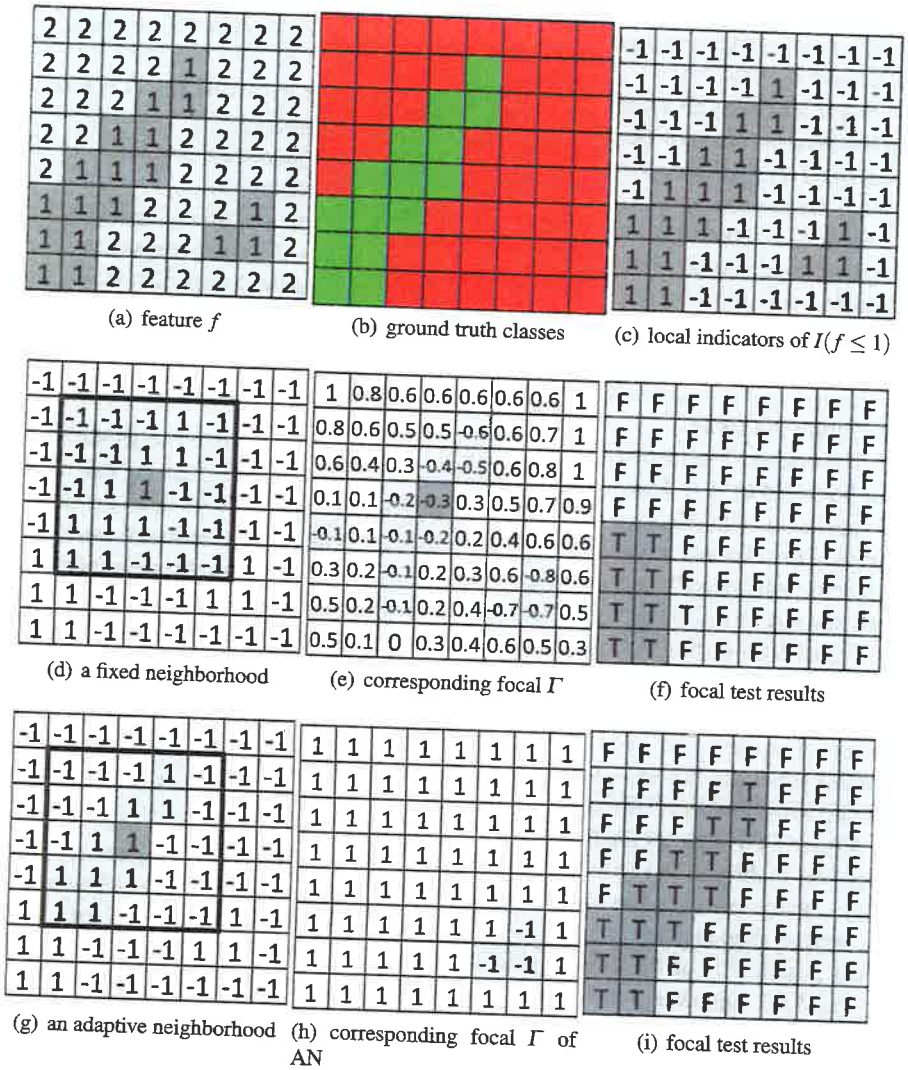
Steps 8 and 9 compute a focal function value based on the neighbors identified and return the value.

Illustration: The entire execution trace of FTSDT-Train with fixed neighborhoods can be found in [19]. Due to space limitations, here we only illustrate the extension of focal tests with adaptive neighborhoods by comparing them with fixed neighborhoods. Consider the example in Fig. 5.6, which describes one iteration of candidate test selection (steps 9 to 10 in Algorithm 2). Assume that the current neighborhood window size is $s = 2$ (i.e., 5-by-5).

Figure 5.6a–c shows current candidate feature f, ground truth classes, and local test indicators on the current test threshold $\delta = 1$, respectively. The feature f map (Fig. 5.6a) contains a wedge-shaped patch (fifteen pixels with feature value 1 on the lower left corner) and three salt-and-pepper noise pixels. The pixels on the wedge-shaped patch are not salt-and-pepper noise and thus should not be smoothed (i.e., should avoid over-smoothing).

Figure 5.6d–f shows the focal test results with fixed neighborhoods. For instance, Fig. 5.6d highlights the fixed neighborhood (in light gray) of a central pixel (in dark gray), which contains too many irrelevant pixels (with indicator value -1) outside the wedge-shaped patch (the one we previously describe in the last paragraph). The focal function Γ of the central pixel is -0.3 (i.e., $\Gamma < 0$) as shown in dark gray in Fig. 5.6e, mistakenly indicating that it is salt-and-pepper noise. Thus, its final focal test result mistakenly flips its local test result from "true" to "false". Similarly, several other pixels in the wedge-shaped patch are also over-smoothed. The final over-smoothed focal test results of the patch are shown in dark gray in Fig. 5.6f.

In contrast, the bottom row of Fig. 5.6 shows focal test results with adaptive neighborhoods. Figure 5.6g highlights an adaptive neighborhood (in light gray) of the central pixel (in dark gray), which is a connected component (with indicator value 1) that contains the central dark pixel. The focal function Γ of the central pixel is now 1 (Fig. 5.6b) based on the adaptive neighborhood. Thus, the final focal test is still "true" ($\Gamma < 0$ is false, "true" xor false is still "true"). The three salt-and-pepper noise pixels are still smoothed. Comparing Fig. 5.6f and i, it is clear that focal tests with adaptive neighborhoods can better separate the two classes (i.e., give higher information gain) due to less over-smoothing of the wedge-shaped area.



(a) feature f (b) ground truth classes (c) local indicators of $I(f \leq 1)$

(d) a fixed neighborhood (e) corresponding focal Γ (f) focal test results

(g) an adaptive neighborhood (h) corresponding focal Γ of AN (i) focal test results

Fig. 5.6 Comparison of focal tests with fixed and adaptive neighborhoods, $s = 2$ (i.e., 5 by 5 window)

5.3.2 *Prediction Phase*

The FTSDT-Predict algorithm (Algorithm 5) uses an FTSDT to predict the class labels of test samples based on their feature values and a spatial neighborhood structure. It is a recursive algorithm. If the tree node is a leaf, then the class label of the leaf is assigned to all current samples. Otherwise, samples are split into two subsets

according to the focal test results in the root node, and each subset is classified by its corresponding subtree.

Algorithm 5 FTSDT-Predict(R, T)

Input:
- R: root of an FTSDT model
- T: rows as samples, columns as features

Output:
- C: $C[i]$ as class label of i^{th} sample

1: **if** $R.type == Leaf$ **then**
2: assign C with $R.class$
3: **return** C
4: $f_0 = R.f$, $\delta_0 = R.\delta$, $s_0 = R.s$
5: $\{T_1, T_2\}$ = **Node-Split** (T, f_0, δ_0, s_0)
6: C_1 = **FTSDT-Predict** $(R.Left, T_1)$;
7: C_2 = **FTSDT-Predict** $(R.Right, T_2)$;
8: **return** C = combine(C_1, C_2)

5.4 Computational Optimization: A Refined Algorithm

This section addresses the computational challenges of the focal-test-based spatial decision tree learning process. It first identifies the computational bottleneck of the baseline training algorithm; proposes a refined algorithm; proves its correctness; and finally provides a cost model for the computational complexity. For simplicity, examples in this section are with a fixed neighborhood. However, the proposed refined algorithm and its analysis are also applicable to the case of adaptive neighborhoods.

5.4.1 *Computational Bottleneck Analysis*

Recall that the baseline algorithm (Algorithm 2) calls a Node-Split subroutine for every distinct value (i.e., candidate threshold) on every feature and neighborhood size. Each call involves focal function computations for all samples and is, therefore, a likely computational bottleneck. To verify this hypothesis, we conducted computational bottleneck analysis with parameter settings $S_{max} = 5$ and $N_0 = 50$. The results, shown in Fig. 5.7, confirm that the focal function computation accounts for the vast majority of total time cost. Furthermore, this cost increases much faster than other costs as training sample sizes increase.