Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Štajdohar

Vizualizacija in analiza prostora napovednih modelov

DOKTORSKA DISERTACIJA

Ljubljana, 2012

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Štajdohar

Vizualizacija in analiza prostora napovednih modelov

DOKTORSKA DISERTACIJA

Mentor: prof. dr. Janez Demšar

Ljubljana, 2012

University of Ljubljana
Faculty of Computer and Information Science

Miha Štajdohar

Visualization and analysis of the space of prediction models

DOCTORAL DISSERTATION

Advisor: Prof. Dr. Janez Demšar

Ljubljana, 2012

Povzetek

Pri odkrivanju znanja iz podatkov običajno zgradimo več modelov, ki razložijo zanimive vzorce. Modeli morajo biti dovolj preprosti, da jih domenski strokovnjaki razumejo. Drevo z več sto vozlišči ali razsevni diagram, ki slika nekaj deset značilk v dvodimenzionalno ravnino, imata lahko odlično napovedno točnost, a sta morda povsem nerazložljiva. Po drugi strani preprost model ne more razložiti zapletenih podatkov z veliko zanimivimi relacijami. Problem rešimo z večjim številom preprostih modelov, kjer vsak predstavlja vpogled v manjši del, skupaj pa tvorijo celotno sliko. Izziv, ki ga predstavlja takšna predstavitev znanja, je, kako množico modelov predstaviti uporabniku. Čeprav zmorejo danes računalniki zgraditi na tisoče modelov, si te obsežne zbirke domenski strokovnjak ne more ogledati brez pomoči računalnika.

Zapletene podatke lahko torej predstavimo z večjim številom ponazoritev, od katerih vsaka izpostavi določen, manjši del podatkov – na primer s preslikavami, kjer vsaka prikazuje manjše število značilk. Podobno lahko dobre napovedne modele zgradimo iz zbirke preprostih (šibkih) modelov, na primer z naključnimi gozdovi klasifikacijskih dreves, kjer se vsako osredotoči na svoj del podatkovne domene. Primanjkuje pa načinov za raziskovanje omenjenih metod. Zanima nas, ali obstaja omejen izbor ponazoritev, ki vsebuje celotno sliko problema; in nadalje, ali se lahko sprehodimo po naključnem gozdu in opazujemo skupne lastnosti dreves v različnih območjih.

V disertaciji predstavimo nov pogled na napovedne modele. Predlagamo postopek za ponazoritev in organizacijo prostora klasifikacijskih modelov, ki ga lahko uporabnik raziskuje z interaktivnim orodjem. Metoda, opisana v disertaciji, lahko poleg napovednih modelov uredi tudi različne vrste vizualizacij podatkov. Poleg kvalitete upošteva njihovo raznolikost in nudi več informacij ob enakem številu pregledanih projekcij.

Mere podobnosti med napovednimi modeli v disertaciji uporabimo tudi za raziskovanje naključnih gozdov. Ti napovedni modeli so sestavljeni iz raznovrstnega nabora napovednih dreves, ki so običajno majhna: imajo do 10 vozlišč. Vsako zato pokrije manjši del prostora napovednega problema. Naključni gozdovi se veliko uporabljajo predvsem zaradi dobre napovedne točnosti. V primerjavi z enim napovednim drevesom pa jih izrazito težko razložimo ali ponazorimo. Metoda predlagana v disertaciji pomaga pri analizi naključnega gozda. Drevesa lahko združimo v gruče in v vsaki izpostavimo le najpomembnejše drevo. Poleg raziskovanja modelov v naključnem gozdu ta metoda razloži napovedi za različne razrede.

Ključni sestavini opisane metodologije sta interaktivno orodje za raziskovanje prostora napovednih modelov in nova metoda za optimizacijo ponazoritve grafa, zasnovana za prikaz omrežja modelov. Večina obstoječih orodij za analizo in ponazoritev omrežij te zgolj riše in računa njihove osnovne lastnosti. Pri njih uporabnik ne more interaktivno raziskovati omrežja, ga spreminjati, uporabljati drugih statističnih metod in metod za odkrivanje znanj iz podatkov pri raziskovanju podomrežij, prikazovati drugih informacij na omrežju in podobno. V okviru tega doktorskega dela

smo razvili več orodij za raziskovanje omrežij in prostora napovednih modelov, ki rešujejo omenjene probleme.

Poleg orodja za raziskovanje smo razvili tudi metodo za vizualizacijo omrežja modelov. Ta pogosto vsebujejo med seboj nepovezane dele – *komponente*. Algoritmi za optimizacijo izrisa omrežij običajno takšne komponente razporedijo naključno, torej neodvisno od preostalega omrežja. Ker so povsem nepovezani deli omrežja pogosto postavljeni skupaj, lahko pride do napačnih tolmačenj. Naša metoda (FragViz) poleg informacije o strukturi omrežja upošteva tudi dodatno informacijo o podobnosti nepovezanih podgrafov. Postopek sestoji iz dveh delov: točke najprej uredi znotraj vsakega podgrafa, nato pa komponente razporedi tako, da razdalja med njimi odraža njihovo sorodnost. Uporabnost metode FragViz predstavimo na resnični domeni in ugotovimo, da nam poda znanje, ki bi sicer ostalo skrito.

Ključne besede

vizualizacija informacij, analiza podatkov, vizualno odkrivanje znanja v podatkih, strojno učenje, klasifikacija, uvrščanje, razpoznavanje vzorcev, meta učenje, kombiniranje algoritmov strojnega učenja, risanje omrežij, analiza omrežij, optimizacija ponazoritve omrežij, odkrivanje gruč v omrežjih

Abstract

Data mining—a search for interesting patterns in the data—typically creates a number of different models. These models need to be simple enough to be *graspable* by the human expert. A tree consisting of hundreds of nodes or a scatter plot projecting dozens of variables into a two-dimensional plane may offer great classification accuracy or class separation, yet they may be impossible to interpret. But simple models cannot describe the complex data which may contain many interesting relations. The solution is to create a large number of simple models, where each of them offers insight into a small part of the problem domain, but together they present a complete picture. The problem, however, is the presentation of the big picture. While a computer can infer thousands of models, the human expert is incapable of reviewing them without assistance.

We argue that a set of partial views can represent complex data. Views may include linear projections, each involving at most a couple of variables and showing a single, particular, and simplified perspective or relation. Similarly, good predictive models can be build using ensembles of simple models, such as random forests of small trees, each covering a part of the problem domain. These approaches lack techniques for manual exploration. Is it possible to select a limited number of visualizations which will provide a complete picture? Can we navigate through the random forest and observe the common properties of models in each region?

We propose a method for creating maps of classification models, which are presented to the user to interactively explore the model space. The proposed technique can, besides organizing predictive models, rank some types of visualizations. We extend existing methods—that rank projections based on quality—to consider projection diversity, and show that our method yields more information when viewing the same number of projections.

We describe a *model-map-based* technique for the visualization and exploration of random forest, a prediction model that is assembled of random prediction trees. Those are normally small (up to 10 vertices), so each covers only a small part of the space of the decision problem. The random forest predicts remarkably well, however it is prohibitively difficult to explain or visualize—in comparison to a single prediction tree. Our method assists an expert with the random forest analysis, for example, in clustering similar trees together to emphasize the diverse ones. In addition to interactive exploration of a random forest, the *model map* can explain predictions of different classes.

The final ingredient of the technique is a versatile interactive tool for exploration of maps and a new network layout technique, particularly suitable for handling the visualization of the *model map* networks. Most existing tools for network analysis are limited to drawing networks and computing their basic general characteristics. With this tools, it is impossible for the user to interactively and graphically manipulate the networks, select and explore sub-graphs using other statistical and data mining

techniques, add and plot various other data within the graph, and so on. We developed tools that address these challenges, widgets and modules for exploration of networks and *model maps* within the general component-based environment Orange.

We propose a network layout optimization algorithm which is designed to visualize fragmented networks: FragViz. Networks of prediction models are usually fragmented. They consist of unconnected components which popular network alignment algorithms place arbitrarily with respect to the rest of the network. This can lead to misinterpretations due to the proximity of otherwise unrelated elements. FragViz incorporates additional information on relations between unconnected network components. It uses a two-step approach by first arranging the nodes within each of the components and then placing the components so that their proximity in the network corresponds to their relatedness. In the experimental study we demonstrate that FragViz can obtain network layouts which are more interpretable and hold additional information that could not be exposed using classical network layout optimization algorithms.

Keywords

information visualization, exploratory data analysis, visual data mining, machine learning, data mining, meta learning, ensemble learning, network layout optimization, network analysis, community detection in graphs

Acknowledgments

First and foremost, I would like to express my sincerest thanks to my mentor, professor Janez Demšar, for a constant exchange of ideas, guidance through tough problems, sharing his knowledge, patience, and most of all, teaching me how to get my work published.

I would like to extend my gratitude to all the members of the Bioinformatics and Artificial Intelligence Laboratories for creating a great working atmosphere. Especially to Minca Mramor for the contribution of biological expertise in case studies in Section 2.3.3, and for teaching me the difference between the acute myeloid and lymphoblastic leukemia, Professor Blaž Zupan for advising on the FragViz method, Lan Umek for countless debates over statistical tests, Marko for guiding me through random forests, Gregor, Jure, Lan, and Matija for writing, erasing, and rewriting ideas together, time and again on the whiteboard.

My deepest gratitude goes also to the ones who have been dear friends during my PhD studies: Kaja for teaching me the English grammar again, and for constantly reminding me to use articles before nouns, Ana, Damjan, Elio, Jan, Luka, and all the others who laugh at my ridiculous jokes. Above all, I thank my family for the support. This dissertation is dedicated to my parents Ivo and Nataša, my sister Lara, and my dearest Jasmina who shares her kindness with me every single day.

Contents

Chapter 1

Introduction

Data mining—a search for interesting patterns in the data—typically creates a number of different models. These models need to be simple enough to be *graspable* by the human expert. A tree consisting of hundreds of nodes or a scatter plot projecting dozens of variables into a two-dimensional projection may offer great classification accuracy or class separation, yet may be impossible to interpret. But simple models clearly cannot describe the complex data which may contain many interesting relations. The solution is then to create a large number of simple models, each offering insight into a small part of the problem domain. Taken together they can provide a complete picture. The problem is the presentation of the big picture. While a computer can infer thousands of models, the human expert is incapable of reviewing them without assistance.

We argue that complex data can be represented using a number of partial views, such as linear projections, each involving at most a couple of variables and showing a single, particular, and simplified perspective or relation. Similarly, good predictive models can be build using ensembles of simple models, such as random forests of small trees, each covering a part of the problem domain. These approaches lack techniques for manual exploration. Is it possible to select a limited number of visualizations which will provide a complete picture? Can we navigate through the random forest and observe the common properties of models in each region? We propose a method for creating maps of classification models, which are presented to the user to interactively explore the model space.

In Chapter 2 we present FragViz, a new method for visualization of fragmented networks. Chapter 3 describes the Net Explorer, our tool for interactive exploration of networks. These methods are required for the techniques developed in the next three chapters. Chapter 4 defines the space of classification models through several definitions of model similarity. In the context of this work, a model can be either a predictive model or a visualization that can be used for making predictions. Chapters 5 and 6 use this paradigm for building maps. Chapter 5 introduces maps of visualizations and shows their advantages over list-based ranking of visualizations.

Chapter 6 shows how model maps can be used for exploration and, potentially, enhancement of ensemble methods such as random forests.

The material in Chapter 2 is based on our paper published in *BCM Bioinformatics* [107], and Chapter 3 is based on the paper to be published in the *Journal of Statistical Software*.

1.1 Thesis Overview

In the thesis we first propose FragViz [107]—a novel network layout optimization algorithm which is designed to visualize fragmented networks. The goal of network layout optimization is to construct a meaningful visualization. Different layout optimization algorithms emphasize different vertex relations. We use the FragViz algorithm in case studies in Sections 4, 5, and 6, since networks of prediction models are usually fragmented (consist of many unconnected components).

In Chapter 3 we overview the current state-of-the-art tools for the analysis, visualization, and interactive network exploration, for instance igraph, statnet, Gephi, Network Workbench, and GUESS. We compare these with our own tool: the Orange Network add-on, which unites methods developed in the process of doctoral research.

We discuss the model space concept—the main driver of our work—in Section 4. We propose a technique for visually organizing a large number of predictive models, including some types of visualizations called *model map*. We define a prediction model network and measures of similarity between prediction models, which we compare with the existing measures of difference between two probability distributions (*e.g.* information gain, mutual information, and others).

Two applications of the *model map* technique follow in Chapters 5 and 6. First, we extend the VizRank method (which evaluates visualizations through classification) proposed by Leban [81, 82] et al. in 2005. VizRank estimates the projection quality based on the separation of data instances with regard to their class. It ranks linear projections: they are first transformed to classification models and then evaluated on the coordinates of the projected instances. The projections are given in a list, ordered by the projection quality. The method is designed to search for the best projection; this is also its biggest disadvantage. It works well if a single best projection is needed, but it fails to find other interesting projections that would offer an alternative view on the data.

We visualize and explore random forests [18]; prediction models that are assembled from a diverse collection of prediction trees. Those are normally small (up to 10 vertices), therefore each covers only small part of the space of the decision problem. Random Forests predict remarkably well, but are prohibitively difficult to explain in comparison to a single prediction tree. We discuss how to use the *model map* to

assist an expert with the random forest analysis; for example, cluster similar trees together to emphasize the diverse ones. In addition to interactive exploration of a Random Forest, the *model map* can explain predictions of different classes. Finally, we explore the possibility to build forests of classification trees with the *model-map-based* algorithm. As different parts of the network cover different aspects of the problem domain, choosing prediction models from various parts of the network should give better predictions.

1.2 Contributions to Science

The dissertation includes the following original contributions to science:

- A network layout optimization method—FragViz, particularly suitable for fragmented networks, which consist of many disconnected components.
- Definition of novel measures of similarity of prediction models, particularly designed to infer a network for the model space analysis.
- A methodology for the interactive exploration and analysis of the prediction model network.
- An extension of the VizRank method that constructs a network instead of a list of visualizations. The model network offers additional information (namely projection diversity) besides the projection quality.
- A general methodology for visualizing an ensemble of prediction models. Special attention is given to the visualization of random forests.

Chapter 2

Visualization of Fragmented Networks

Data collections with millions of instances and thousands of attributes are common in the present information era. Although methods capable of visualizing this volume of data exist, their visualization is useless in most cases. For example, trying to visualize more than a few dozen attributes in parallel coordinates plot results in an unreadable image, where it is impossible to observe the influence of individual attributes on a particular data instance. The goal of information visualization is not only the visualization, but gaining a deeper understanding of the data.

This chapter first reviews the area of exploratory data analysis with a focus on networks and network layout optimization. We then present a novel network layout optimization algorithm that is suitable for fragmented networks and will also be used in the following chapters.

2.1 Information Visualization

Information visualization [55, 72] is the interdisciplinary study of the visual representation of large-scale collections of complex data. Visual representation and interactive tools help scientists gain insight, understand, and explore extensive amounts of information.

2.1.1 Exploratory Data Analysis

In his ground-breaking work, Exploratory Data Analysis [111], Tukey promoted informal data analysis, for instance using simple graphical models. Friedman and Tukey [42] also implemented a method called *projection pursuit* based on earlier

ideas by Kursaal [77, 78]. The method searches for interesting linear projections of high-dimensional unlabeled data. It is based on the notion of interestingness, defined as the opposite of randomness. Optimization algorithm is used to find one or a few best-ranked projections which are shown to the user. The method can be treated as a special case of *grand tour* [5, 3], which shows a dynamic two-dimensional projection which is being rotated, either randomly or using some heuristic strategy, to give the user a multidimensional perception of the data and potentially showing it from interesting perspectives.

In Section 4.3 we propose the *model map* method that is similar to *grand tour* when applied to linear projection models only. The difference is that *grand tour* shows a continuous set of visualizations, one at a time, without organizing them into structure. Our method analyzes a finite set of projections which are prepared in advance, organizes them into a network, and lets the user manually pick and view one at a time.

Leban *et al.* [81, 82] explored ways to rank visualizations according to their ability to separate instances from different classes. His method, VizRank, is mostly focused on linear projections, which are turned into predictors by using the projected coordinates for k-NN classification. Constructed projections are sorted according to their quality, as estimated by the predictive accuracy of the associated k-NN classifiers.

The shortcoming of VizRank is that it searches for optimal visualizations instead of diverse ones. It is useful for obtaining the single best visualization out of a dozen or even few dozens of very similar visualizations presented at the top of the list, while it is difficult to dig out other interesting visualizations from the remainder of the list.

There are a number of useful tools for exploratory data analysis. Typical graphical techniques, apart from the ones mentioned above, are also: box plot, histogram, Pareto chart, star plot, mosaic plot, stem-and-leaf plot, run-sequence plot, scatter plot, multidimensional scaling, principal component analysis, to name but a few. Increasingly popular with the accumulation of data in many areas are also network visualization techniques.

2.1.2 Network Analysis

Complex network research, visualization, and knowledge discovery from networks addresses the challenge of information explosion and is now the main topic in many scientific fields. Fast Internet (the world wide web) growth, the emergence of social networks and other forms of messaging have triggered an ascent of vast data collections. Those are not only collections of isolated entities, but also aggregates of relations among stored objects. As a rule, relations contain a great amount of implicit information that uncover complex properties which could otherwise remain hidden. Network analysis [90, 91] is the study of relations between objects.

Network analysis is an exceptionally diverse field. It includes subfields such as: social network analysis [29], link analysis, graph mining, community detection [39], and others. The field is closely related to graph theory, data mining, machine learning, statistics, and spectral analysis.

In mathematical and computer science Graph theory [13] is a study of graphs—mathematical structures used to represent relations between objects. Formally, a graph G is a collection of vertices $V(G)$ (objects) and edges $E(G)$ (relations) that connect pairs of vertices $G = (V(G), E(G))$. A network is a graph with added information about vertices or edges and is a tool for abstraction of the world. Network theory is a subject of applied mathematics and physics and supplements the graph theory. It is used in many scientific fields, for example in computer science, biology, economy, and sociology.

Studying vertex degrees in graphs has received a lot of attention in the recent years. It was shown that a distribution of vertex degrees in the real-world networks is different from the random ones, what was assumed in the past. The latter have vertex degrees distributed closely around the mean, while the vertex degrees in the real-world networks are scale-free. This type of networks has a few vertices with a high degree (called hubs) and many vertices with only a few edges. It appears that the vertex degree follows a power law. Such networks are called scale-free networks [7].

Real-world networks are often an assembly of communities [47]. Communities are highly interconnected sets of vertices with only a few edges to other parts of the network. They play an important role in complex systems (supply chains, protein interactions, social networks, and others).

Many studies have been done in the field of network analysis. In conjunction with the experimental analysis and data mining, new methods were developed, producing fascinating results. Interdisciplinary projects merge ideas from statistics, machine learning, algorithms and other scientific fields with the exciting new science of networks.

2.2 Network Layout Optimization

The network layout optimization is a technique for graphical representation of the network (a set of vertices and edges) in a two- or three-dimensional plane. Numerous graph layout optimization techniques exist, each highlighting different relations in the data. A good visualization in general is one that indicates various features of the underlying data. The goal is thus to maximize the information gained by the use of a particular visualization technique. Choosing the best graph representation depends on the graph type and desired objectives. For example, two identical graphs are drawn in Figure 2.1. The left one explicitly indicates a tree structure, which is hidden in the right image.

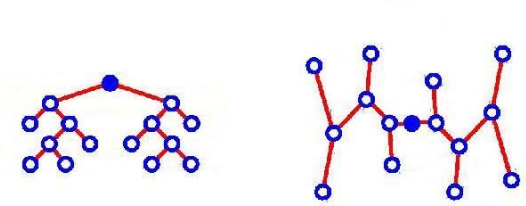


Figure 2.1: Different images of a graph can highlight different relations.

A graph layout optimization technique should assist an expert in easily interpreting the presented graph structure. Though most of the visualization requirements are subjective, a few generally accepted and measurable criteria exist: minimize edge crossings, minimize the image area, distribute vertices evenly without disproportionately long edges, and reflect the inherent symmetry. Some of these goals are mutually exclusive. In general, the problem of finding the best layout for the given objectives is an NP-complete problem [34].

Force-directed algorithms are heuristic layout algorithms that result in aesthetically pleasing visualizations. They position the nodes of a graph in a two- or three-dimensional space so that all the edges are more or less equal in length and there are as few edge crossings as possible. Strictly speaking, the name "force-directed algorithm" is appropriate when we talk about a procedure that involves calculating forces exerted on vertices, moving them along the direction of the forces, and repeating until the system comes to an equilibrium. Due to the aesthetically pleasing results, this class of algorithms was chosen for the basis of FragViz [107], the graph layout algorithm of choice for *model map* visualization. In principle, any force-based algorithm such as Eades's [35], Kamada-Kawai's [70], Fruchterman-Reingold's [45], Hu's [63], and others could be used with FragViz. We based FragViz on the method developed by Fruchterman and Reingold (F-R algorithm) [45]. The algorithm is an adaptation of the spring-embedder model of Eades for drawing undirected graphs with straight lines.

2.2.1 Overview of Force-directed Methods

The spring-embedder model of Eades could be thought of as set of pegs connected with springs [35]. The forces in springs are not modeled by the Hooke's law. Repulsive forces are applied among all pairs of vertices, whereas attractive forces are only applied to the neighboring vertex pairs. Kamada and Kawai also published their own version of the Eades algorithm in 1989 [70]. They modeled a graph as a spring system, but unlike Eades they used the Hooke's law. The concept of the ideal distance between two points, which should be proportional to the length of the shortest path between them, was added to the algorithm. The problem of graph drawing was designed as a process of reducing the energy of the system of springs which connect the rings. The total energy of the system is expressed by Equation 2.1, where p_i is

a position of a peg belonging to vector v_i; $k_{i,j}$ is a Hooke's constant between p_i and p_j; and $l_{i,j}$ is the optimal distance between vectors v_i and v_j.

$$\sum_{0 \leq i < j \leq |V|} k_{i,j} \times (|p_i - p_j| - l_{i,j})^2 \tag{2.1}$$

The energy is reduced using gradient descent. Each vertex is moved in the direction which decreases the energy. This step is repeated until the energy decrements achieved at each step fall below the specified threshold.

2.2.2 Fruchterman-Reingold (F-R) Algorithm

The F-R algorithm is based on two principles:

1. connected vertices should be drawn near each other and
2. vertices should not be drawn too close together.

The F-R algorithm can easily be explained with analogy. Imagine the vertices as atoms or celestial bodies influencing each other with repulsive and attractive forces. Those result in the movement of points. Similar to the method of Eades, repulsive forces are applied to all vertex pairs and the attractive ones only between the connected vertices. Although the system is based on the physical systems of springs and gravitation, using the word "force" in this case is not correct. In physics, a force is related to a change of velocity (acceleration) in the time unit. In the F-R algorithm, however, forces are used to compute the displacement of vertices. This distinction is crucial, considering that the true definition of force would result in dynamic equilibrium and that in this case a static equilibrium is needed. The pseudo code of the F-R algorithm is as follows:

```
area := W * L; { W and L are the width and length of the frame }
G := (V, E); { the vertices are assigned random initial positions }
l := sqrt(area / |V|)

function fa(d) := begin return d^2 / l end;
function fr(d) := begin return l^2 / d end;

for i := 1 to iterations do begin
    { calculate repulsive forces }
    for v in V do begin
        { each vertex has two vectors: .pos and .disp }
        v.disp := 0;
```

```
        for u in V do
            if (u ≠ v) then begin
                { Δ is short hand for the difference }
                { vector between the positions of the two vertices }
                Δ := v.pos - u.pos
                v.disp := v.disp + (Δ / |Δ|) * fr(|Δ|);
            end
        end
    end

    { calculate attractive forces }
    for e in E do begin
        { each edge is an ordered pair of vertices  .v and .u }
        Δ := e.v.pos - e.u.pos;
        e.v.disp := e.v.disp - (Δ / |Δ|) * fa(|Δ|);
        e.u.disp := e.u.disp + (Δ / |Δ|) * fa(|Δ|);
    end

    { limit the maximum displacement to the temperature t }
    { and then prevent from being displaced outside frame }
    for v in V do begin
        v.pos := v.pos + (v.disp / |v.disp|) * min(v.disp, t);
        v.pos.x := min(W / 2, max(-W / 2, v.pos.x);
        v.pos.y := min(L / 2, max(-L / 2, v.pos.y);
    end
    { reduce the temperature as the layout approaches a better }
    { configuration }
    t := cool(t);
end
```

Each iteration consists of three steps. First, the influence of the attractive force acting on each point is computed. Then all repulsive forces are added and finally, the displacement is limited by the temperature. The idea is to set the displacement limit (temperature) high in the beginning and then to gradually decrease it in each step to reach the equilibrium. This technique, called *simulated annealing*, is used to avoid getting trapped at the local minima in the first iterations.

Forces in F-R algorithm are defined as follows. Let constant l, which denotes optimal distance between two vertices, be defined as:

$$l = C \times \sqrt{\frac{area}{number\ of\ vertices}}, \tag{2.2}$$

where the constant C is determined experimentally. Vertexes should be evenly distributed in the frame. Attractive and repulsive forces then need to be in equilibrium at the distance l. Equations 2.3 and 2.4 give attractive and repulsive forces respectively.

$$f_a(d) = \frac{d^2}{l} \tag{2.3}$$

$$f_r(d) = \frac{-l^2}{d} \tag{2.4}$$

Figure 2.2 illustrates the forces and their sums versus distance. In the point where the sum of forces crosses the vertical axis the two forces are in equilibrium. This is exactly l, the ideal distance between two vertices.

Our implementation of F-R algorithm is written in the C++ programming language, using Qt libraries for data structures. Vertex coordinates are of the *double* type. Optimization is implemented in the 2-dimensional Euclidean space. Vertexes are initialized with random positions. A temperature cooling schedule is then computed for the given number of steps. Best results were achieved with a 2-phase linear schedule. In the first few steps (usually the first 20 steps), the temperature decreases rapidly. For the next several thousand steps in the second phase, the temperature decreases slowly until it reaches one pixel accuracy.

As proposed in the paper by Fruchterman and Reingold, a straightforward improvement of the algorithm was used to compute the attractive force only between pairs of neighboring vertices ($O(|E|)$ time complexity). Another major speed improvement was gained by computing repulsive forces only among pairs of vertices that are at most $2l$ apart (see Figure 2.3). The resulting time complexity of the algorithm when the distribution of vertices is approximately uniform is thus $O(|V|log|V| + |E|)$.

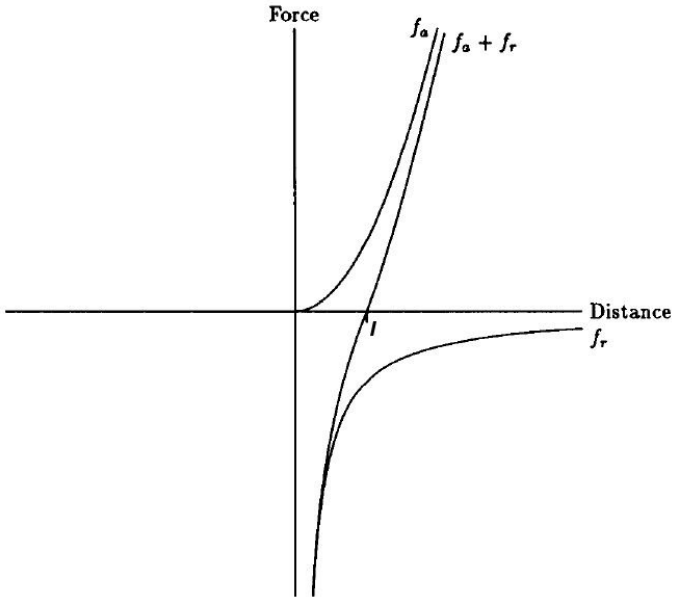


Figure 2.2: The sum of attractive (f_a) and repulsive (f_r) forces versus distance.

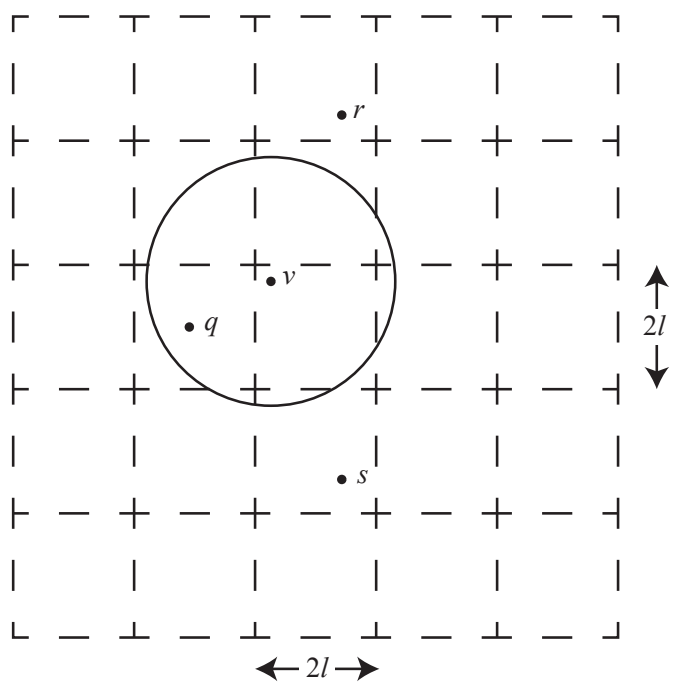


Figure 2.3: Calculate repulsive forces among vertices in close proximity only.

2.3 Visualization of Fragmented Networks

The network often consists of a large number of unconnected components, like the recently published yeast protein interaction network [9] and a drug similarity network [65], each with 160 and 240 unconnected components, respectively. Classical network layout techniques such as Fruchterman-Reingold [45], Kamada-Kawai [70] and Frick *et al.* [41] algorithms arrange unconnected components arbitrarily, which can wrongly suggest a relation between otherwise unrelated components.

For illustration, consider the network from Figure 2.4, which depicts four compo-

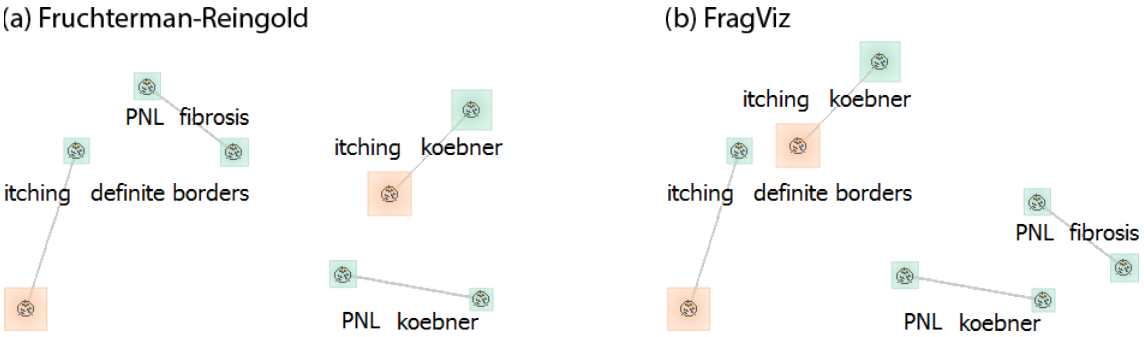


Figure 2.4: Four components from the *model map* of the Dermatology data set from the UCI Machine Learning Repository. The layout was optimized by a standard Fruchterman-Reingold algorithm (a) and by FragViz (b). FragViz optimization additionally used the information on vertex distances.

nents from the network of models in Figure 5.12. Here, nodes represent classification models learned on the Dermatology data set from the UCI Machine Learning Repository. Edges reflect a similarity in their predictions and as it appears in this case, the models built on a similar feature set are similar themselves and thus connected. From the layout in the Figure 2.4.a with an arbitrary component placement, one could (incorrectly) conclude that the models accounting features "itching" and "definite borders" are more similar to those accounting "PNL" and "fibrisis" than to other models in the graph. Misinterpretations like this can be avoided by displaying the network's main component, if one exists, separately, and then listing other (smaller) components. This type of display has been used, for instance, in the recently published disease gene network [48].

We introduce a generally applicable technique called FragViz for placing the components according to the background data on their similarity. Rendering a network from Figure 2.4.a by our algorithm yields the layout in Figure 2.4.b, from which we can infer that there is a strong relation between four models accounting features "PNL," "fibrosis," and "koebner". These are indeed correct relations.

FragViz uses a two-step network layout optimization procedure. It first applies the standard Fruchterman-Reingold algorithm separately on each unconnected component to optimize the layout of its vertices. Then it optimizes the global placement and orientation of components using a semi-physical model where the forces between components are inferred from similarities between the corresponding vertices in these components.

The data on similarity of the network vertices can either come from the same data source used to infer the structure of the network, or can be provided by supplying additional information. Most often, the network's structure itself is derived from the *scored* relations between objects (*e.g.* the correlation in expression of two genes [101], the degree of SNP synergy in phenotype prediction [93], the number of disorder-specific genes shared by two diseases [12]). Edges then connect pairs of vertices for which the corresponding score exceeds some user-defined threshold. In such cases, the vertex pair similarity scores can be used as additional data for our procedure. If relations in the graph are not obtained by imposing thresholds on numerical data, other data source can be used to describe the vertex similarities. For instance, in the experimental study reported in Chapter 2.3.3 we show a protein-protein interaction network in which the vertex similarities are computed based on the biological function of the proteins.

2.3.1 Optimization Algorithm

The input to the FragViz method is a list of network components and a matrix of (dis)similarities between the network's vertices. FragViz first uses a network layout optimization technique, like Fruchterman-Reingold algorithm [45], to determine the placement of vertices within each of the connected components. Then, it finds a

placement of components which reflects their mutual similarities. It is this second step that is an original contribution of our method, and which we describe below in detail.

Formally, we are given a graph $\mathcal{G} = (V, E)$ which consists of p disjunct components $V = \bigcup_{k=1}^{p} V_k$, and a $|V| \times |V|$ dissimilarity matrix D. The internal layout of each component V_k is fixed and given by positions of its vertices inside its own fixed coordinate system. We will denote the position of vertex v_i by $\mathbf{v_i}$. We also assume that the internal coordinate systems are centered, *i.e.* $\sum_{v_i \in V_k} \mathbf{v_i} = \mathbf{0}$ for each component V_k.

The task is to find the placement $\mathbf{c_k}$ and orientation $\boldsymbol{\phi_k}$ of coordinate systems for all components which reflect the given dissimilarities D.

We will base the method on a physical metaphor. Imagine each component as a board with vertices as pegs. Pegs from different components are connected with springs of different lengths corresponding to the given dissimilarities D. The nature (or, in our case, a computer simulation) "optimizes" the system by finding the lowest energy configuration of the boards (components).

Assume that all vertices have equivalent mass m. The mass of the component V_k is

$$m_k = |V_k| m \tag{2.5}$$

and component's moment of inertia is

$$I_k = m \sum_{v_i \in V_k} ||\mathbf{v_i}||^2. \tag{2.6}$$

The force between a pair of points (v_i, v_j) is defined by Hooke's law,

$$\mathbf{F_{ij}} = (d_{ij} - ||\mathbf{g_i} - \mathbf{g_j}||) \frac{\mathbf{g_i} - \mathbf{g_j}}{||\mathbf{g_i} - \mathbf{g_j}||}, \tag{2.7}$$

where $\mathbf{g_i}$ and $\mathbf{g_j}$ are positions of vertices in a global coordinate system,

$$\mathbf{g_i} = \mathbf{v_i} + \mathbf{c_k}, \tag{2.8}$$

where k is such that $v_i \in V_k$.

Let $\mathbf{F_i}$ be the sum of forces acting on vertex v_i:

$$\mathbf{F_i} = \sum_{v_j \in V} \mathbf{F_{ij}}. \tag{2.9}$$

The force causes linear acceleration,

$$\mathbf{a_k} = \frac{\sum_{v_i \in V_k} \mathbf{F_i}}{m_k}, \tag{2.10}$$

and angular acceleration,

$$\boldsymbol{\alpha_k} = \frac{\sum_{v_i \in V_k} \mathbf{F_i} \times \mathbf{v_i}}{I_k}, \tag{2.11}$$

of the component. We shall assume infinite friction, so the component does not retain any momentum. At each instance, the component moves by a distance proportional to the linear acceleration, $\Delta \mathbf{c_k} \sim \mathbf{a_k}$, and rotates by an angle proportional to the angular acceleration, $\boldsymbol{\Delta} \phi_\mathbf{k} \sim \alpha_\mathbf{k}$, so

$$\Delta \mathbf{c_k} \sim \frac{\sum_{v_i \in V_k} \mathbf{F_i}}{|V_k|} \tag{2.12}$$

and

$$\Delta \phi_\mathbf{k} \sim \frac{\sum_{v_i \in V_k} \mathbf{F_i} \times \mathbf{v_i}}{\sum_{v_i \in V_k} ||\mathbf{v_i}||^2}. \tag{2.13}$$

These equations allow for a computer simulation of the physical process. Starting from a random placement of components, we iteratively compute the forces $\mathbf{F_i}$ and move and rotate the components accordingly until the system reaches an optimum in which all $\mathbf{F_i}$ are negligible.

2.3.2 Approximate Solution

We can speed up the computer simulation by first computing the positions of components and then rotating them in place. The result is only approximately optimal with regard to the total stress (Equation 2.7), yet we will experimentally show that the difference is negligible.

For positioning the components, the approximate method measures and optimizes distances between components rather than the distances between vertices. We define the distance between components V_k and V_l as the average of distances between the corresponding vertices, similar to average linkage in hierarchical clustering analysis [103]:

$$\delta_{kl} = \frac{1}{|V_k||V_l|} \sum_{\substack{v_i \in V_k \\ v_j \in V_l}} d_{ij}. \tag{2.14}$$

The task is then to find the positions in a two-dimensional plane in which the distance between every pair of component centers $\mathbf{c_k}$ and $\mathbf{c_l}$ matches the given δ_{kl} as close as possible. This approach is much faster than the simulation from the previous section since the computation of all pairwise distances at each step of optimization is replaced by a single such computation in Equation (2.14). This translates the problem of placing the components into the familiar multidimensional scaling problem (MDS). There exist many efficient solutions of the MDS, such as, for

instance, SMACOFF [84], which optimizes the overall energy of the system without computing its gradient, the force defined in Equation (2.7).

By considering only the centers of components, MDS ignores their sizes, which can cause the components to overlap. This can be fixed by introducing a scaling factor between the global coordinate system and the internal coordinate systems of components by replacing the Equation (2.8) by

$$\mathbf{g_i} = \mathbf{v_i} + K\mathbf{c_k}. \tag{2.15}$$

The scaling factor is equal for all components and should be such that the components are as large as possible without excessive overlapping. A simple rule of thumb is to use the ratio between the average size of components $\bar{v}$ and the average distance between them, $\bar{g}$, so

$$K = \bar{v} \ / \ \bar{g} \tag{2.16}$$

where

$$\bar{v} = \frac{1}{p}\sum_{k=1}^{p} \frac{1}{|V_k|(|V_k|-1)} \sum_{\substack{v_i, v_j \in V_k \\ i \neq j}} ||\mathbf{v_i} - \mathbf{v_j}|| \tag{2.17}$$

and

$$\bar{g} = \frac{2}{p(p-1)} \sum_{k<l} ||\mathbf{c_k} - \mathbf{c_l}||. \tag{2.18}$$

For the rotation of components we use the original vertex-wise definition of force in Equation (2.7) computed in the scaled coordinate system, defined in Equation (2.15). We apply the same procedure as in the exact simulation, except that we only compute the rotation without the translation. To avoid ending up in local minima, we use simulated annealing where the component can also rotate in the "wrong direction," with the probability of doing so decreasing with time. Although this optimization recomputes the pairwise distances between all vertices at each step, it is not overly time-consuming since it requires only a small number of iterations.

In the remainder of the section we only show layouts optimized by the approximate method.

2.3.3 Case Studies

This section presents several case studies, first published as a part of the original paper about FragViz [107]. The expert knowledge and comments were provided by Dr. Minca Mramor.

The performance of the proposed algorithm was assessed on four different networks (N1, N2.1, N2.2, and N3) showing relations between genes which were most differentially expressed in the leukemia gene expression data set [49]. The original data set

includes 4,860 genes whose expression was measured using DNA microarrays in 72 tissue samples classified either as acute lymphoblastic leukemia (ALL, 48 samples) or acute myeloid leukemia (AML, 25 samples). For N1, N2.1, and N2.2 we selected 1,025 differentially expressed genes with expression levels significantly smaller or larger (p-value < 0.01) according to Student's t-statistic with respect to the null distribution of the statistic. The null distribution was obtained by randomly permuting the class labels and calculating the t-statistic for all the genes. Network N3 was built with 131 out of 4,860 originally measured genes for which the information on their protein interactions was available in the MIPS mammalian protein-protein interaction database [94]. In the visualizations, genes represented with solid circles were significantly over-expressed in the ALL samples, while genes shown as hollow circles had higher expression in the AML samples.

Based on different means to estimate the gene similarity, we defined four distinct gene networks:

- N1—biological function similarity score: the similarity of genes relates to their biological functions and was calculated based on their membership in canonical biological pathways using the Jaccard index [109]. The information on the membership of genes in biological pathways was acquired from the Molecular Signature Database [95] (C2 collection, canonical pathways). Figure 2.5 shows the network where the similarity threshold was set to 0.7 and all the unconnected genes were ignored.

- N2.1—Huttenhower similarity score: the similarity between genes as computed by [64] using the information on all publicly available gene expression and protein interaction data, combined with prior knowledge from the Gene Ontology, KEGG, HPRD, and other biological databases. Similarity scores above 0.999 for the leukemia genes were used to build the network. Only the genes connected to at least one other gene are included (Figure 2.6).

- N2.2—Huttenhower similarity score: for the N2.2 network, we used the same similarity scores and threshold as in N2.1 (the Huttenhower *et al.*, 2009 similarity score). As opposed to N2.1, N2.2 also includes isolated vertices (genes which are not connected to any other gene) in order to observe the similarity of all the differentially expressed genes (Figure 2.7).

- N3—protein-protein interaction network: the leukemia genes were connected into the network based on their protein interactions from the MIPS mammalian protein-protein interaction database [94]. In addition, we used the biological function similarity score (described under N1) for positioning the interacting protein components based on the similar biological functions of the proteins comprising them (Figure 2.8).

The goal of FragViz is to find the network layout in which the arrangement of components uncovers new insights on relations between them and their constituents.

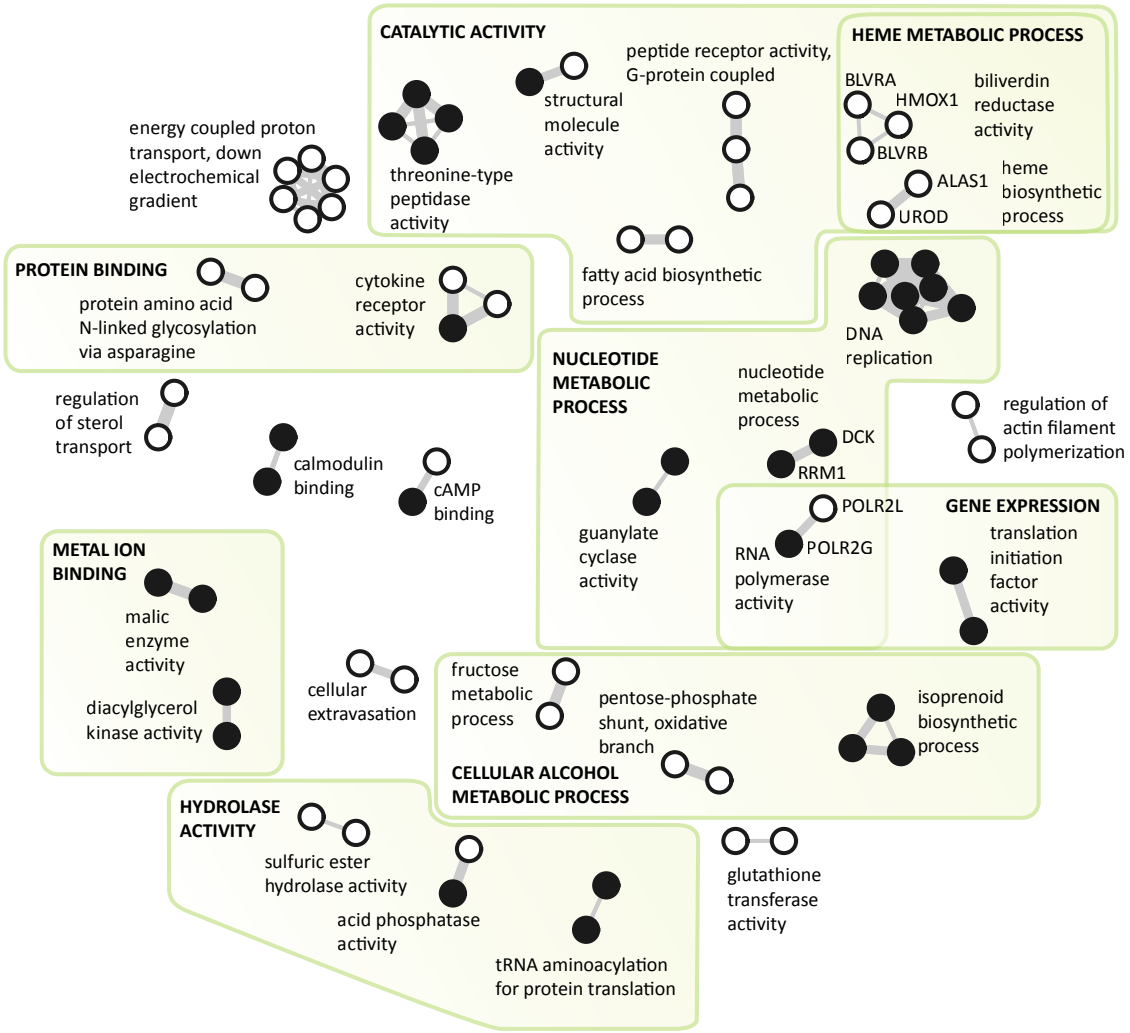


Figure 2.5: The network (N1) of the significantly differentially expressed genes from the leukemia data set, where the similarity between the chosen genes was calculated based on their co-membership in biological pathways. Only components with at least two vertices (similarity threshold equal to 0.7) were included in the network. Genes represented with solid circles were significantly over-expressed in the ALL samples and genes shown as empty circles had higher expression in the AML samples. The individual components were named according to the prevailing Gene Ontology annotation. Components were grouped and labeled manually by the expert.

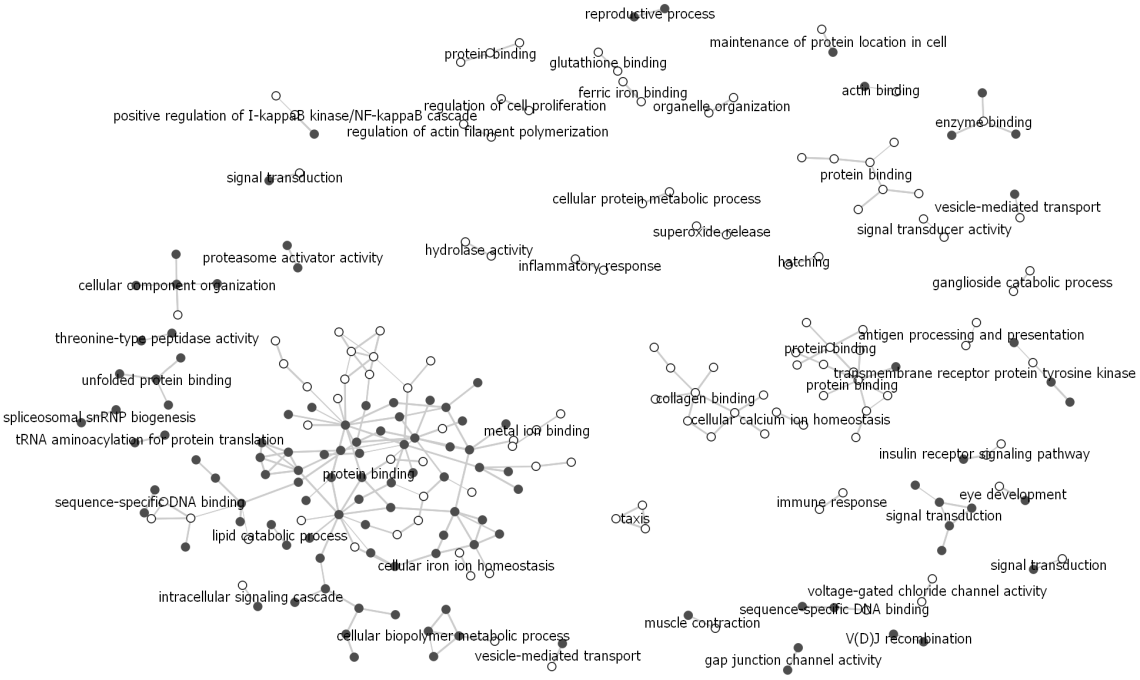


Figure 2.6: The network (N2.1) of the most differentially expressed genes from the leukemia data set. The similarity matrix of the chosen genes was taken from the recently published work of Huttenhower *et al.*, 2009. The genes represented with solid circles were significantly over-expressed in acute lymphoblastic leukemia and the genes shown as empty circles had higher expression in acute myeloid leukemia.

We evaluated the method in an experimental study which considered FragViz visualization of the leukemia gene networks N1, N2.1, N2.2, and N3. For additional assistance to the domain expert, the network components were named according to their most specific term from biological process or molecular function aspect of Gene Ontology [14].

Leukemia Gene Network (N1)

Our goal was to obtain a clear visualization relating the most important genes and their biological functions for two major types of acute leukemia, yielding insight and valuable clues about the disrupted biological processes and pathways in leukemic cells. Solid vertices in Figure 2.5 represent genes significantly over-expressed in the ALL samples, while empty circles are genes that had a higher expression in the AML samples.

FragViz allows for the exploration of biological processes related to acute myeloid and acute lymphoblastic leukemia on different levels, from specific to more general ones. In Figure 2.5, additional Gene Ontology terms were assigned to groups of clusters which were determined manually by an expert to elucidate the disrupted biological pathways on a more general level, as they cover a higher number of differentially expressed genes. These ontological terms apply to all the genes in the

Figure 2.7: The network (N2.2) of the most differentially expressed genes from the leukemia data set as the network in Figure 2.6, but including the isolated vertices (genes not connected to any other gene) in order to observe the similarity of all the differentially expressed genes.

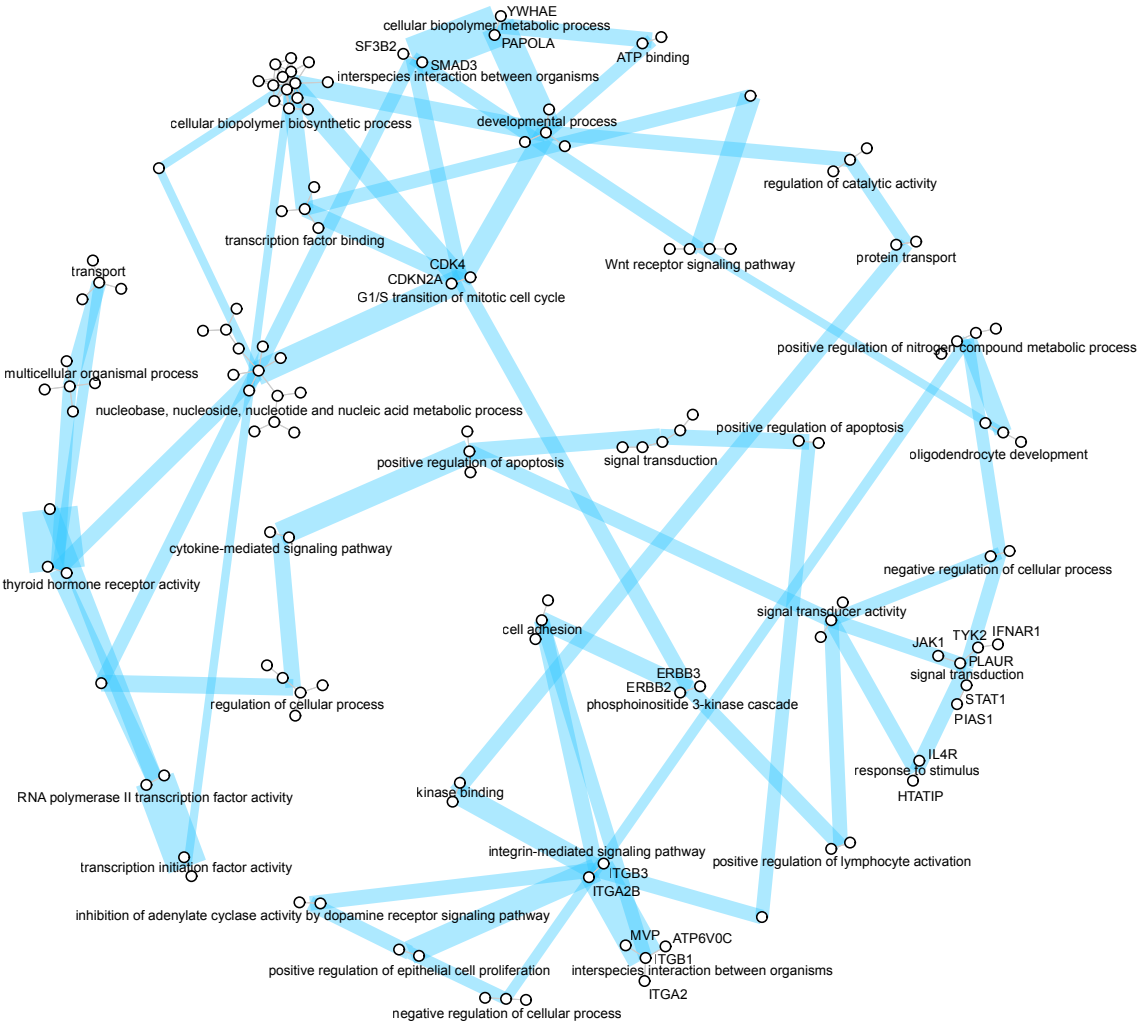


Figure 2.8: The network (N3) of genes from the leukemia data set. Vertexes are connected based on their protein interactions from the MIPS database. Only the interactions with the confidence level equal to 1 are shown. The (dis)similarity matrix was added from a different data source and relates to genes' biological functions. The individual components and their clusters are named according to the prevailing Gene Ontology annotation. Blue lines are drawn in the background, connecting each component with two most similar components. Line width corresponds to component similarity.

marked areas and are significantly enriched with a p-value < 0.01. The components of the graph that are close to each other have similar biological and/or molecular functions, according to Gene Ontology, demonstrating similarity between genes constituting them.

For example, the "guanylate cyclase activity," the "nucleotide metabolic process," the "RNA polymerase activity," and the "DNA replication" components in Figure 2.5 all connect genes significantly over-expressed in acute lymphoblastic leukemia. All of these genes have a function in nucleotide metabolism and DNA biosynthesis. It is well known that in lymphoblastic cells, the enzymes responsible for nucleotide metabolism enabling excessive proliferation of transformed cells are several-fold more active [102]. Moreover, some of the pathways active in nucleotide metabolism, for example de novo purine synthesis (DNPS), have been recognized as important targets of antileukemic agents (*e.g.* methotrexate, mercaptopurine). In combination with other therapeutical agents, these drugs have improved survival of children with ALL to an overall cure rate of approximately 80 percent [97]. The network shown in Figure 2.5 clearly demonstrates this characteristic of acute lymphoblastic leukemia.

Huttenhower Similarity Network (N2.1 and N2.2)

The N1 and both N2 networks contain the same 1,025 differentially expressed genes from the leukemia data set. However, in N2.1 and N2.2 a combined gene distance score was used, computed from multiple biological data sources (*e.g.* gene expression, protein-protein interactions, and biological function) as proposed by Huttenhower *et al.* [64]. N2.1 shows only vertices with at least one edge. N2.2 also includes isolated vertices (genes not connected to any other gene) in order to observe the similarity of all the differentially expressed genes.

As in the N1 network, most of the graph components in N2 networks (Figures 2.6, 2.7) connect genes that are over-expressed in one of the two investigated kinds of leukemia (all genes in the component are of the same color). One can observe that the genes significantly differentially expressed in the two investigated leukemias cluster together (Figures 2.6, 2.7). This reflects the well-known phenomenon that not only individual genes, but whole processes and pathways are disrupted in cancer cells [52]. In Figure 2.7, the empty circles (AML) are clustered in the right part of the graph and the solid ones (ALL) in the left part, again demonstrating that expression changes in cancer tissues are disrupted on the level of pathways and processes.

For example, the genes in components "spliceosomal snRNP biogenesis," "tRNA aminoacylation for protein translation," "sequence-specific DNA binding," and the nearest genes in the component "protein binding" participate in processes of cell proliferation. All these genes have a higher expression in ALL samples. Excessive cell proliferation is a characteristic of all leukemic cells. However, previous studies [115, 69] have shown that the proliferative index of ALL cells is significantly higher compared to AML cells.

Since the distance information is used to adjust the position of unconnected components, the layout allows for the exploration of the data on different levels, using genes from a single component or from clusters of biologically related components.

Protein-protein Interaction Network (N3)

The placement of unconnected components in a fragmented network can be optimized using the vertex distance information from a source other than that used in the inference of network structure. For example, the N3 network (Figure 2.8) shows the protein-protein interactions for the leukemia genes from the MIPS database. The network is fragmented into many smaller unconnected components. We used the biological function similarity score to calculate the similarities between the components and optimizing the network layout.

Several gene products (proteins) that lie close to each other in the FragViz optimized network (Figure 2.8) are actually in interaction, as is reported in Human Protein Reference Database (HPRD) [73], another public repository that stores protein-protein interactions identified by experimental results.

For example, in HPRD, the protein Integrin beta 3 (*itgb3*) is in interaction with protein Integrin beta 1 (*itgb1*). Also, proteins Poly A polymerase alpha (*papola*) and *smad3* are both in interaction with protein *smad2*. According to HPRD, protein interactions also exist among the component which includes proteins *il4r* and*htatip* and the near-lying component. To outline them in the network, the vertices that correspond to these proteins (in Figure 2.8) are labeled accordingly. While our goal was not to use network layout optimization for protein interaction prediction, the cases mentioned here demonstrate the potential utility of different data sources in network layout optimization.

The similarity between network components can be visualized by blue lines in Figure 2.8. Each component is connected to two most similar components and the line width represents the magnitude of the similarity. In Figure 2.8, most connected components are placed close to each other. In some cases though, similar components are positioned apart. This is, first and foremost, a technical problem: the optimization gets stuck in a local optimum. But in fact, this may happen when two components belonging to different clusters of components nevertheless share a common function or when some component essentially belongs to two clusters. For example, genes in the component "G1/S transition of mitotic cell cycle" influence gene expression, as do most of the genes in the nearest cluster of components. But the same component also participates in the apoptotic pathway which is reflected in its connection with the "phosphoinositide 3-kinase cascade" component, a representative of components related to the apoptotic processes.

2.3.4 Performance Comparison

Table 2.1 compares the running times of simulation for six different layout optimization approaches: the Fruchterman-Reingold algorithm, the exact and approximated method of FragViz, MDS, and two applications of the clustered graph visualization.

Clustered graphs include groups of vertices that are somehow related. Clusters can be determined by observing the density of mutual connections between vertices or based on other data describing the vertices. Eades *et al.* [36] proposed a method for plotting clustered graphs, which models them in terms of four layers: the entire graph, clusters, abridgments, and pictures (groups of points in a particular projection). A corresponding model includes forces between connected vertices, between all vertices in each cluster, and between meta-vertices representing entire clusters.

The clustered graph visualization needs a graph G' and a cluster tree T'. We transformed the original graph G and a dissimilarity matrix D to a clustered graph $C' = (G', T')$ in two different ways:

- Eades 1: the component structure G was embedded in a two-level cluster tree (root-components-data objects) T' in which every component from G represents a cluster in the first level of T'. Object dissimilarities D were used as weights in a complete graph G'.
- Eades 2: the original graph was retained ($G' = G$) and a hierarchical clustering method was applied on the dissimilarity matrix D to construct a cluster tree T'. Four different linkage functions were tested (average, single, complete, and Ward's linkage). Since they all produced similar results, we report only on the performance of average linkage.

For N3, the MDS was run on the dissimilarity matrix data. We used the standard SMACOFF algorithm for MDS; an exhaustive comparison of various heuristic enhancements is beyond the topic of this research.

All measurements were conducted on a desktop PC, with Intel Core 2 Duo 2.20GHz processor and 4 GB of RAM, using the 64-bit Windows 7 OS. The results represent an average of $10,000$ runs of the algorithms on the N1-N3 networks, starting from random positions of vertices.

Table 2.1: Average layout optimization time in seconds for all four networks.

network	F-R	FragViz	FragViz (approx)	MDS	Eades 1	Eades 2
N1	0.4	33	6	36	3	1
N2.1	1.3	63	6	64	31	2
N2.2	8	301	240	320	410	29
N3	1.1	76	14	55	8	1.5

The Fruchterman-Reingold algorithm is by far the fastest, but it uses less data than the others and the resulting projections are much less informative. Running times of Eades 2 are comparable to those of Fruchterman-Reingold. This was expected as both approaches run on a similar graph. Eades 1 employs a complete graph, which makes it much slower. On large networks, Eades 1 (N2.2) is even slower than MDS. The running times of FragViz simulation are similar to those of MDS, which is also expected. The approximate method runs much faster, except for the large network N2.2, where most vertices are unconnected, which essentially translates the visualization problem to MDS.

Table 2.2 compares the quality of layouts in terms of Pearson correlation between the vertex similarities and their distances in the projection. Although the approximate method runs much faster than the simulation, the decrease in quality is small. Moreover, the approximate method sometimes outperforms the exact one, which suggests that the optimization can get trapped in a local minimum.

For all four networks, the correlation coefficients of the FragViz algorithms are very similar. The correlation was always lower with the F-R algorithm and, for three out of four networks, the highest correlation was obtained with MDS. In one of the compared networks (N2.2), MDS performed slightly worse than the approximation, suggesting MDS got trapped in a local minimum. As expected, when the vertices were arbitrary placed in the graph, the correlation between the position of vertices in the graph and their actual distances is close to 0.

Clustered graph approaches (Eades 1 and Eades 2) were in general faster than FragViz, but performed worse in terms of layout quality. Eades 2 performed better than Eades 1 on smaller graphs (N1, N2.1, and N3), whereas Eades 1 had a high correlation for a large network (N2.2). However, Eades 1 approach is not appropriate for analyzing large fragmented networks as it is prohibitively slow.

Note that the compared algorithms pursue different goals. The tests were run on data suitable for the method presented in this chapter, while in other contexts another method could give better results. In particular, clustered graph methods could not be directly applied to the original data, so its results depend on the proposed transformation of the original problem.

Table 2.2: Pearson's correlation between elements of the gene distance matrix and the Euclidean distance between the corresponding vertices in the two-dimensional network layout.

network	F-R	FragViz (sim)	FragViz (approx)	MDS	random	Eades 1	Eades 2
N1	0.311	0.391	0.380	0.415	0.007	0.173	0.215
N2.1	0.086	0.290	0.302	0.654	0.002	0.009	0.156
N2.2	0.401	0.591	0.609	0.593	0.006	0.391	0.043
N3	0.179	0.224	0.285	0.361	0.060	0.092	0.199

2.3.5 Impact of Network Fragmentation

We also investigated the behavior of layout optimization methods with respect to the degree of network fragmentation. We constructed 1,000 networks of the most differentially expressed genes from the leukemia data set (visualized in Figure 2.5) with the similarity threshold required for an edge from 0.0 (the graph is fully connected) to 1.0 (no edges between vertices). Figure 2.9.a shows the correlations between the network layout and the (dis)similarities matrix for the F-R, MDS, and FragViz algorithms. Figure 2.9.b shows how the *average local clustering coefficient* [114] and the number of components change with different threshold values.

FragViz and F-R algorithms are equivalent when the network consists of only one component (threshold values lower than 0.1). For the F-R algorithm, the correlation decreases when the network gets more fragmented. However, when the fragmentation increases (threshold value greater than 0.2), the correlation score of the FragViz algorithm increases and rises above the best score obtained by the F-R algorithm. Correlation for MDS does not depend on the threshold.

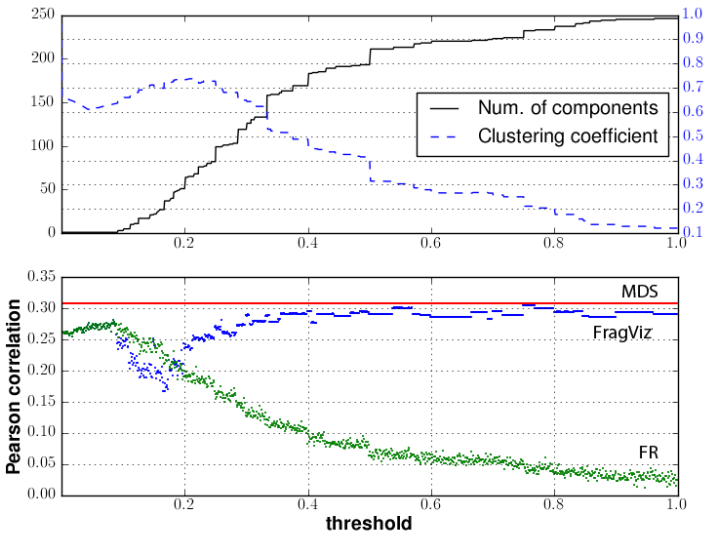


Figure 2.9: Influence of the selected similarity threshold on the layout optimization. Biological function similarity score was used on input. The horizontal axis measures the connectedness of the network, where 0 represents a complete graph and 1 means the graph has no edges.

Chapter 3

Orange Net Explorer

Most existing tools for network analysis are limited to drawing networks and computing their basic general characteristics. The user can not interactively and graphically manipulate the networks, select, and explore subgraphs using other statistical and data mining techniques, add and plot various other data within the graph, and so on. This chapter introduces a tool we developed to address these challenges, a widget for exploration of networks within the general component-based environment Orange.

3.1 Related Tools

Tools for graph drawing and computing graph characteristics are readily available, either for free (Pajek [10], NetworkX [51], Graphviz [37], Gephi [8], Network Workbench [92]) or in commercial packages (*e.g.* Net Miner [27]). An overview of such tools and their capabilities is in Table 3.1.

Table 3.1: An overview of the software for network analysis.

	Open-source	Interactive UI	Scripting interface (in Python)
Pajek			
NetMiner		×	
NetworkX	×		× (×)
Graphviz	×		× (×)
igraph	×		× (×)
statnet	×		×
Gephi	×	×	
Network Workbench	×	×	
Orange Network	×	×	× (×)

Most graph visualization and analysis tools analyze graphs as a whole (for instance Pajek, igraph [25], and statnet [53]). Pajek is a package for the analysis of large networks, while statnet's specific focus is a simulation of exponential random graph models and statistical analysis. Much less effort has been put into making graph analysis software interactive, by allowing local exploration of the graph, testing how the graph structure depends on its construction parameters, letting the user extract data from subgraphs and use it for further analysis, and so on.

For example, edges in most graphs are abstractions of numerical relations. In genetic networks, for instance, two genes are connected if they are sufficiently co-expressed. Two journal papers are related if they share a sufficient number of keywords, and people in a social network can be connected for having enough shared interests or friends. We would wish for a tool in which changes in the graph construction parameters (*e.g.* connection thresholds) can be immediately observed in the graph itself, since sometimes precise tuning is required to get a graph which reveals interesting relations.

A similar problem is filtering of the graph's nodes. A geneticist might wish to focus on certain genes, or a computer network expert might want to plot the graph with only the major nodes or, for another purpose, with every single client. It would be helpful if they were able to perform such operations interactively, in a script, or both.

Interactive tools are also required to explore the local graph structure itself. Questions like "Which computers are at most two connections away from those infected?" are asked more often than "How well do the degrees of the graph's nodes match the power law?".

Finally, exploring the graph often leaves us with a subgraph, or a subset of nodes that we want to explore further. After identifying a strongly connected component of a social network, we might want to learn about how this group differs from the general population. If we find a set of drugs with similar effects on the organism, we might want to discover what is typical of their active parts. Such issues can be answered with less effort if the network visualization tool is integrated into a larger statistical, machine learning, or data mining framework.

We developed a collection of widgets and Python modules for interactive network visualization and analysis for the data mining toolbox Orange [30]. There are some excellent Python packages for network analysis (*e.g.* Graphviz and NetworkX). However, Orange network widgets are, to our knowledge, the only Python software for interactive network visualization and exploration. A brief introduction to the Orange framework is in Appendix A.

3.2 Explorative Network Analysis (in Orange)

`Net Explorer` is a widget for Orange Canvas which visualizes graphs and lets the user explore them. We also implemented several auxiliary widgets for reading the network, constructing it from data, and for computing the general statistical properties of the network.

The network object is stored in the data structure defined in the module NetworkX. We added some new algorithms (*e.g.* community detection and frequent graph pattern discovery) and methods to integrate it seamlessly into the Orange environment.

3.2.1 Data Preparation

Orange includes three widgets which load or construct graphs. `Net File` reads the popular Pajek format, Graph Modeling Language (GML [59]), and a NetworkX graph in Python's pickle format. The objects corresponding to the nodes can be described with vectors of continuous, discrete, or textual variables (or, in machine learning terminology, attributes). The user can supply additional data in different formats: tab-delimited, comma-separated, or in several other formats used by other software. The second optional file contains the data about the edges. The data needs to include two columns marked u and v with indexes of node pairs, and an arbitrary number of other columns with the data about the given pair.

`Net from Distances` constructs a graph from a distance matrix. Edges are defined by several criteria: connect nodes with distances within the specified interval, connect each node with its k-nearest neighbors, or nodes with distances up to the i^{th} percentile. The widget provides a histogram with the number of node pairs at each distance. The distance matrix can come from various sources: a widget that reads it from a file, one that computes distances between objects, and others. `Net from Distances` also includes some basic filters: output the entire graph, the largest component, or nodes with at least one edge. The distance matrix can include vectors of attributes about the objects. Such data is copied to the constructed network.

The last widget for construction of networks is `SNAP`, which downloads the network from the Stanford Network Analysis Project graph library (`http://snap.stanford.edu/data`).

3.2.2 `Net Explorer` Widget

3.2.2.1 Input and Output Signals

The `Net Explorer` widget has a number of input and output slots to exchange data with other widgets. The inputs are:

Network: the network to plot.

Items: data about the nodes (overrides any such data that is already present in the network).

Item Subset: a list of nodes to be marked in the graph.

Distances: distances between graph nodes, required by some layout optimization algorithms.

Net View: a custom plug-in widget for extending the `Net Explorer`.

The widget can output a selected subgraph and the corresponding distance matrix or descriptions of the marked, selected, or unselected nodes.

3.2.2.2 Layout Optimization

`Net Explorer` supports several layout optimization algorithms:

No optimization: if the network object contains the nodes placement data (this is supported, for instance, in the Pajek format), they are placed accordingly.

Random: the nodes are scattered randomly.

Fruchterman-Reingold (F-R): the algorithm [46] positions pairs of connected nodes to a certain fixed small distance and the unconnected ones to the fixed large distance. A simulated annealing algorithm is used to optimize the layout.

F-R Weighted: a variation of the above which also considers the edge weights: the larger the weight, the smaller the desired distance between the two nodes.

F-R Radial: an F-R-type algorithm which places a node selected by the user at the center and optimizes the layout around it. The optimization procedure ensures that nodes with shorter paths to the central node are closer to it than those with longer paths.

Circular Crossing Reduction: a local optimization algorithm which arranges the nodes on a circumference and minimizes the number of edge crossings [11].

Circular Original: nodes are placed on a circumference in the order they are given.

Circular Random: the nodes are placed on a circumference in random order.

FragViz: the FragViz [107] algorithm for visualization of networks that consist of multiple unconnected components. The algorithm combines the standard Fruchterman-Reingold algorithm for laying out individual components with an MDS-style algorithm for placement and rotation of components.

MDS: the SMACOF [84] multidimensional scaling algorithm using stress-majorization. The stress model simulates a set of balls corresponding to graph nodes that are connected by springs. The lengths of the springs correspond to the desired distances between the graph nodes.

Pivot MDS: an approximation of the classical MDS algorithm, where k pivots (columns) are randomly selected to reduce the distance matrix size [15].

The last three algorithms require information on preferred distances between the nodes provided on a separate input slot. The last two algorithms place nodes according to the provided distances only, disregarding the network edges.

The user can specify the number of iterations of the optimization procedure where applicable. The default is set so that the optimization is expected to take approximately five seconds. With a higher number of iterations, the optimization takes longer but the results are often considerably better.

In practice, if the number of nodes is large, the most applicable algorithms are Fruchterman-Reingold algorithms, with the Random placement used to reinitialize the optimization if it gets stuck in a local minimum. The user can also move individual nodes or groups of selected nodes after or even during the layout optimization.

3.2.2.3 Setting Visual Parameters

The widget allows the user to set a number of parameters (see, for instance, Figure 3.4(b)). The widget can print a label with the meta-data beside each node. When the number of nodes is excessive, the user can reduce the clutter by reducing the number of shown variables, or by printing out only the data about the node below the mouse pointer. Another option is to annotate only the marked nodes. It is also possible to print some data at each node, and display more on the mouse hover over the node.

Nodes can be colored or sized according to values of the corresponding objects' attributes. The widths of edges can correspond to their weights, and can be colored according to the value of the selected attribute from the edge descriptions.

3.2.2.4 Marking and Selecting

The widget uses a two-stage procedure for node selection. This allows for a very flexible manipulation of subsets of nodes (see, for instance, Figure 3.4(a)). Nodes can be marked or selected or both. *Marking* is often a pre-stage to *selecting* or *deselecting*. Selected nodes are shown by filled circles (as opposed to the rest, which are hollow), whereas marked nodes have a black border.

Nodes can be *selected* manually or with rectangle tool. Another way to select nodes is to add or remove the marked nodes to or from the selection, or to replace the current selection with the marked nodes. The selected nodes can be dragged to manually enhance the layout. The user can also hide the selected or the unselected nodes, and rerun the optimization. The selected subgraph, the data or distance sub-matrix about the corresponding objects, can be sent to other widgets.

The nodes can be *marked* upon values of attributes of the corresponding objects or upon network properties. To select local portions of the graph, the user can mark the neighbors of a node pointed to by the mouse, or the neighbors of the selected nodes up to a certain distance (*e.g.* up to three edges away). Hubs, poorly connected nodes, and similar local phenomena can be observed by marking the given number of the most connected nodes, the nodes with more or less edges than the given number, or with more connections than their average neighbor or than any of its neighbors. Finally, the set of marked nodes can be specified by the data sent from another widget using the input slot Item Subset.

3.2.2.5 Plug-ins

A network plug-in is a widget that controls which part (subgraph) of the original network is visualized in the `Net Explorer`. It is particularly useful for visualizing large networks. A plug-in widget connects to the `Net Explorer`'s "Net View" signal.

`Net Inside View` widget is an example of a basic plug-in. It provides a local view on the network. On selection, it smoothly moves selected node to the center, and hides distant nodes (nodes that are more than k edges away). An expert can then interactively explore the network by clicking on neighboring nodes.

`Pubmed Network View` is a plug-in which provides a view on the large Pubmed article network. The user can first filter the articles and then select a set to display. In addition to selected articles, neighboring articles are also included if they satisfy selection criteria (edge distance from selected, maximum number of neighbors, and edge threshold). Right-clicking the article in the `Net Explorer` pops up a menu of options: to remove, expand, or score the corresponding article. In the example in Figure 3.1 the article "Detecting protein function and protein-protein interactions from genome sequences" was selected and displayed together with the most similar articles (at most 2 edges away and with edge weight higher than 0.5).

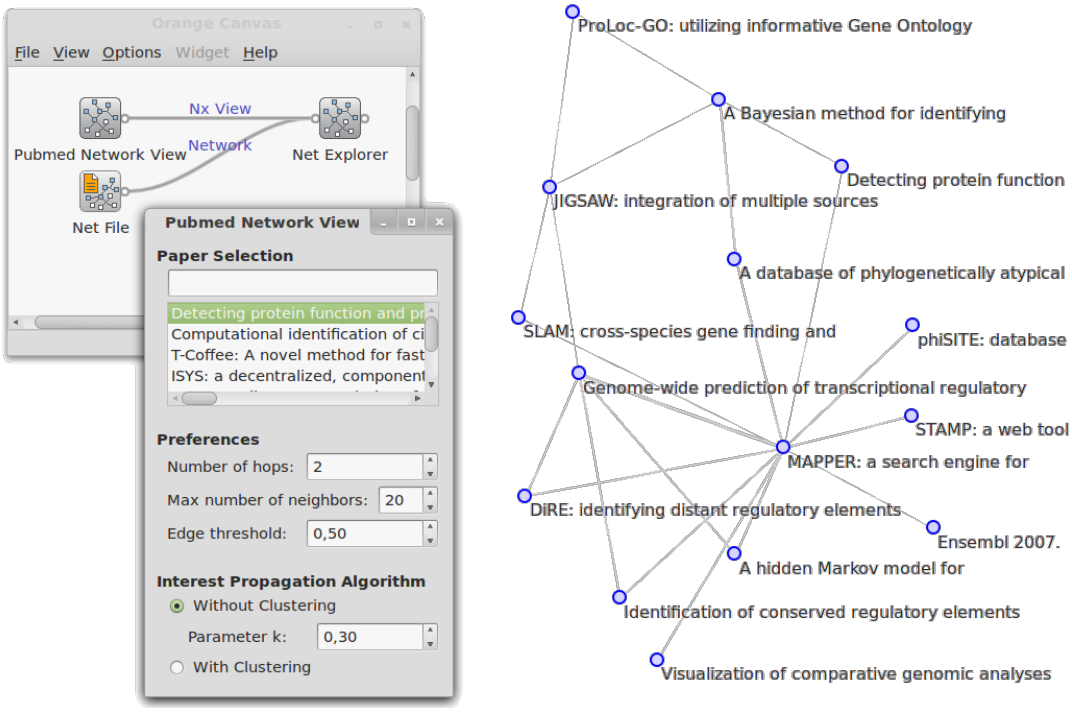


Figure 3.1: A subgraph of the Pubmed article network with the selected article "Detecting protein function and protein-protein interactions. . . ," visualized together with 14 similar articles.

3.2.3 Network Analysis

The `Net Analysis` widget computes graph- and node-level statistics for the given network. Graph-level indexes are computed and displayed in the widget (Figure 3.2(a)). These include the number of nodes and edges, average node degree, graph diameter, average shortest path length, density, degree assortativity coefficient, graph clique number, graph transitivity, average clustering coefficient, number of connected components, number of attracting components, and others.

For the node-level indexes (Figure 3.2(b)), the widget outputs another network in which the computed indexes are appended to each node in the same way as, for instance, the `Net File` widget appends the data read from the file. This data can then be explored in the `Net Explorer` widget or other data analysis widgets, for example to build a prediction model based on the network structure. For an example, see the case study in Section 3.5. The computed node level indexes are node degree, average neighbor degree, clustering coefficient, number of triangles in which the node participates, number of cliques, degree centrality, closeness centrality, betweenness centrality, information centrality, core number, eccentricity, and others.

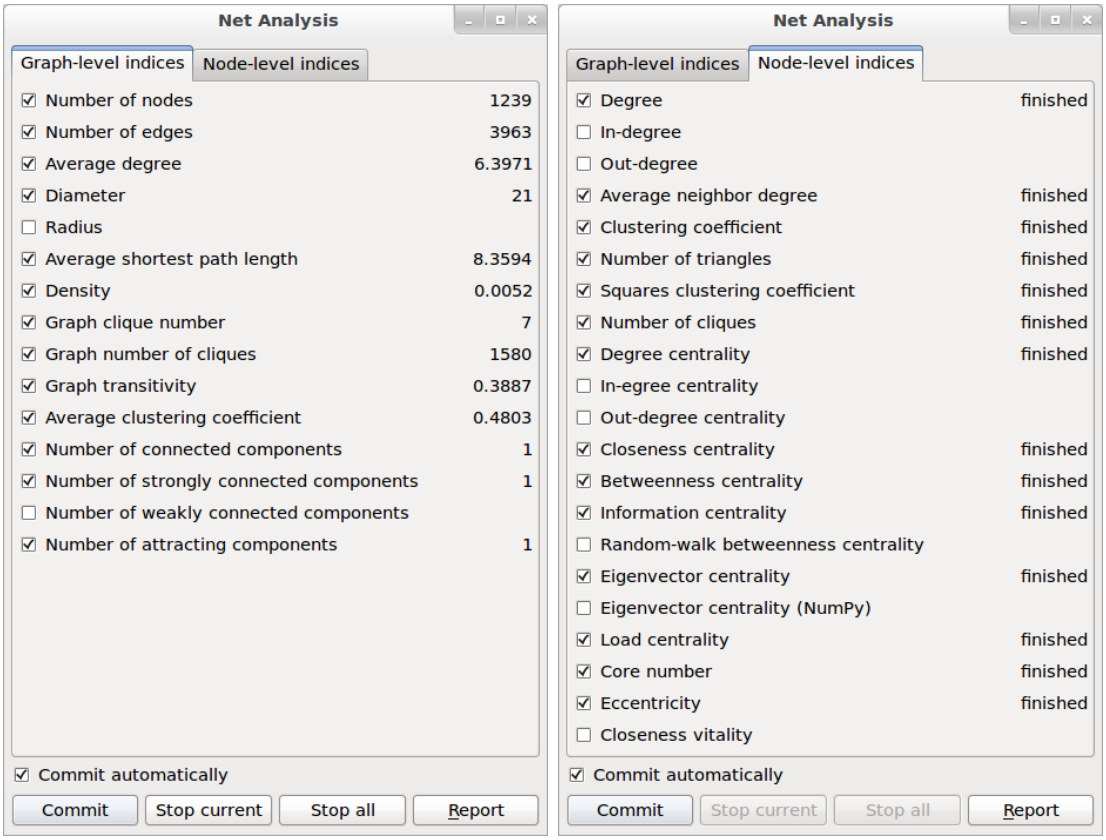
The computation of network analytic tasks is parallelized to save time. One processor core is always left free—in multi-core computers—for smooth user experience.

3.2.4 Community Detection in Graphs

Two label propagation clustering algorithms [99, 83] are implemented in the `Net Clustering` widget. As both algorithms are iterative, the user must set the number of iterations. The widget gets a network on the input and appends clustering results to the network data. Communities can then be explored in `Net Explorer`.

3.3 Using the `Net Explorer` Widget

We demonstrate the use of the widget on a network of musical groups and artists obtained from the Last.fm web radio, `http://last.fm`. The site provides a list of the five most similar artists or groups for each given artist, where the similarity is measured by the number of common listeners (the exact definition is not publicly available). The site does not publish the complete list of available artists, so we constructed the network using a search algorithm that started with a small set of



(a) graph-level indexes (b) node-level indexes

Figure 3.2: Graph- and node-level indexes computed on the Last.fm network, which is described in Section 3.3.

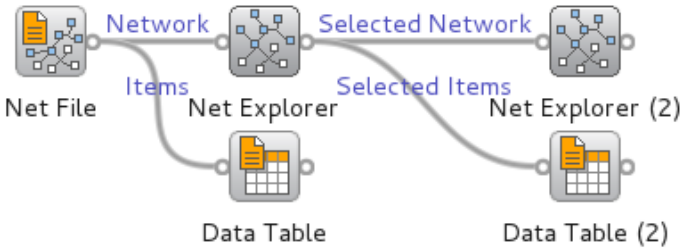


Figure 3.3: Orange Canvas scheme to view the graph and the corresponding meta data.

artists from diverse musical styles (U2, Johann Sebastian Bach, Norah Jones, Erik Satie, and Spice Girls), and then expanding it by querying for the artists who are similar to those in the set. We stopped the search after several days, when the set contained around 320.000 nodes. After removing the nodes for which we did not query for similar artists, we got a set with around 26.000 artists. We further reduced it by taking only the 2000 most connected artists.

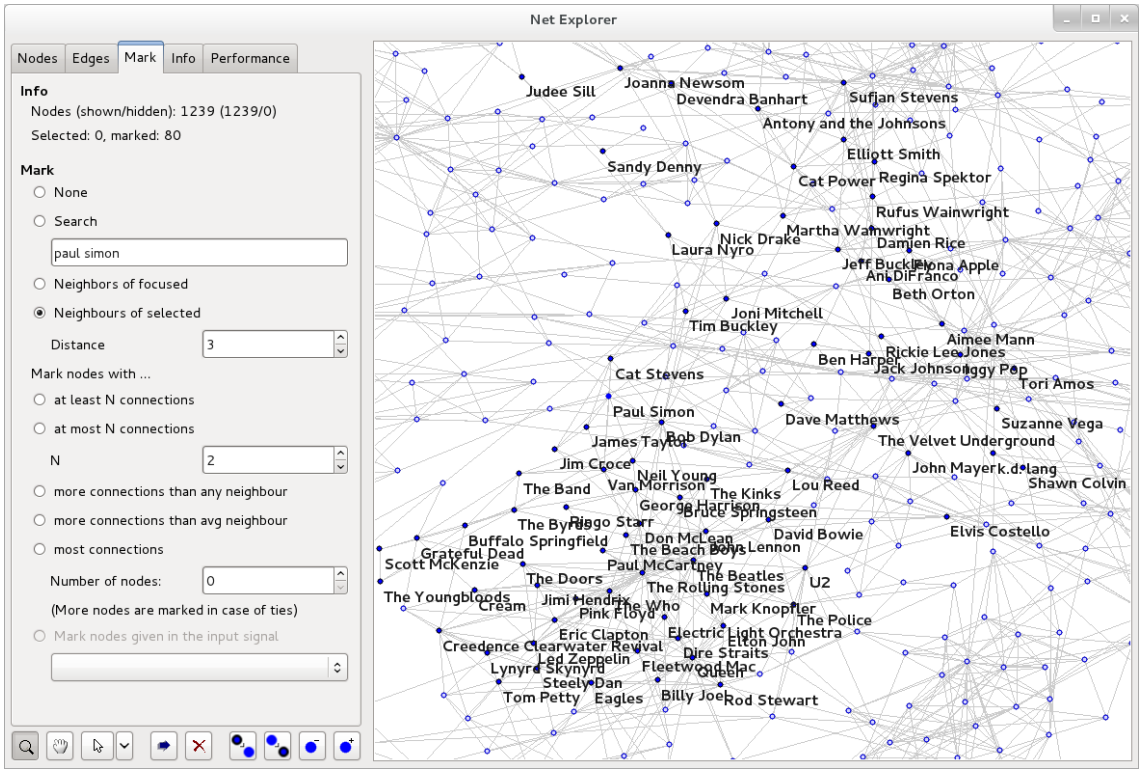
We extracted the largest connected subgraph (popular music), and retrieved additional data about the corresponding artists and groups: the number of albums released, the year of their first and last album release, the number of times they have been played on Last.fm and the number of distinct listeners, and lists of user-assigned tags (like "country," "60s," "female vocalists," "Russian rock," and similar). Last.fm weights the tags according to how well they correspond to a particular artist. For each artist, we identified the tag with the greatest weight and assumed that it corresponds to the genre to which the artist belongs. The data was retrieved in March 2007.

Since some of these data (album count, first and last album release year) were retrieved from another service, AOL music `http://music.aol.com`, we removed the artists for which the queries returned ambiguous results, mostly due to more than one group having the same name. The resulting graph contains 1262 nodes representing the most established popular music groups and artists.

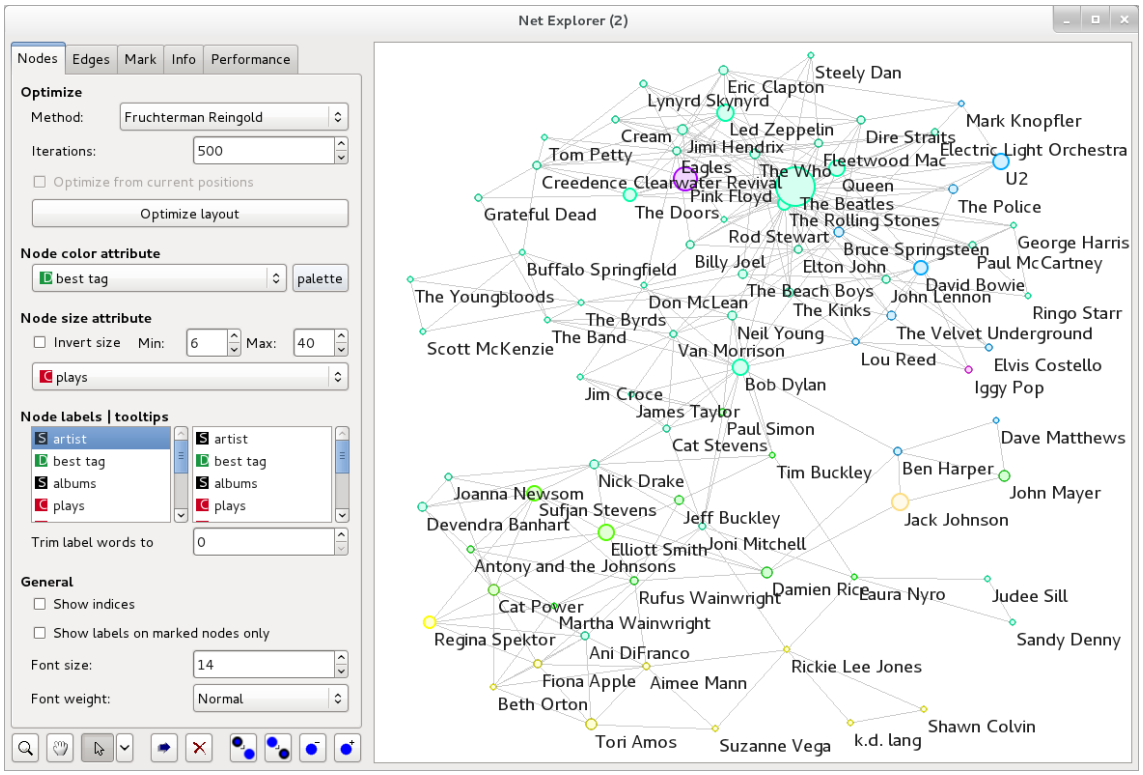
The purpose of this study is not to provide new insights into the Last.fm data, but to demonstrate the use of the `Net Explorer` widget. An interested reader will find the detailed analysis of this network in the works of [22] and [89].

3.3.1 Basic Graph Manipulation, Marking and Selecting

The scheme from Figure 3.3 loads the graph and additional data about the nodes (`Net File` widget). The data is shown in the `Data Table`, and the graph is plotted in the `Net Explorer` widget. To observe the subgraphs we are going to select in the `Net Explorer`, we attached another `Net Explorer` and `Data Table` to its output.



(a) Vicinity of Paul Simon in the entire graph.



(b) Subgraph with additional data: the number of plays as size and genre as color.

Figure 3.4: Exploration of artists similar to Paul Simon.

When we open the `Net Explorer` and select the variable "artist" for the node labels, the graph—despite being optimized by the Fruchterman-Reingold algorithm—looks like a huge cloud of unreadable overlapping labels. Let us say that we are interested in exploring the vicinity of Paul Simon. In the "Mark" tab we select "Find nodes" and type "Paul Simon." This finds and *marks* the node corresponding to Paul Simon (the node is drawn as a filled circle with a black border, as opposed to others that are hollow). We can then click the button for selecting the marked nodes. This *selects* the Paul Simon's node (the node is now filled but has no border). We can proceed by choosing "Mark neighbors of selected nodes," and set the distance to, say, two. This *marks* all nodes that are at most three connections from Paul Simon. To further reduce the visual clutter, we check "Show labels on marked nodes only," and zoom in the marked part of the graph. The result is shown in Figure 3.4(a).

We can again select the marked nodes, and output the corresponding subgraph. Since the second `Net Explorer` shows only the subgraph, the resulting layout is much nicer, as it is not affected by other artists in whom we are not interested at the moment. We added the information about genres by coloring the nodes according to the tag that best describes them, and resized them according to their popularity (the number of times their music has been played) on Last.fm.

The result in Figure 3.4(b) shows a graph with Paul Simon and a few other artists (in particular Bob Dylan) between two stronger components—classic rock with the Rolling Stones as the central point on one side, and various representatives of a more acoustic and vocal style on the other.

3.3.2 Grouping Nodes by Genres

We again drew the entire graph with the layout optimized by Fruchterman-Reingold algorithm and colored all nodes by the most important tag, which presumably gives the genre. The result in Figure 3.5 shows that there is a good correspondence between genres (the most important tags) and the groups which can be visually observed in the graph. We can easily label regions of the graph by the corresponding "genre" based on the prevalent tag. This confirms that the graph can be—with a grain of salt and caution—used for subjective visual clustering.

3.3.3 Development of Genres

To observe how the musical genres evolved over time we plotted the same graph as before, but used the `Select Data` widget (Figure 3.6) to select subsets of artists based on the year of their first album release, and used this data to select the nodes in the `Net Explorer` by feeding it to the "Items Subset" slot. The artists active before the selected year were represented with filled symbols.

Figure 3.7 shows graphs for artists active before years 1985, 1990, 1995, and 2000.

Before 1985, most active artists performed "classic rock" (this probably covers multiple genres which most Last.fm's listeners do not distinguish between). We can notice some soul, hard rock and pop groups, and of course, the 80's tag. In the next image (before 1990) even more groups in these genres are active. We also see the emergence of funk music. By 1995 most of the popular groups from these genres are present, and there are a lot of new genres, such as metal, hip hop, rap, R&B, indie, punk, and hardcore. The last image adds electronic and emo, and the only style still missing is minimal.

3.3.4 K-means Clustering in Graphs

Using the widget for k-means clustering (Figure 3.8(a)), we split the musicians into 22 clusters. The distance was defined as the Manhattan distance between the

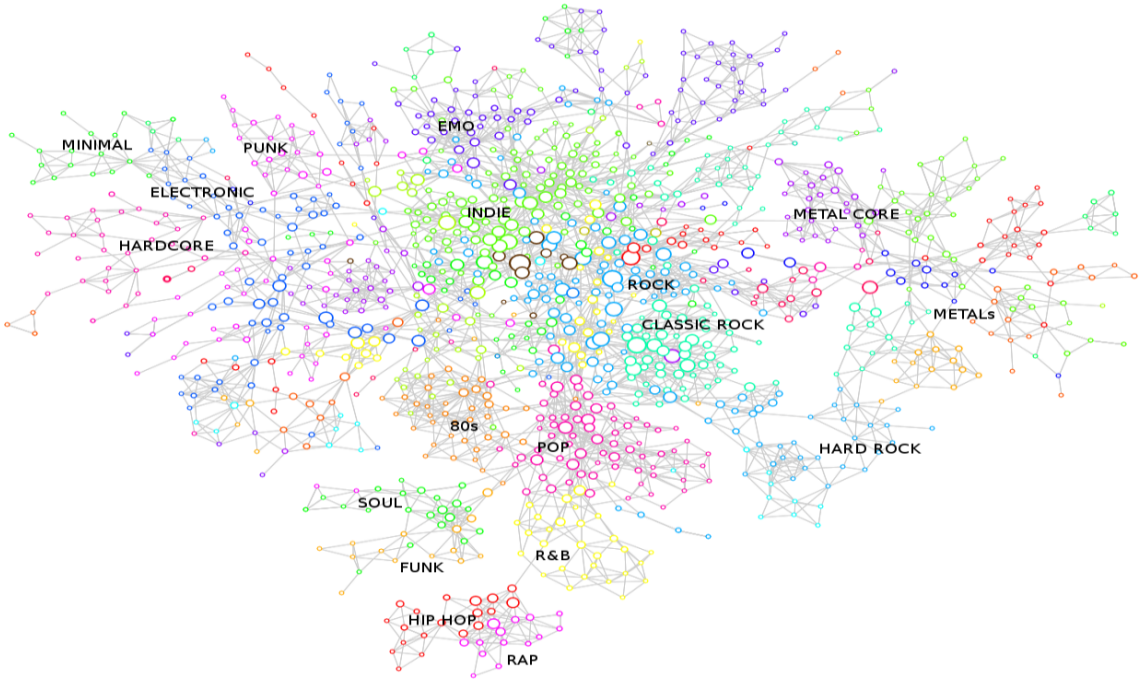


Figure 3.5: Genres are named by the prevalent tag in each region, as judged by a human expert.

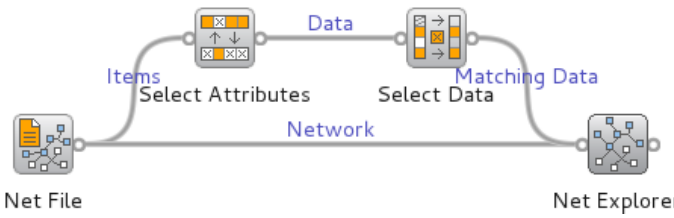
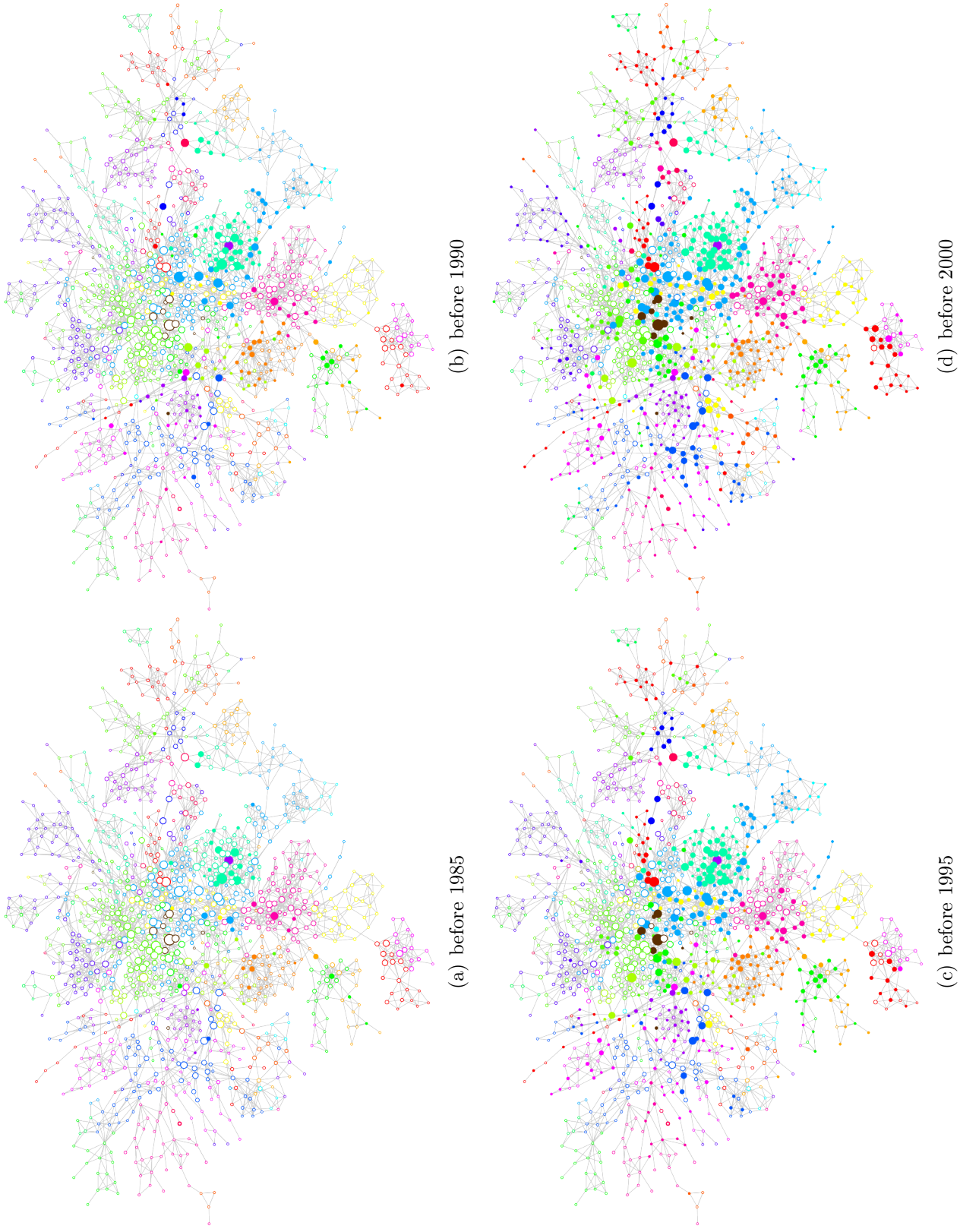


Figure 3.6: Orange Canvas scheme for analyzing the appearance of musical genres.



(a) before 1985 (b) before 1990 (c) before 1995 (d) before 2000

Figure 3.7: Genre appearance by periods.

weights of tags (tags not appearing at a certain artist were assigned a weight of 0). The number of clusters was determined by the BIC criterion [98]. We colored the nodes corresponding to their respective clusters (Figure 3.8(b)), and found a good correspondence between the apparent "graph clusters" and k-means clustering. This can be the result of using the tags (the basis for the clustering) in the definition of similarities (the basis for the graph) as reported by Last.fm. However, since the available documentation on Last.fm states that the definition is *mostly* based on the number of common listeners, the most plausible explanation is that a typical listener sticks to a certain kind of music denoted by the same set of tags. This can thus be an informal practical verification of the reliability of the tagging system.

3.3.5 Genres and Number of Albums

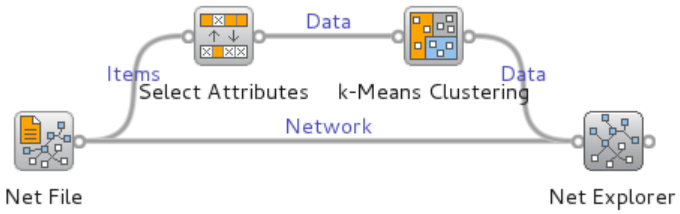
Sizing the nodes by the number of released albums (Figure 3.9) gives us a rough impression about the number of albums per genre and its variation. The picture suggests that artists from some genres generally release much more albums than their colleagues belonging to other genres. However, there is a correlation between the number of albums released and the year of publication of the first album (Spearman rank correlation is -0.70), so the different number of albums can be attributed to the different ages of genres.

3.3.6 Popularity and Influence

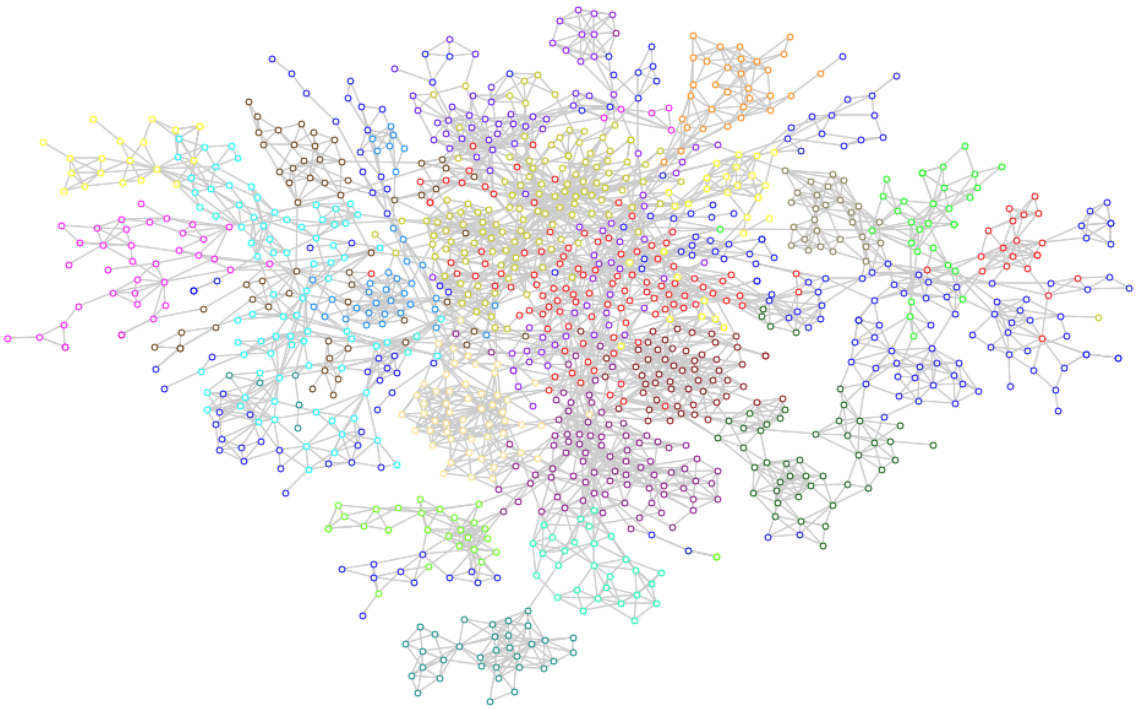
To demonstrate how the graph widget can be used to select data, we first plotted the artists in a scatter plot in which we separated the groups by their clusters by using the cluster ID (genre) for the x axis, while the y axis represents the number of times the artist was played on Last.fm (Figure 3.10). The colors represent different clusters and the size of points corresponds to the number of released albums. Then we used the `Net Explorer` to select the 50 most connected nodes; these nodes represent the most influential artists. We fed these hubs to the `Scatter Plot` widget, which marked them by filling the corresponding symbols. With only 50 out of 1262 artists selected, we see that a disproportionate number of them appears at the top of their corresponding clusters.

3.3.7 Manipulating Graph Data from Scripts

We will show how to use the Python module behind the `Net Explorer` widget directly by scripting in Python. We will verify the findings from the previous example, namely, that influential groups (those with more connections) are also among the most popular groups (that is, the most listened to) within their respective genres.



(a) `Net Explorer` connected to the k-means widget.



(b) Nodes in the same cluster share the same color.

Figure 3.8: Network clusters found by the k-means clustering of network meta-data.

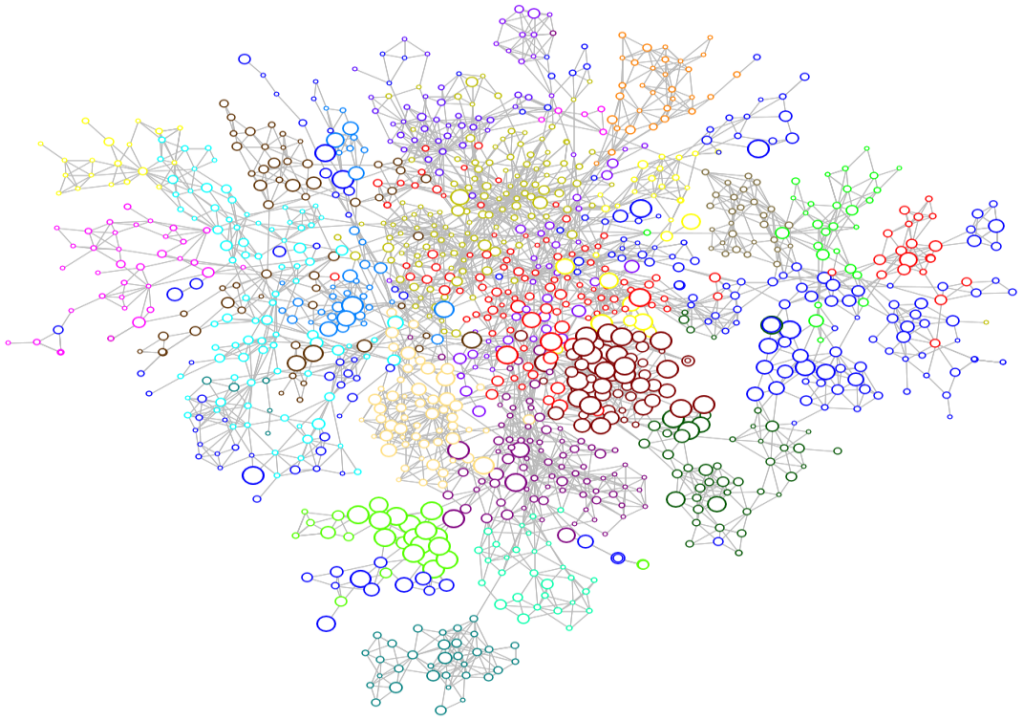
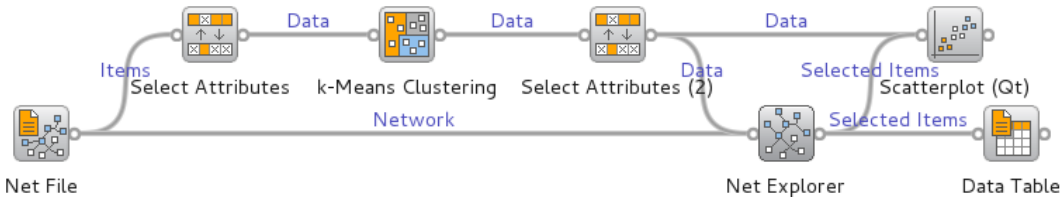


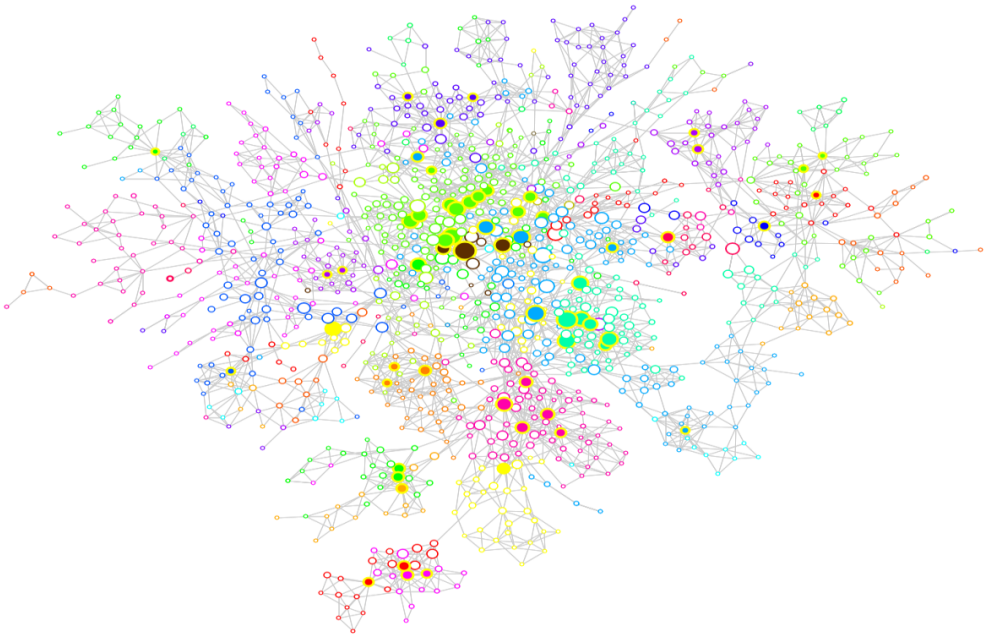
Figure 3.9: Graph with the number of albums represented by the node size.

To exclude the effect of the genre, we divided the number of plays for each artist by the maximum number of plays in the corresponding genre (genres were defined by the k-means clustering). We then used the Mann-Whitney test to compare the number of plays for the group of the 50 most connected artists with the other artists. The difference was highly significant ($p < 0.01$).

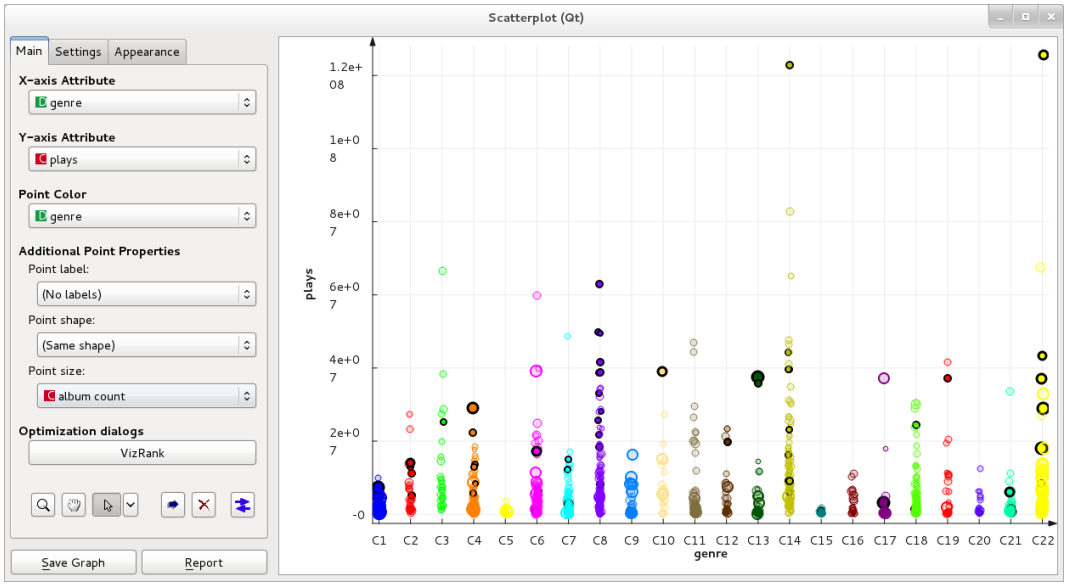
```
import numpy, scipy.stats, Orange
from operator import itemgetter
from Orange.clustering import kmeans

# read the data
net = Orange.network.readwrite.read("lastfm.net")
data = Orange.data.Table("lastfm_tags.tab")

# cluster the artists into genres
manhattan = Orange.distance.Manhattan
kmeans = kmeans.Clustering(data, centroids=22, distance=manhattan)

# normalize the number of plays by dividing by the maximum within the cluster
plays_clust =   [(row["plays"], clust) for (row, clust) in zip(data, kmeans.clusters)]
maxs =   [max(plys for (plys, clust) in plays_clust if clust == c) for c in range(22)]
normalized = [plys / maxs[clust] for (plys, clust) in plays_clust]
```



(a) Orange Canvas scheme for observing the popularity of network hubs.



(b) Network with marked hubs and node size set to popularity.



(c) A scatter plot with marked hubs.

Figure 3.10: Correspondence between the influence and popularity within different genres.

```
# split the data into number of plays of hubs and non-hubs
hubs, degrees = zip(*sorted(net.degree().items(), key=itemgetter(1))[-50:])
hubp = numpy.array([normalized[i] for i in hubs])
not_hubp = numpy.array(
 [plys for (i, plys) in enumerate(normalized) if i not in hubs])

# compute the Mann-Whitney test
u, prob = scipy.stats.mannwhitneyu(hubp, not_hubp)
print "Mann-Whitney U: %d, p=%e" % (u, prob)
```

Note that this is only a toy example. The experimental procedure is invalid since the same data is used to formulate and to validate the hypothesis. Besides, if the similarity between two artists depends upon the number of mutual fans not normalized by the number of fans of each individual artists, the artists which are more popular get more connections simply due to their popularity, and the discovered relation follows directly from the definition of similarity. We cannot verify this since Last.fm keeps the exact definition of similarity secret.

3.4 Extending Existing Data Analysis in Orange Canvas

A major advantage of the `Net Explorer` widget is that it can be used in existing Orange Canvas schemes, which often yields additional insight about the problem domain. Consider the example in Figure A.1, where the aim is to build a prediction model on the Iris data set. Although the classification accuracy of the k-nearest neighbors classifier is rather high (0.94), we wish to further explore the instances where the model fails.

We connect the `Select Data` with the `Example Distance` widget. Next, we add the `Net from Distances` and `Net Explorer` widgets as in Figure 3.11(a), and select all misclassified instances of the k-nearest neighbors model in the `Confusion Matrix` widget. In the `Net from Distances` widget we connect each node with the 3 most similar nodes, to simulate the behavior of the `K-nearest Neighbors` widget's model exactly.

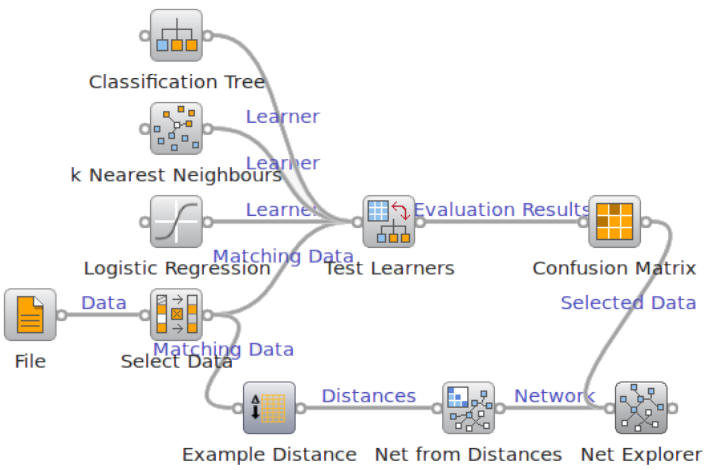
After observing the outliers in the `Net Explorer` (Figure 3.11(b)), and toying with different distance measures, we notice that it is not possible to further increase the classification accuracy without overfitting.

3.5 Network Mining

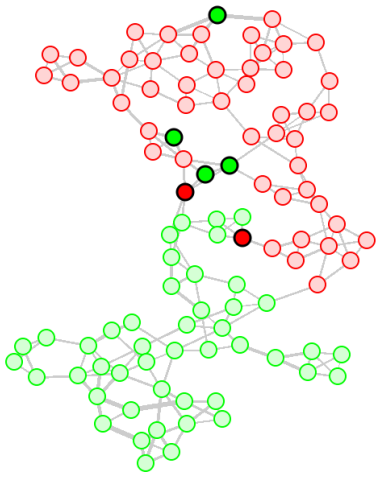
The last case study will demonstrate how to use network widgets in Orange to seamlessly integrate machine learning algorithms into the analysis of network data. In this example we explore the air traffic data. The network was constructed by parsing timetables of three major airlines: Lufthansa, United, and American Airlines. Graph nodes represent airports; a pair of nodes is connected if any of the listed airlines provides a direct flight between them. Airport data was extracted from the World Airport Traffic Report issued by the Airport Council International in 2006. The network data set includes: the number of aircraft movements (landing or take-off of an aircraft), the number of passengers arriving or departing via commercial aircraft, the cargo handled in tonnes, the airport category as specified by the Federal Aviation Administration (Large Hub, Medium Hub, Small Hub, Nonhub Primary, and Nonprimary Commercial Service), and a class variable "FAA Hub" specifying whether the airport is considered a hub by the FAA or not.

The Orange Canvas scheme of the analysis is shown in Figure 3.12. At first glance, there are two major communities of airports. We confirmed this with the `Net Clustering` widget: applying the community detection method by [99] and coloring the nodes by the clustering result indeed revealed two clusters. The results in Figure 3.13 show nodes colored by the two discovered communities. We then set the node tooltips to show the attribute "city," and hovered over some airports from each cluster. We discovered that the airports are clustered according to the continents, and the two large communities represent airports in North America and Europe.

Our next objective was to explore the correlation between the network topology and



(a) Orange Canvas scheme (extended from Figure A.1).



(b) The 3-nearest neighbors network; misclassified instances are marked.

Figure 3.11: Analysis of misclassified examples on Iris data using `Net Explorer`.

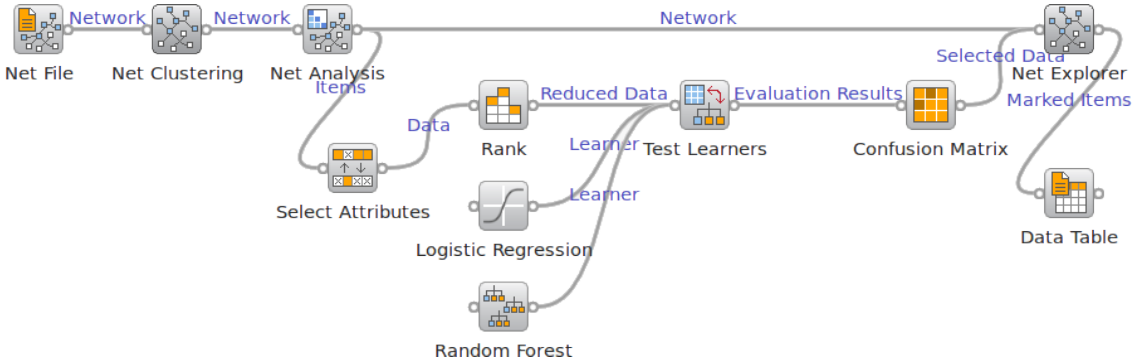


Figure 3.12: The complete Orange Canvas scheme used in the analysis of air traffic data.

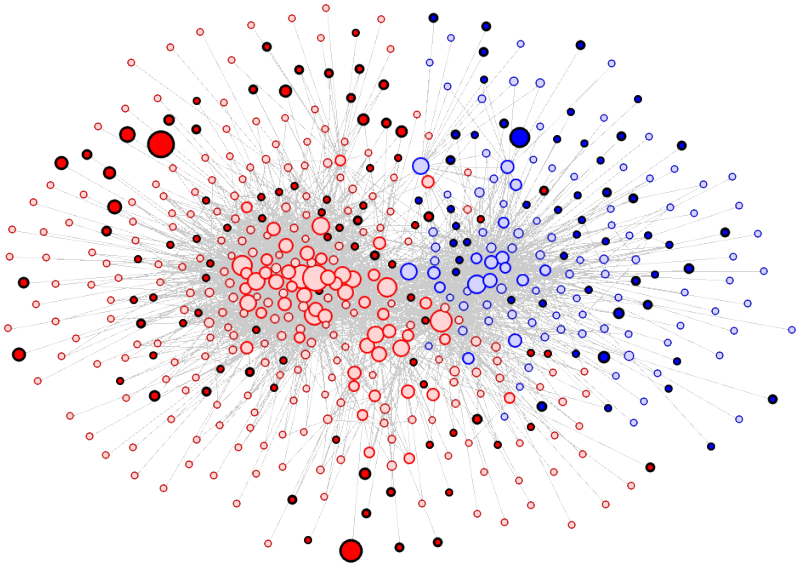


Figure 3.13: Air traffic network colored by communities. Nodes that represent misclassified instances are marked.

the airport type; more specifically, to predict whether the airport is a hub or not from the network topology. First, 14 node-level indexes were computed with the `Net Analysis` widget (Figure 3.2(b)). They were scored and compared in the `Rank` widget. The top half of the ranked attributes (according to the ReliefF measure of [75]) were used to test different learning algorithms (*logistic regression*, *naive Bayes*, *random forest*, and *support vector machines*). Examples that were misclassified by the logistic regression learner (the model with the best classification accuracy, 0.73) were selected in the `Confusion Matrix` widget, marked in the `Net Explorer`, and finally listed in the `Data Table` (see the marked examples in Figure 3.13). It seems that most of the classification errors were made on peripheral nodes which could be a result of the incompleteness of the analyzed air traffic network.

Chapter 4

Space of Classification Models

Complex data can be represented using a number of partial views, such as linear projections, each involving at most a couple of variables and showing a single, particular, and simplified perspective or relation. Similarly, good predictive models can be built using ensembles of simple models, such as random forests of small trees, each covering a part of the problem domain.

In this chapter, we propose a method for creating maps of classification models which are graphically presented to the user and allow for interactive exploration of the models. The method constructs the map of the model space that reflects the similarity of models. The crucial idea is that two models are similar—from the expert perspective—not if they *look* similar (*e.g.* both are trees or use the same attributes), but rather if they give similar predictions. This enables us to present a combination of many kinds of different models. The algorithmic challenge is to allow a real-time exploration of a large set of models.

4.1 Data Mining and Meta Learning

Data mining [117, 76] is a field of computer science which deals with extracting patterns from large data sets by combining methods of artificial intelligence, statistics, and visualization. We would like to automatically extract patterns of learning data by processing data instances and background knowledge. The accumulated knowledge is often presented in the form of a formal model that can be used to make predictions about new (unseen) data instances. Most data mining methods originate from machine learning [86]. There are four typical tasks of data mining:

- **Regression.** The task of regression is to find a function of the independent variables called the regression function. Regression analysis is used for prediction of the dependent variable.

- **Clustering.** The task of clustering is to find groups and the structure of the data.
- **Learning association rules.** The task of learning association rules is to find logic relations among variables.

The data mining and knowledge discovery suites contain many different algorithms, yet they lack the guidelines for selecting the best method given the problem at hand. The goal of meta learning [16] is to understand principles of learning algorithms, how they perform on different learning tasks with the intention of improving the learning process. Meta learning is a branch of machine learning that applies learning algorithms to the *meta data*, describing the machine learning experiments—the learned models. Examples of meta data include learning problem features (general, statistical, information-theoretic, and others) and learning algorithm features (type, parameters, performance measures, and others).

Meta-learning tries to understand the link between the learning problem (the data) and the efficiency of various learning algorithms. The learning algorithm might perform remarkably well on one problem, while it does exceptionally bad on the other. Learning algorithms are different in their presumptions regarding the data, which is called a learning bias. The bias characterizes any proclivity of the learning algorithms toward choosing one of the hypotheses over the other (for other definitions of the learning bias see [50, 85]). Each learning algorithm is therefore suitable only for a problem domain that best fits the presumptions. Many real-world application of meta learning exist, but the exact definition is still not clearly defined; the following are different approaches to meta-learning:

- **Algorithm recommendation.** The task is essentially a search problem. The search space contains machine learning algorithms and the goal is to find the best set of algorithms for the given data. We would like to discover rules that govern the connection between learning algorithms and data domains. Usually, we first derive the domain features that are related to the learning algorithm. The rules for efficient algorithm recommendation can then be learned from that feature set [2].
- **Combine base-level ML systems.** The task is to build a combined learning system with superior hypothesis abstraction. Examples of those are boosting, stacked generalization [118], cascading, arbitrating, and meta decision trees.
- **Dynamic bias selection.** Dynamic bias selection is a research on how to choose suitable presumptions for learning. It should be used when we can choose among different presumptions for the given learning problem [100].
- **Inductive transfer.** Inductive transfer is a research problem in machine learning that focuses on applying previously gained knowledge to similar problem domains. Instead of starting to learn from scratch, the existing knowledge is transferred to new domains [21, 110].

- **Learning classifier system (LCS).** A learning classifier system [80] is a machine learning system with close links to reinforcement learning and genetic algorithms. It was first described by John Holland [62]. The basic principle of his LCS was a population of binary rules on which a genetic algorithm altered and selected the best rules.

The term "meta learning" has been defined differently by various research groups [112] and is used for multiple subjects. We consider exploring relations among various learning algorithms a meta learning technique.

4.2 Model Distance Measure

Models can be considered similar in many aspects, for example: they can either be of the same type, trained on the same set of attributes, or have the same classification bias. We define novel measures of distance between classification models under assumption that two models are similar when they make similar predictions. Concretely, two models m_i and m_j are similar when they predict the same class (or class probabilities) for a given instance. We compare novel model distance measures with existing measures of the distribution (dis)similarity.

4.2.1 Class-based Distance Measures

The first similarity measure considered is the relative frequency of the same-class predictions by models m_i and m_j. The distance is defined as an inverted similarity,

$$\delta(m_i, m_j) = 1 - \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} |\{c_{m_i}(e)\} \cap \{c_{m_j}(e)\}|, \tag{4.1}$$

where $\mathcal{E}$ is a set of instances and $c_{m_i}(e)$ is the class predicted for an instance e by the model m_i.

Another measure we experimented with is the inverted normalized mutual information—symmetric uncertainty [117], given by:

$$\delta(m_i, m_j) = 1 - 2\frac{I(c_{m_i}; c_{m_j})}{H(c_{m_i}) + H(c_{m_j})}, \tag{4.2}$$

where $c_{m_i} : \mathcal{E} \to \mathcal{C}$ is a discrete random variable corresponding to the class predictions by the model m_i, $\mathcal{C}$ is the set of classes, $I(c_{m_i}; c_{m_j})$ is the mutual information between c_{m_i} and c_{m_j}, and $H(c_{m_i})$ and $H(c_{m_j})$ are the marginal entropy.

4.2.2 Probability-based Distance Measures

Class-based distance measures are coarse, what limits the construction of network of models. For example, when we connect nodes more similar than some threshold, we may get a network of isolated nodes for a particular threshold, and a densely connected one for the next possible threshold. Another reason to extend the basic class-based distance measure is that we strive for a measure that would resemble the proposed objective: *models are similar when they make similar predictions*. The probability of predicted classes describe models in greater detail.

Assuming that the models estimate class probabilities over the same set of classes, we define the model distance by the average Euclidean norm of the difference between predicted class probabilities:

$$\delta(m_i, m_j) = \frac{1}{\sqrt{2}|\mathcal{E}|} \sum_{e \in \mathcal{E}} \|p_{m_i}(e) - p_{m_j}(e)\|, \tag{4.3}$$

where $p_{m_i}(e)$ and $p_{m_j}(e)$ are the vectors of predicted probabilities for all classes for example e by models m_i and m_j. The measure is normalized to the range $[0, 1]$. We also propose a variant with the Manhattan norm for performance reasons:

$$\delta(m_i, m_j) = \frac{1}{\sqrt{2}|\mathcal{E}|} \sum_{e \in \mathcal{E}} \|p_{m_i}(e) - p_{m_j}(e)\|_1, \tag{4.4}$$

as we expected a significant performance boost by avoiding the computationally intensive operations of square root and squaring. The difference is notable (the Manhattan norm runs approximately 1.5 times faster).

Lastly, we compare our distance measures with a measure of statistical dependence between models. Since we are focused on the space of *classification* models, we are particularly interested in the rank of probabilities. The distance is defined as the inverted Spearman's rank correlation coefficient [106],

$$\delta(m_i, m_j) = 1 - \frac{\sum_{e \in \mathcal{E}; c \in \mathcal{C}} (r^c_{m_i}(e) - \overline{r_{m_i}})(r^c_{m_j}(e) - \overline{r_{m_j}})}{\sqrt{\sum_{e \in \mathcal{E}; c \in \mathcal{C}} (r^c_{m_i}(e) - \overline{r_{m_i}})^2 (r^c_{m_j}(e) - \overline{r_{m_j}})^2}}, \tag{4.5}$$

where $r^c_{m_i}(e)$ and $r^c_{m_j}(e)$ are the ranks of the predicted probabilities of class c for example e by models m_i and m_j, while $\overline{r_{m_i}}$ and $\overline{r_{m_j}}$ are the average ranks.

4.2.3 Comparison of Distance Measures

We compared distance measures on models learned on five UCI data sets: Breast Cancer Wisconsin, Dermatology, Iris, Voting, and the Zoo data set, which have a different number of class values and attributes (the details are in Table 4.1). For each data set, a series of data projections and corresponding k-NN classification models

in the projected space were generated. Their number depends on the number of attributes and is summarized in the Table 4.2. Each model is learned and evaluated 10 times within the 10-fold cross-validation technique to obtain representative prediction probabilities for each data instance. Model distance matrices are computed for distance measures defined in Section 4.2.1 and 4.2.2 on all data instances.

We evaluated the distance measure similarity with the Spearman rank test. The results are in Table 4.3. The correlation of all pairs of distance matrices is high ($\rho > 0.7$), with all the corresponding p-values less than the significance level $\alpha = 0.01$ (Appendix B.1). The test indicates small differences between the distance metrics.

The computation of Manhattan norm distance measure defined in Equation 4.4 is 1.77 times faster on average then the Euclidean norm distance measure defined in Equation 4.3. Computation times of different model distance matrices, measured on the UCI data sets, are in Table 4.4.

Our goal is to draw a map in the Euclidean plane, so we prefer the Euclidean-like squared differences over other measures of distribution (dis)similarity, such as the Kullback-Leibler divergence. If the predicted class probability distributions are interpreted as $|\mathcal{C}|$-dimensional vectors, the difference in Equation 4.3 represents the average Euclidean distance. Since there was an almost perfect agreement in ranking between the probability-based measures: the average Euclidean and Manhattan norm, we can choose either of them. In all subsequent case studies we will use the average Manhattan norm distance measure defined in Equation 4.4.

Table 4.1: Properties of the five UCI data sets.

	Examples	Discrete	Continuous	Class values
Breast Cancer Wisconsin	683	9	0	2
Dermatology	366	0	34	6
Iris	150	0	4	3
Voting	435	16	0	2
Zoo	101	16	0	7

Table 4.2: The number of different projections by projection type generated for comparison of distance measures.

	SPCA	Radviz	PolyViz	scatter plot
Breast Cancer Wisconsin	501	501	501	36
Dermatology	1,000	1,000	1,000	561
Iris	10	10	10	6
Voting	1,000	1,000	1,000	120
Zoo	1,000	1,000	1,000	120

Table 4.3: The Spearman rank correlation coefficients measuring correlation between different model distance matrices of models of the five data sets and the corresponding p-values. Distance measures are labeled as D[*equation number*].

		Spearman ρ			
		D4.2	D4.3	D4.4	D4.5
Breast Cancer Wisconsin	D4.1	0.941	0.918	0.918	0.790
	D4.2		0.909	0.909	0.875
	D4.3			1.000	0.851
	D4.4				0.851
Dermatology	D4.1	0.861	0.768	0.825	0.806
	D4.2		0.755	0.812	0.838
	D4.3			0.966	0.830
	D4.4				0.883
Iris	D4.1	0.848	0.852	0.852	0.749
	D4.2		0.807	0.810	0.767
	D4.3			1.000	0.917
	D4.4				0.918
Voting	D4.1	0.815	0.838	0.838	0.820
	D4.2		0.860	0.860	0.828
	D4.3			1.000	0.856
	D4.4				0.856
Zoo	D4.1	0.867	0.893	0.907	0.826
	D4.2		0.874	0.887	0.841
	D4.3			0.989	0.849
	D4.4				0.872

Table 4.4: Computation time of different model distance matrices on four UCI data sets. Computation time is not reported for the Iris data set, since the number of possible projections is too small for significant differences. Time is measured in minutes on the Intel Core2 2.4 GHz CPU.

	D4.1	D4.2	D4.3	D4.4	D4.5
Breast Cancer Wisconsin	6.84	10.49	1.31	0.54	169.05
Dermatology	26.33	150.69	4.07	3.26	554.58
Voting	15.53	30.25	2.48	1.51	122.26
Zoo	17.12	129.33	2.75	1.86	323.85

4.3 Model Maps

We define a method that visually organizes models in the set to allow manual analysis. Concretely, we construct and plot a graph of models in which similar models are put close to each other. The graph represents a visualization of the "model space," therefore a *model map*.

The proposed technique starts with set of classification models and predicted class probabilities for each example in the data. As is usual in machine learning, examples used for prediction need to be held out of the training data. We used a 10-fold cross-validation in the case study. Essentially, each model was fit to the 90% of the data, and evaluated ten times on the remaining 10%. Class probability predictions from all 10-folds were then merged to form the input to the model map algorithm.

Let $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$ be a set of classification models and let $\delta(m_i, m_j)$ be a measure of difference between models m_i and m_j (we assume that the set includes models, which may be homo- or heterogeneous). The proposed algorithm first reduces the set $\mathcal{M}$ into a set of representative models $\mathcal{M}'$. To keep the diversity of models, $\mathcal{M}$ is split into subsets $\mathcal{M}_1$, $\mathcal{M}_2$, ..., $\mathcal{M}_l$ each containing a single kind of model (*e.g.* $\mathcal{M}_1$ contains classification trees, $\mathcal{M}_2$ contains SVMs, ...). For homogeneous $\mathcal{M}$, $l = 1$ and no splitting is needed.

For each subset $\mathcal{M}_i$, we use an arbitrary clustering technique to find the representative models (we will show that several clustering techniques yield similar results). A single representative is determined for each cluster, depending on our objective, for instance: a model with the highest performance score or the model with the smallest average distance to other models in that cluster. We collect the representatives of all clusters of $\mathcal{M}_i$ into set $\mathcal{M}'_i$. The final set of representatives is the union of representatives from all subsets, $\mathcal{M}' = \bigcup_i \mathcal{M}'_i$. Finally, we connect each model in $\mathcal{M}'$ with k most similar models to form a graph. Various layout algorithms can be used to visualize the graph. Most graphs we present consist of several small components, which are best visualized with the FragViz network visualization technique.

Note that the procedure is constructed so that it does not need to store the potentially immense matrix of distances between the models, since $\delta(m_i, m_j)$ can be computed on the fly without significantly increasing the time complexity.

4.4 Exploration of Model Space

We developed an interactive tool for exploring the maps: Model Maps add-on for Orange [32]. We designed a map of diverse models to highlight the add-on's aggregative aspect. Models were inferred on the Zoo data set, a simple database containing 17 discrete attributes, describing animal characteristics; animals belong to 7 different species (classes).

The collection of models consists of two classification algorithms:

- k-nearest neighbors ($k = \sqrt{|\mathcal{E}|}$, as default in Orange),
- naive Bayes,

both on different subsets of attributes; and four types of projections:

- supervised PCA [6],
- Radviz,
- PolyViz [61], and
- scatter plot,

which are translated to the k-NN models, induced on the coordinates of the projected instances. We prepared 500 models on randomly chosen subsets of attributes for each learning algorithm except for the scatter plot, where a maximum number of 120 different projections was used. Together, they form a *model map* of 3120 models.

We constructed a graph in which each model is connected to the k most similar models (a typical value for k is 1 or 2). We find clusters in each graph using a label propagation technique for community detection in graphs [99]. The number of clusters is decided by the algorithm itself; the typical number of clusters in our experiments was around 60 per subset $\mathcal{M}_i$.

The main part of the `Model Map` widget shows a graph such as the one in Figure 4.1. Each node corresponds to a model and the icon shows the type of the model. The same icons are used in Orange environment itself, so the user is presumably familiar with them. The size of the icon is proportional to the model's predictive accuracy. In Figure 4.1, scatter plots (the models at the southeastern rim of the map) have lower accuracies, which is consistent with the properties of the domain. The linear projections on the northeastern island have similarly low accuracies. The width of the edges corresponds to similarities between the models. The scatter plot's rim is barely connected, while there are two islands with three very similar naive Bayesian classifiers in the western and southern parts of the map. In the southeast there is also an island of four models of different types of a k-NN classifier, a naive Bayesian classifier and two classification trees.

To observe the properties of a certain region, the widget dynamically shows the chart with the classification accuracy of the models in the vicinity of the mouse cursor. The widget also offers general operations on graphs, such as selecting subgraphs, either manually or by pointing to a node and selecting vertices up to a given number of connections away, zooming in, cutting out parts of the graph, and so forth.

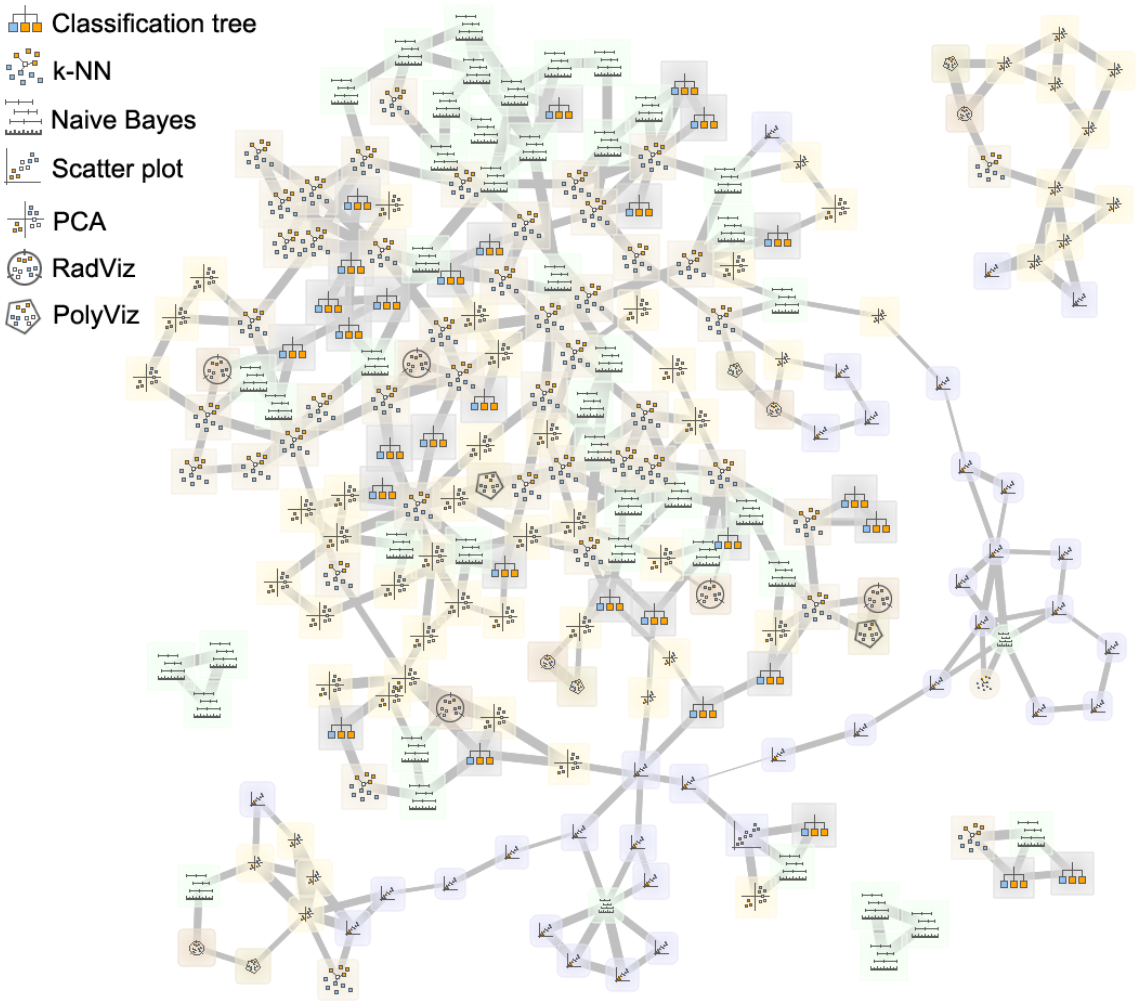
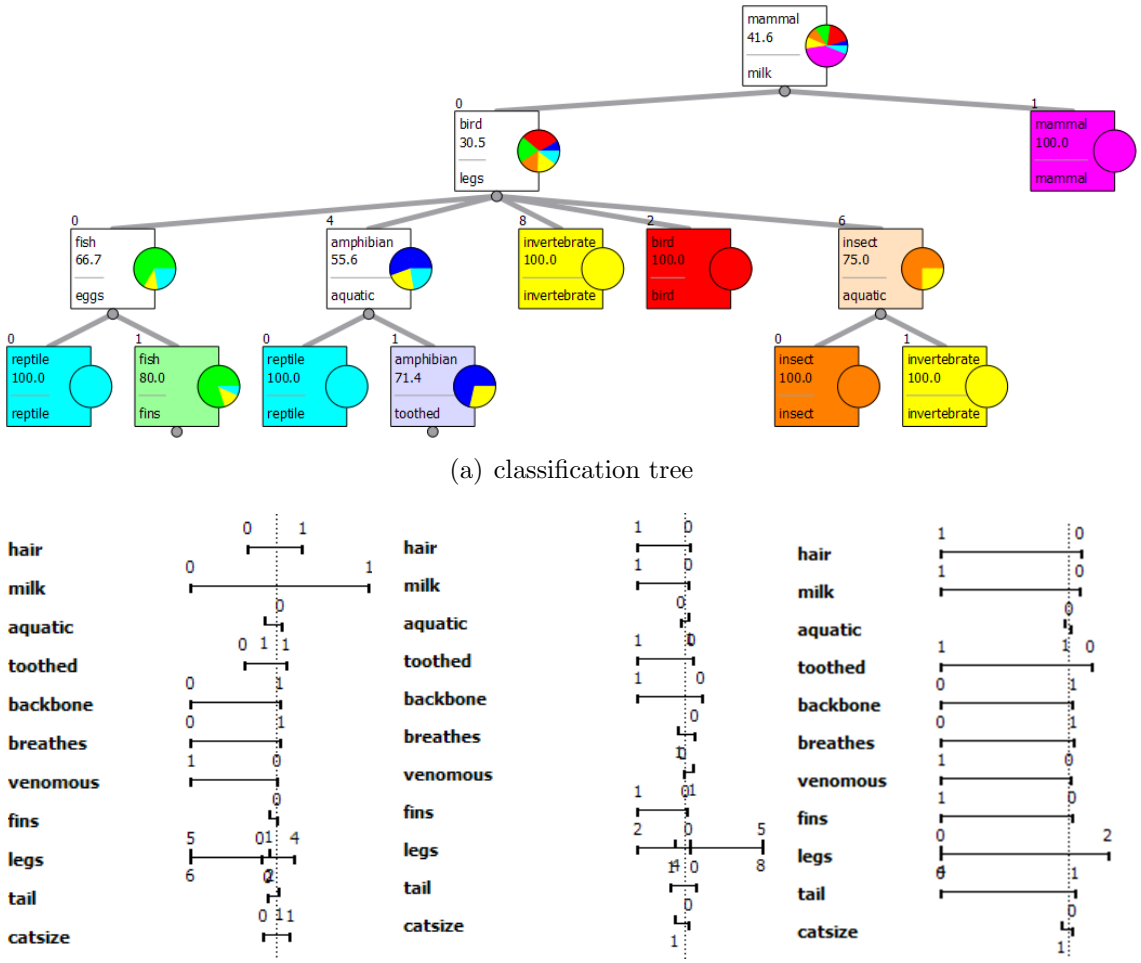


Figure 4.1: The *model map* of the Zoo data set.

Clicking the node brings up a visualization of the corresponding model, as shown in Figure 4.2 which shows an example of two similar models of different kinds. The classification tree (Figure 4.2(a)) uses the same attributes as the neighboring naive Bayesian classifier, visualized by nomograms showing odds ratios for individual classes [87]. Mammals are classified according to whether they give milk, by both the tree (the root node) and the Bayesian classifier (Figure 4.2(b), see the axis labeled *milk*). For birds and invertebrates, the most significant attribute is the number of legs; in the classification tree, the corresponding leaves immediately follow the node checking the number of legs, and in nomograms (Figures 4.2(c) and 4.2(d)) this is the attribute with the highest positive log odds.



(a) classification tree

(b) class: *mammal*

(c) class: *invertebrate*

(d) class: *bird*

Figure 4.2: A classification tree model learned on the Zoo data set and nomograms, visualizing a similar naive Bayes model. Only the top three levels of the tree and the central part of the nomograms are shown to conserve space.

Chapter 5

Space of Data Visualizations

Finding a good visualization is an important issue, considering the rapid growth of data collections and the increasing number of visualization methods. The task is equally valuable even for small data sets. Let us assume that we wish to find the best Radviz projection of the Zoo data set; we would have to review more than 600 billion projections. Several attempts were made to automatically search for interesting projections. VizRank [82] is one of the most successful and general methods. While the ranking of projections by the VizRank method assists in finding the best projections, the research on projection diversity is still an open issue. A list of the ten best projections is of no use to the analyst, if the projections are almost identical.

We propose a technique for organizing a large number of projections. We have used the idea of Leban *et al.* [81] to transform visualizations (such as linear projection, Radviz, or PolyViz) to a k-nearest neighbors classifier (k-NN), evaluated on the data instances in the projected space. The models are arranged according to the similarity of the views of the data they offer. Our method considers projection diversity while ranking the projections according to the evaluation of their usefulness. We will show that our method yields more information than VizRank when viewing the same number of best projections. In principle, *model-map-based* projection ranking could be used with any kind of projection scoring technique.

5.1 Related Work

Data visualization is a powerful technique in data analysis, as it matches our natural abilities to interpret data holistically. Attributes of the data (such as patterns, trends, structure, and outliers) which can be hidden in models are exposed in visual presentations.

In his book, William S. Cleveland states the importance of visualization methodologies: "Visualization is critical to data analysis. It provides a front line of attack, revealing intricate structure in data that cannot be absorbed in any other way." [24]. However, not all visualizations yield equal degree of insight and finding important data visualizations is a time consuming task. The abundance of visualization methods, which include scatter plots, parallel coordinates, mosaic displays, sieve diagrams, Radviz, and PolyViz, to name but a few, and selecting the right subset of attributes makes finding the best visualization by hand exceptionally hard, if not impossible. The number of possible visualizations typically increases exponentially with the number of concurrently visualized attributes.

Not many methods exist which assist in finding a good projection of data to two dimensions. Principal component analysis (PCA) [96] and Factor analysis (FA) [105] compute a projection from the original space to the lower-dimensional one with the aim to preserve the variance. The two are related, yet not identical. PCA is a descriptive statistical technique, while latent variable models, including FA, use regression modeling techniques to test hypotheses producing error terms. Since we are interested in models derived from the classified data, statistical dimension reduction methods like PCA and FA are not particularly suitable as they analyze the attribute space, ignoring the class.

Linear discriminant analysis (LDA) finds a linear combination of attributes that separates two classes. The resulting combination may be used as a linear classifier, or for a dimensionality reduction before later classification. LDA attempts to maximize the distance between the means of two classes while minimizing the variance within each class. For more than two classes, we can extend the Fisher linear discriminant [38] to find a subspace. The generalization is called multiclass linear discriminant analysis. In 1998, Dhillon introduced class-preserving projections, based on the LDA [33], by using discriminant functions as axes in a scatter plot. Its drawback is the small number of possible scatter plots (LDA generates the number of classes minus one attributes) and the inability to interpret the projection. Each axis is a linear combination of original attributes which is hard to interpret.

Topology Preserving Maps [74] were originally created as a visualization tool, enabling the representation of high-dimensional data sets onto two-dimensional maps and facilitating the human expert the interpretation of data. Similarly to LDA, this family of algorithms can be modified to serve as data classifiers, exploiting its inner pattern recognition capabilities.

Projection Pursuit [42] is another technique which involves finding the most "interesting" projections in multidimensional data. It searches for projections which deviate the most from the normal distribution. As each projection is found, the data is reduced by removing the component along that projection and the process is repeated to find new projections. Projection Pursuit suffers from similar problems as LDA. An axis in a projection is again a linear combination of attributes; so the projection is difficult to interpret. Its advantage over the LDA method is a higher number of possible projections and the fact that they are numerically evaluated.

Likewise, projections are scored in the VizRank method of Leban *et al.* [81, 82]. The criterion of "interestingness" is different, though. VizRank can score any projection of classified data instances to a two-dimensional plain by learning a k-nearest neighbors model on the projected points and then evaluating the performance of the induced classifier. The underlying rationale is that an analyst prefers projections with pure, well separated classes, and that the measure of classification performance will favor such projections to those where the classes would overlap.

5.2 VizRank

VizRank evaluates possible projections, given a data set and a visualization method. The result is an ordered list of projections, according to the assessment of their "usefulness". Projections are evaluated and added to the list in two steps:

1. Generate the data representation (projection) using a set of attributes.

2. Asses the "visual usefulness" of the representation.

While the first step is straightforward (we will test the same visualization methods as in the original article: scatter plot and Radviz), the second step will be explained in greater detail. VizRank tries to measure how likely it is that the user will spot a visual pattern in the data. Authors claim that the accuracy of the induced classifier on the projection space (new attributes are x and y positions of data instances) is indicative of projection quality as it gives higher scores to projections with well separated classes. They propose a k-nearest neighbors method (k-NN), since its bias towards forming the decision boundary produces "visually obvious" classification rules. We will use the Euclidean distance metric and set the parameter k to the square root of the number of instances, as proposed by Dasarathy [28].

Of the four scouring functions proposed for VizRank, we achieved best results with the average probability $\overline{P}$ that the classifier assigns to the correct class:

$$\overline{P} = E(P_f(y|x)) = \frac{1}{N}\sum_{i=1}^{N} P_f(y_i|x_i), \tag{5.1}$$

where N is the number of instances and $P_f(y_i|x_i)$ is the probability assigned to the correct class value y_i for instance x_i by the classifier f, a k-NN classifier in our case. This measure is used in all the reported results. Projections were evaluated with a 10-fold cross-validation technique.

5.3 Ranking Projections with Model Map

We extend the idea of *model map*, described in Section 4.3. Since $\mathcal{M}$ contains a single kind of model (in the case study: a scatter plot, Radviz on 3 or 4 attributes), there was no need to split $\mathcal{M}$ into subsets $\mathcal{M}_1$, $\mathcal{M}_2$, ..., $\mathcal{M}_l$. Clustering is done on the set of models $\mathcal{M}$ directly (in contrast to community detection in the graph in the original proposal of the *model map*) with probability-based distance measure, defined in Equation 4.4. The result is an ordered list of projections, corresponding to the cluster representatives $\mathcal{M}'_i$. We considered several clustering techniques:

- K-means clustering where k equals the number of constructed projections ($k = 15$ in the case study below). The shortcoming of this approach is the necessity to specify the number of projections in advance, which is an issue in exploratory data analysis.
- We address the above problem with an iterative k-means clustering. The algorithm starts with $k = 2$. In each iteration, k-means is run, representatives are selected from each cluster, sorted by $\overline{P}$ and added to the output list of projections (a projection is skipped if it is already present in the output list). Iteration is repeated with increased k until $k > |\mathcal{M}|$ or an output limit is reached.
- Hierarchical clustering with different linkage functions: average, complete, and single linkage. Projections are ranked similarly to the iterative k-means technique. Instead of rerunning the clustering algorithm, k topmost clusters were chosen iteratively.

The case study is performed on 11 data sets. In addition to the four in Table 4.1 (Iris data set is not considered because of the small number of possible projections on 4 attributes), seven data sets are added to the test pool (Table 5.1).

Table 5.1: Properties of the seven additional data sets. Six of them are from the UCI Machine Learning Repository, while Marketing is one of the example sets in the book of Hastie *et al.* [57].

	Examples	Discrete	Continuous	Class values
Adult	32.561	8	6	2
Glass	214	0	9	10
Marketing	8.993	13	0	9
Mushroom	8.124	22	0	7
Primary Tumor	339	17	0	21
Vehicle	846	0	18	4
WDBC[1]	569	0	20	2

[1]Wisconsin Data on Breast Cancer

We investigate to what extent the first i visualizations (ordered by the *model map* or the VizRank ranking method) improve our knowledge of the class and of the data domain in general. We are interested in how our knowledge increases after observing each new projection. The number of projections was chosen *ad hoc*, assuming an analyst would rarely view a lot of visualizations of the same type. Our aim is to prove that for the *model map* approach, the same amount of information is gained faster, therefore by viewing less visualizations as for the VizRank ranking.

First, we measured the degree of information gained after observing i projections ($i \in \{1, 2, \ldots, 15\}$). The rationale is that similar projections will separate classes in similar fashion, thus they will make similar (if not entirely equal) predictions. Since a k-NN model is induced from each projection, we can calculate the entropy of the predicted class attribute and compare it with the joint entropy of multiple projections:

$$H(P_1, \ldots, P_i) = -\sum_{c_1} \ldots \sum_{c_i} P(c_1, \ldots, c_i) \log_2 \big(P(c_1, \ldots, c_i)\big), \tag{5.2}$$

where c_i iterates over the domain of the predicted class attribute of a projection P_k and $P(c_1, \ldots, c_i)$ is the probability of the class values $c_1, \ldots, c_i$ occurring together in corresponding projections $P_1, \ldots, P_i$. For example, if the projections P_1 and P_2 are similar and give the same predictions, their entropy equals their joint entropy:

$$H(P_1) = H(P_2) = H(P_1, P_2). \tag{5.3}$$

On the contrary, the sum of entropies of completely different projections P_1 and P_2 would equal their joint entropy:

$$H(P_1) + H(P_2) = H(P_1, P_2). \tag{5.4}$$

We show that the joint entropy of the projections ranked by the *model map* technique increases faster than if the projections are ordered by VizRank. We expect the *model map* technique would guide the analyst through a more diverse set of projections, which follows directly from the definition of model distance.

Next, we discuss whether this additional knowledge contributes to the understanding of the original class attribute. We explore the relation of the information yielded by projections about the class variable. We measure how the uncertainty of the class decreases as we are given knowledge of i projections ($i \in \{1, 2, \ldots, 15\}$), namely the conditional entropy of the original class attribute given the class attributes predicted by projections:

$$H(Y|P_1, \ldots, P_i) = H(Y, P_1, \ldots, P_i) - H(P_1, \ldots, P_i), \tag{5.5}$$

where $H(Y, P_1, \ldots, P_i)$ is the joint entropy of the original class and all projected class attributes, which is computed as in Equation 5.2.

Finally, we sum up the results with the Friedman χ_F^2 statistical test showing our method is significantly better than the VizRank technique.

5.4 Ranking Scatter Plot Projections

A scatter plot and its variants [56] is one of the most used visualization methods. It uses Cartesian coordinates to display values of two attributes for a set of data in a two-dimensional plane. We mapped the class attribute to the color of the point representing a data instance. When visualizing a larger data set, the points can substantially overlap. To avoid this, we used a simple solution, altering the positions of points by a small random noise called "jitter" [23].

The case study included all possible scatter plot projections (see Table 5.2) of 11 data sets. Figure 5.1 displays the joint entropy of the scatter plot projections on the data sets. We see that the joint entropy of projections ranked with any of the *model map* methods increases faster than the VizRank for all data sets but Adult, where both methods perform similarly. In other words, projections presented by *model maps* are more diverse than those by VizRank. While a common trend can be observed for *model map* methods, hierarchical clustering with average linkage consistently yields the most diverse projections.

In Figure 5.2 we see that projections which were generated with the *model map* methods yield more information about the class than the same number of projections generated with the VizRank method. Again, this is true for all data sets except Adult, where both methods perform similarly.

We obtained the best results on Dermatology, WDBC, and Vehicle data sets, where the best 15 projections by the VizRank methods tell us only a little more than the single best projection, and for the Voting data set, where all first 15 projections are essentially the same.

Table 5.3 lists the remaining level of uncertainty about the original class when 5, 10, and 15 scatter plot projections are given. Again, the VizRank method is competitive on the Adult data set only. The values listed below are used in the significance tests in Section 5.6.

Table 5.2: The number of scatter plot projections included in the case study.

	Number of projections
Adult	91
Breast Cancer Wisconsin	36
Dermatology	561
Glass	36
Marketing	78
Mushroom	231
Primary Tumor	136
Vehicle	153
Voting	120
WDBC	190
Zoo	120

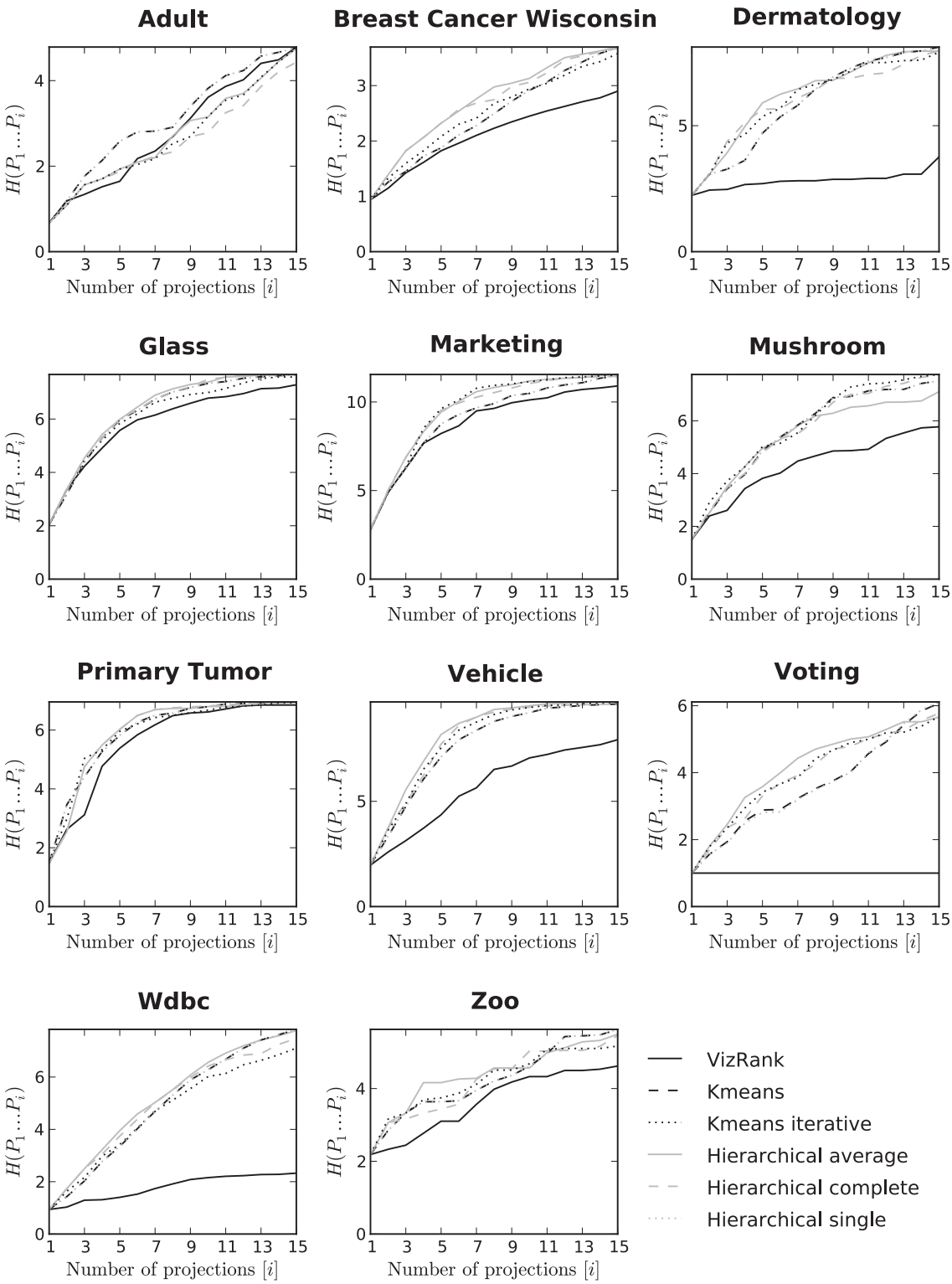


Figure 5.1: The diversity of the best i scatter plot projections, returned by the VizRank and 5 different *model map* ranking methods.

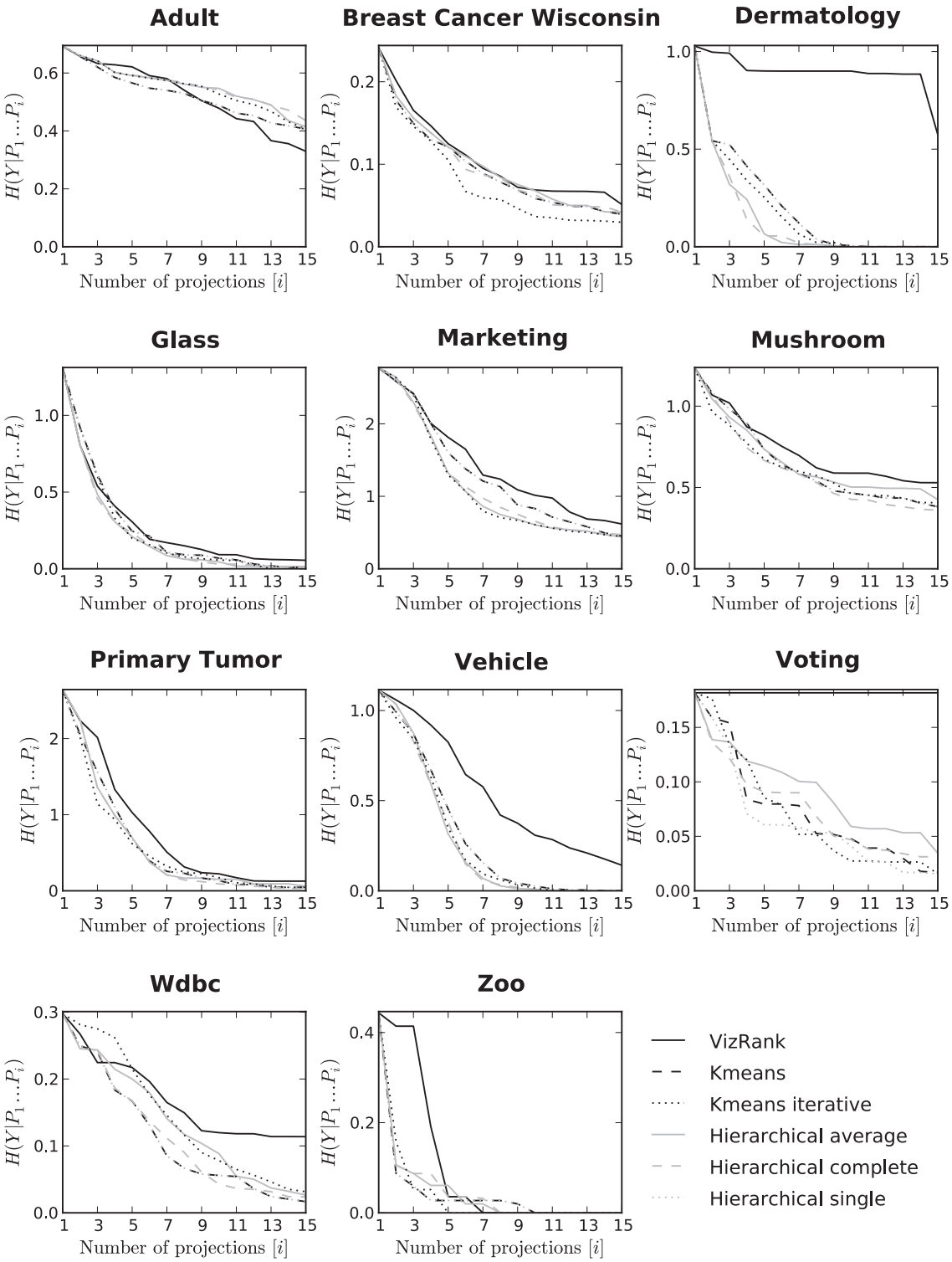


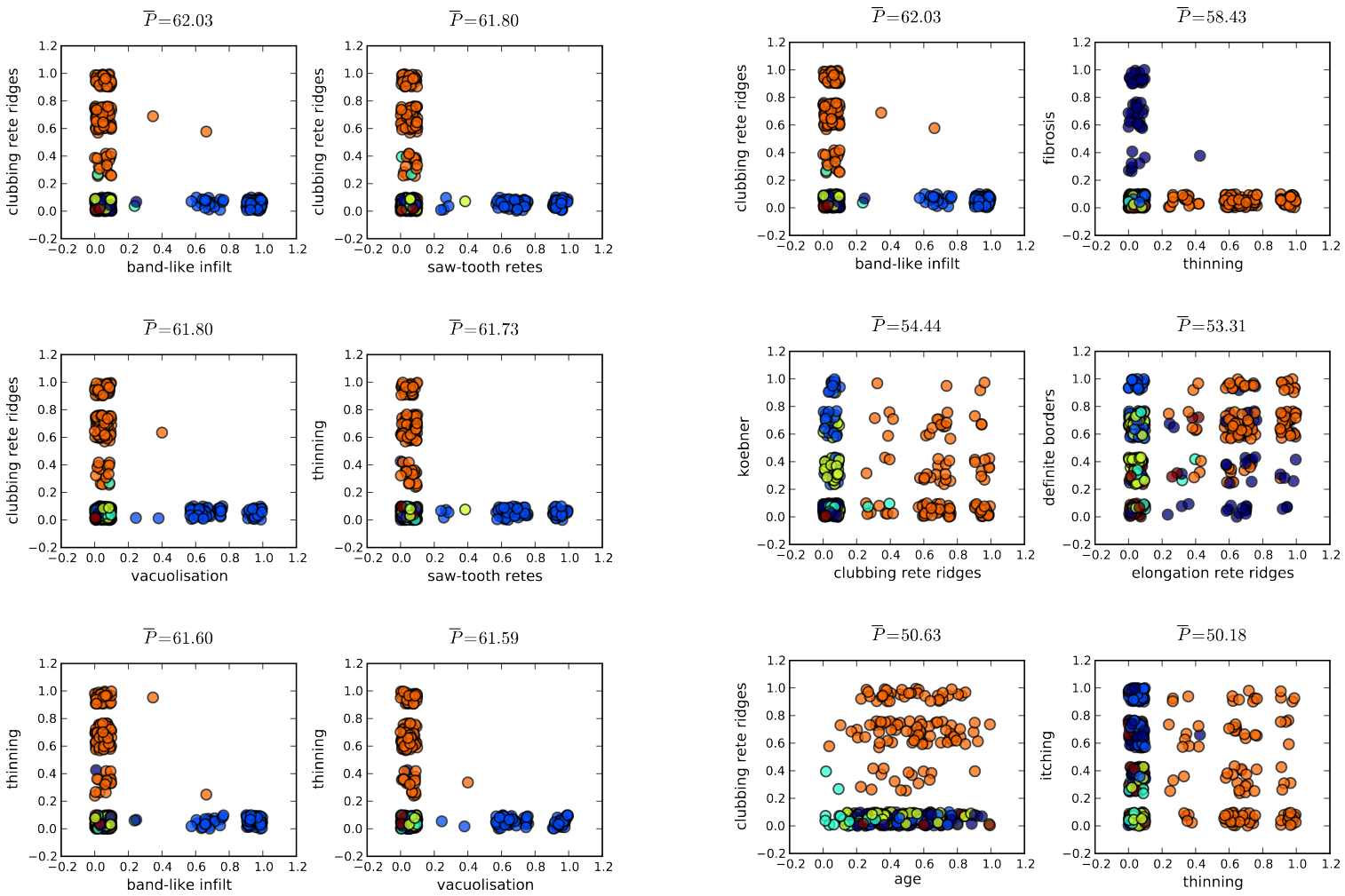
Figure 5.2: The remaining uncertainty of the original class Y by the best i scatter plot projections, returned by the VizRank and 5 different *model map* ranking methods.

An example of the first six scatter plot projections of VizRank and *model map* with hierarchical clustering (average linkage) is in Figure 5.3. VizRank projections look nearly the same, as all include the *thinning* attribute. On the contrary, the *model map* method separates the classes *pit rubra pilaris* (projection #5) and *pityriasis rosea* (projections #3 and #4) which are merged with the other classes in all of the first six VizRank projections. More examples of the scatter plot projections are listed in the Appendix C.1.

Table 5.3: The remaining level of uncertainty about the original class when 5, 10, and 15 scatter plot projections are given.

		$H(Y\|P_1...P_5)$	$H(Y\|P_1...P_{10})$	$H(Y\|P_1...P_{15})$
Adult Sample	VizRank	0.621	**0.478**	**0.330**
	K-means	**0.566**	0.489	0.406
	K-means iterative	0.591	0.528	0.406
	Hierarchical (average)	0.591	0.547	0.415
	Hierarchical (complete)	0.594	0.541	0.436
	Hierarchical (single)	**0.566**	0.489	0.406
Breast Cancer Wisconsin	VizRank	0.125	0.069	0.051
	K-means	0.121	0.058	0.039
	K-means iterative	**0.105**	**0.037**	**0.030**
	Hierarchical (average)	0.121	0.068	0.042
	Hierarchical (complete)	0.121	0.063	0.042
	Hierarchical (single)	0.121	0.058	0.039
Dermatology	VizRank	0.900	0.900	0.574
	K-means	0.317	0.006	**0.000**
	K-means iterative	0.254	**0.000**	**0.000**
	Hierarchical (average)	0.066	**0.000**	**0.000**
	Hierarchical (complete)	**0.056**	**0.000**	**0.000**
	Hierarchical (single)	0.317	0.006	**0.000**
Glass	VizRank	0.306	0.092	0.055
	K-means	0.245	0.069	0.009
	K-means iterative	**0.199**	0.057	**0.000**
	Hierarchical (average)	0.209	0.045	0.015
	Hierarchical (complete)	0.233	**0.030**	0.015
	Hierarchical (single)	0.245	0.069	0.009
Marketing	VizRank	1.813	1.014	0.618
	K-means	1.602	0.831	0.453
	K-means iterative	**1.293**	0.609	0.447
	Hierarchical (average)	1.322	**0.606**	**0.439**
	Hierarchical (complete)	1.318	0.655	0.472
	Hierarchical (single)	1.602	0.831	0.453
Mushroom	VizRank	0.820	0.588	0.529
	K-means	0.733	0.465	0.381
	K-means iterative	0.675	0.463	0.406
	Hierarchical (average)	0.736	0.502	0.426
	Hierarchical (complete)	**0.666**	**0.427**	**0.361**
	Hierarchical (single)	0.733	0.465	0.381
Primary Tumor	VizRank	1.029	0.223	0.127
	K-means	0.681	0.133	0.045
	K-means iterative	**0.620**	0.172	0.045
	Hierarchical (average)	0.699	0.152	0.061
	Hierarchical (complete)	0.699	**0.091**	**0.030**
	Hierarchical (single)	0.681	0.133	0.045

Vehicle	VizRank	0.824	0.308	0.142
	K-means	0.453	0.026	**0.000**
	K-means iterative	0.348	0.015	**0.000**
	Hierarchical (average)	**0.315**	0.008	**0.000**
	Hierarchical (complete)	0.381	**0.007**	**0.000**
	Hierarchical (single)	0.453	0.026	**0.000**
Voting	VizRank	0.182	0.182	0.182
	K-means	0.080	0.048	0.017
	K-means iterative	0.085	**0.027**	**0.016**
	Hierarchical (average)	0.115	0.059	0.034
	Hierarchical (complete)	0.091	0.047	0.031
	Hierarchical (single)	**0.061**	0.041	**0.016**
WDBC	VizRank	0.217	0.120	0.114
	K-means	**0.167**	0.056	**0.017**
	K-means iterative	0.215	0.078	0.032
	Hierarchical (average)	0.200	0.088	0.027
	Hierarchical (complete)	0.167	**0.043**	0.024
	Hierarchical (single)	**0.167**	0.056	**0.017**



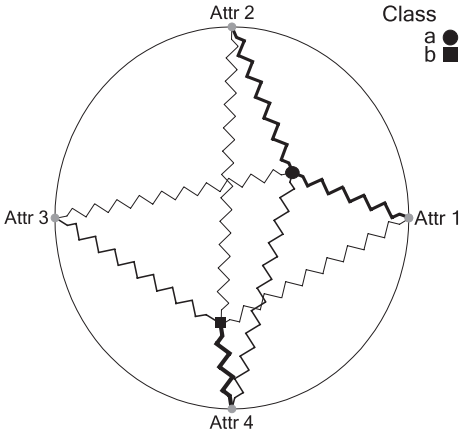
(a) VizRank

(b) *Model map (hierarchical: average linkage)*

Figure 5.3: The best six scatter plot projections of the Dermatology data set.
Node colors: orange (*psoriasis*), blue (*lichen planus*), light blue (*pit rubra pilaris*), green (*pityriasis rosea*), brown (*seboreic derm*), and violet (*cronic dermatitis*).

5.5 Ranking Radviz Projections

Radial visualization [61] is a method in which the data instances are displayed as points within a circle and the visualized attributes are represented as points, equidistantly distributed along the circle's circumference. Data point positions can be described with an analogy to the system of springs (Figure 5.4). Imagine springs attached to the data point on one end, and to the attribute point on the other. The force of each spring is determined in terms of Hook's law, determined by the corresponding attribute value. The greater the attribute value, the greater the force.



Attr 1	Attr 2	Attr 3	Attr 4	Class
0.75	0.75	0.1	0.3	a
0.2	0.1	0.3	0.95	b

Figure 5.4: The underlying system of springs of the Radviz projection.

We limited the Radviz visualization to 3 and 4 attributes, as we strive to cover the projection space as much as possible. Nonetheless, we had to limit the maximum number of projections to 9.000 in the four attribute projections of Dermatology, Mushroom, Vehicle, and the WDBC data sets (Table 5.4) due to performance and memory constraints.

Similarly to scatter plot projections, results in Figures 5.5 and 5.7 suggest a more diverse set of projections returned by the *model map* method. In more complex Radviz projections, VizRank is also outperformed on the Adult data set.

For most data sets *model map* ranking decreased the uncertainty about the original class attribute faster than the VizRank method (Figures 5.6 and 5.8). The most significant difference is for the Adult, Dermatology, Mushroom, Vehicle, and Voting data sets. Interestingly, the prediction of the animal species in the Zoo data set appears to be trivially solved with a small number of Radviz projections (4 projections for the Radviz on 3 attributes and 3 projections for the Radviz on 4 attributes).

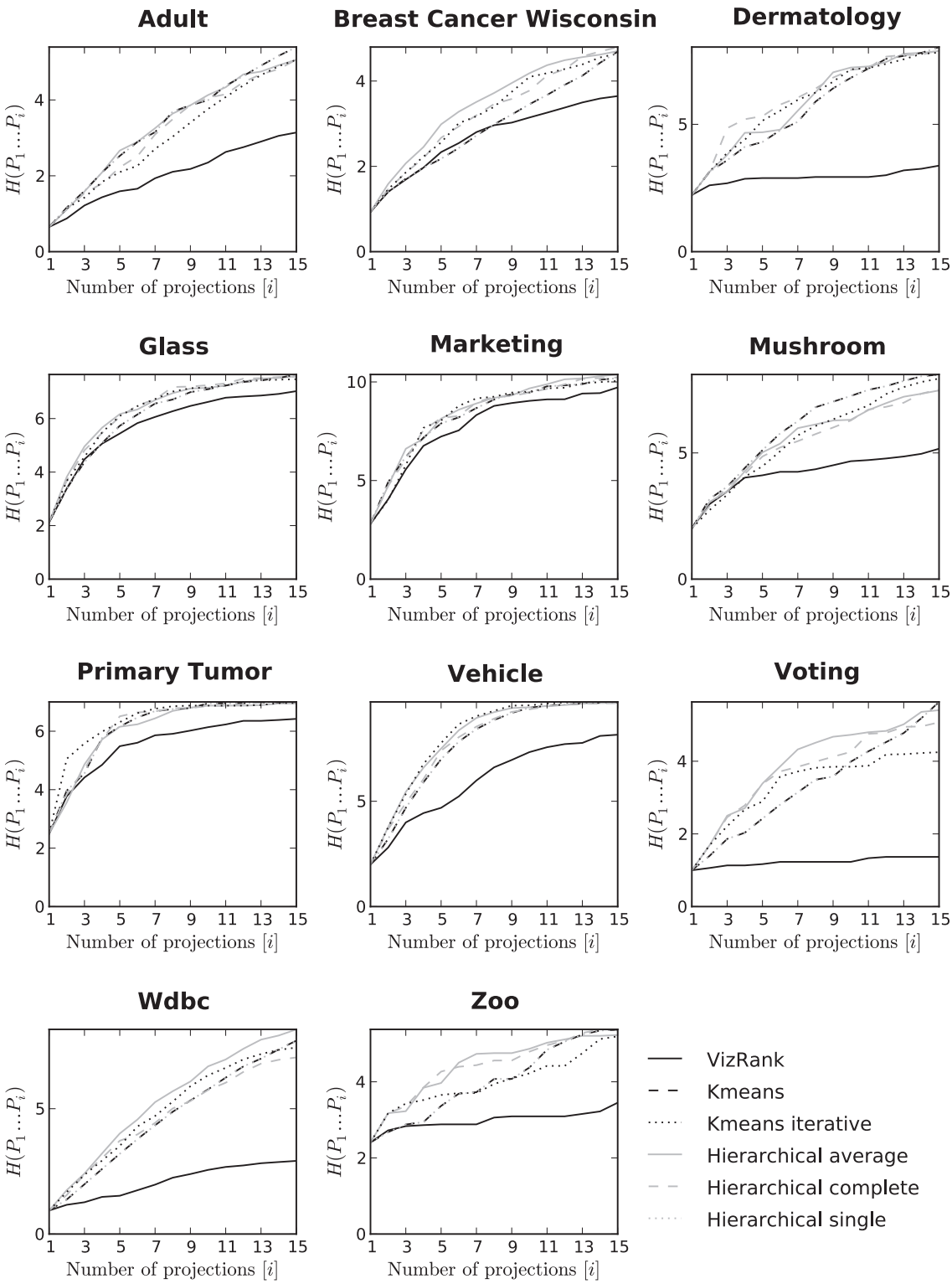


Figure 5.5: The diversity of the best i Radviz projections of 3 attributes, returned by the VizRank and 5 different *model map* ranking methods.

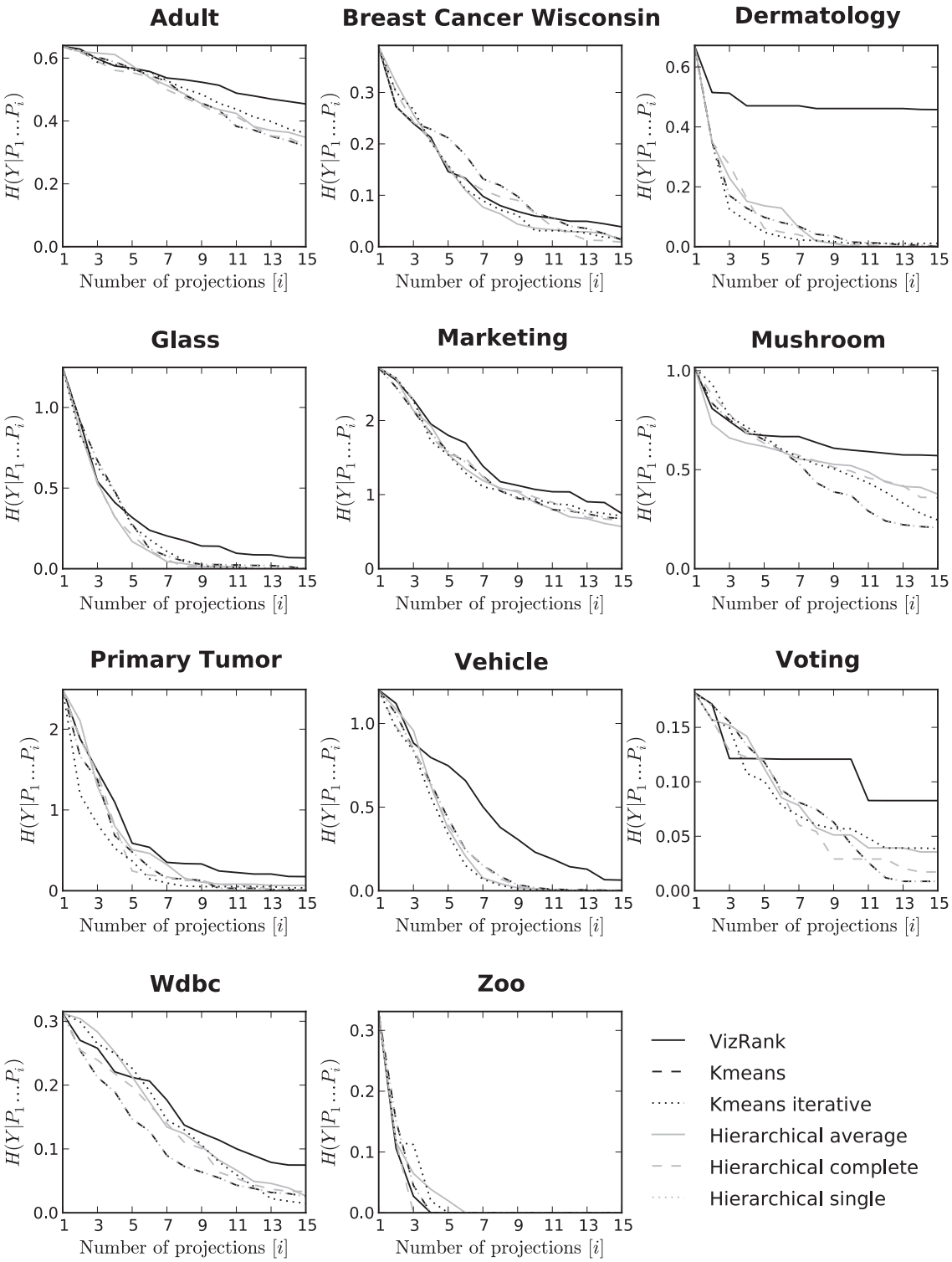


Figure 5.6: The reduction of uncertainty of the original class Y by the best i Radviz projections of 3 attributes, returned by the VizRank and 5 different *model map* ranking methods.

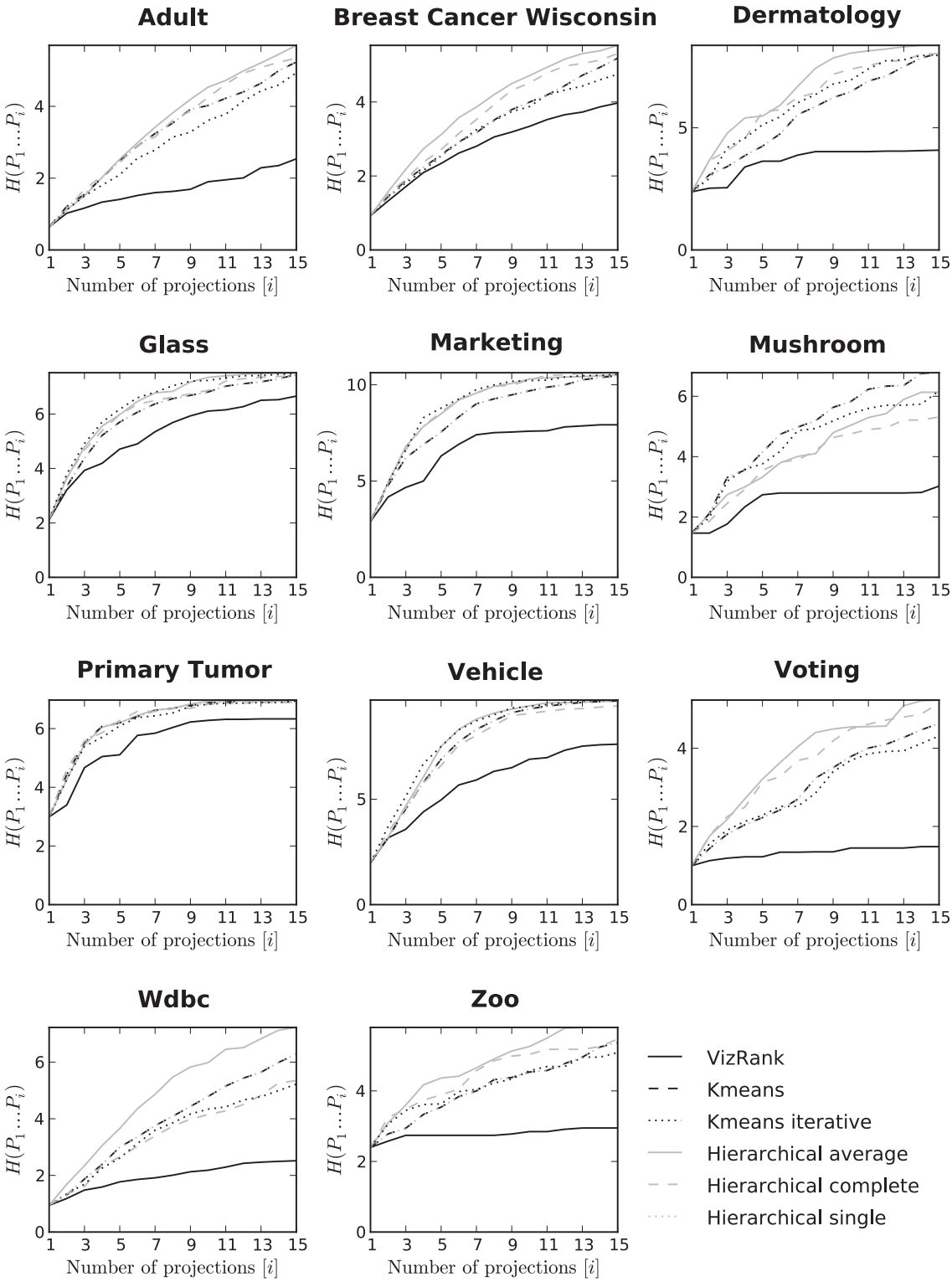


Figure 5.7: The diversity of the best i Radviz projections of 4 attributes, returned by the VizRank and 5 different *model map* ranking methods.

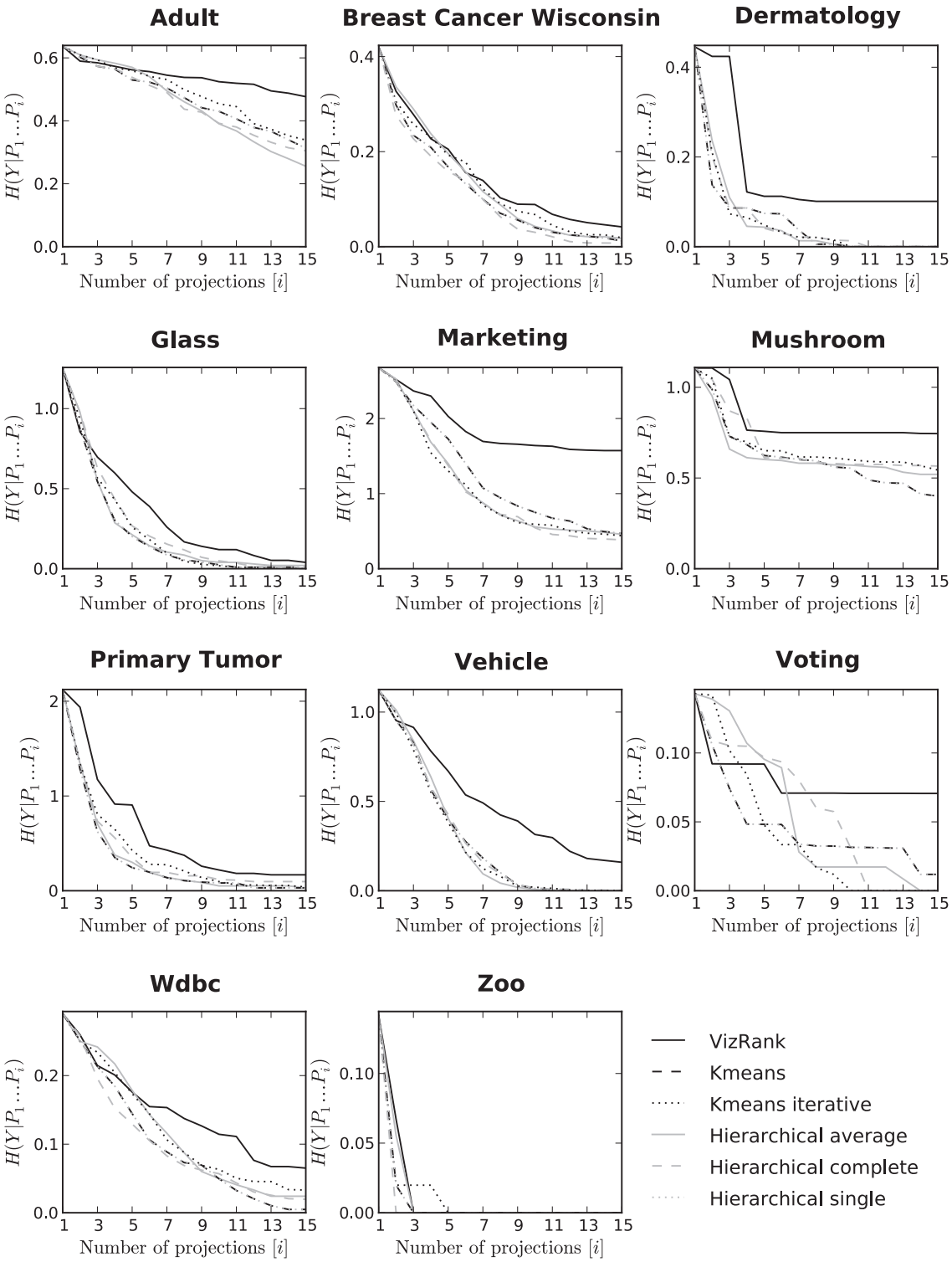


Figure 5.8: The reduction of uncertainty of the original class Y by the best i Radviz projections of 4 attributes, returned by the VizRank and 5 different *model map* ranking methods.

Tables 5.5 and 5.6 show the remaining level of uncertainty about the original class when 5, 10, and 15 Radviz projections are given. The VizRank method outperforms the *model map* ranking in one out of 66 cases. The values listed below are used in the significance test in Section 5.6.

Table 5.4: The number of Radviz projections included in the case study.

	3 attributes	4 attributes
Adult	364	3003
Breast Cancer Wisconsin	84	378
Dermatology	5.984	9.000
Glass	84	378
Marketing	286	2.145
Mushroom	1540	9.000
Primary Tumor	680	7.140
Vehicle	816	9.000
Voting	560	5.460
WDBC	1.140	9.000
Zoo	560	5.460

Table 5.5: The remaining level of uncertainty about the original class when 5, 10, and 15 Radviz projections of 3 attributes are given.

		$H(Y\|P_1...P_5)$	$H(Y\|P_1...P_{10})$	$H(Y\|P_1...P_{15})$
Adult Sample	VizRank	0.568	0.514	0.454
	K-means	0.567	0.435	**0.318**
	K-means iterative	0.562	0.456	0.361
	Hierarchical (average)	0.576	0.436	0.348
	Hierarchical (complete)	**0.554**	**0.423**	0.321
	Hierarchical (single)	0.567	0.435	**0.318**
Breast Cancer Wisconsin	VizRank	**0.146**	0.060	0.039
	K-means	0.212	0.065	0.014
	K-means iterative	0.159	**0.031**	0.017
	Hierarchical (average)	0.154	0.036	0.012
	Hierarchical (complete)	0.157	0.065	**0.009**
	Hierarchical (single)	0.212	0.065	0.014
Dermatology	VizRank	0.470	0.461	0.458
	K-means	0.098	0.015	**0.000**
	K-means iterative	**0.048**	0.011	0.011
	Hierarchical (average)	0.137	**0.000**	**0.000**
	Hierarchical (complete)	0.062	0.006	0.000
	Hierarchical (single)	0.098	0.015	**0.000**
Glass	VizRank	0.318	0.139	0.067
	K-means	0.277	0.025	**0.000**
	K-means iterative	0.264	**0.009**	**0.000**
	Hierarchical (average)	**0.168**	0.013	**0.000**
	Hierarchical (complete)	0.209	0.013	**0.000**
	Hierarchical (single)	0.277	0.025	**0.000**
Marketing	VizRank	1.800	1.070	0.745
	K-means	1.584	0.912	0.677
	K-means iterative	**1.536**	0.940	0.711
	Hierarchical (average)	1.543	**0.893**	**0.569**
	Hierarchical (complete)	1.542	0.964	0.651
	Hierarchical (single)	1.584	0.912	0.677
Mushroom	VizRank	0.672	0.599	0.571
	K-means	0.650	**0.371**	**0.207**
	K-means iterative	0.667	0.470	0.246
	Hierarchical (average)	**0.617**	0.520	0.377
	Hierarchical (complete)	0.634	0.493	0.359
	Hierarchical (single)	0.650	**0.371**	**0.207**

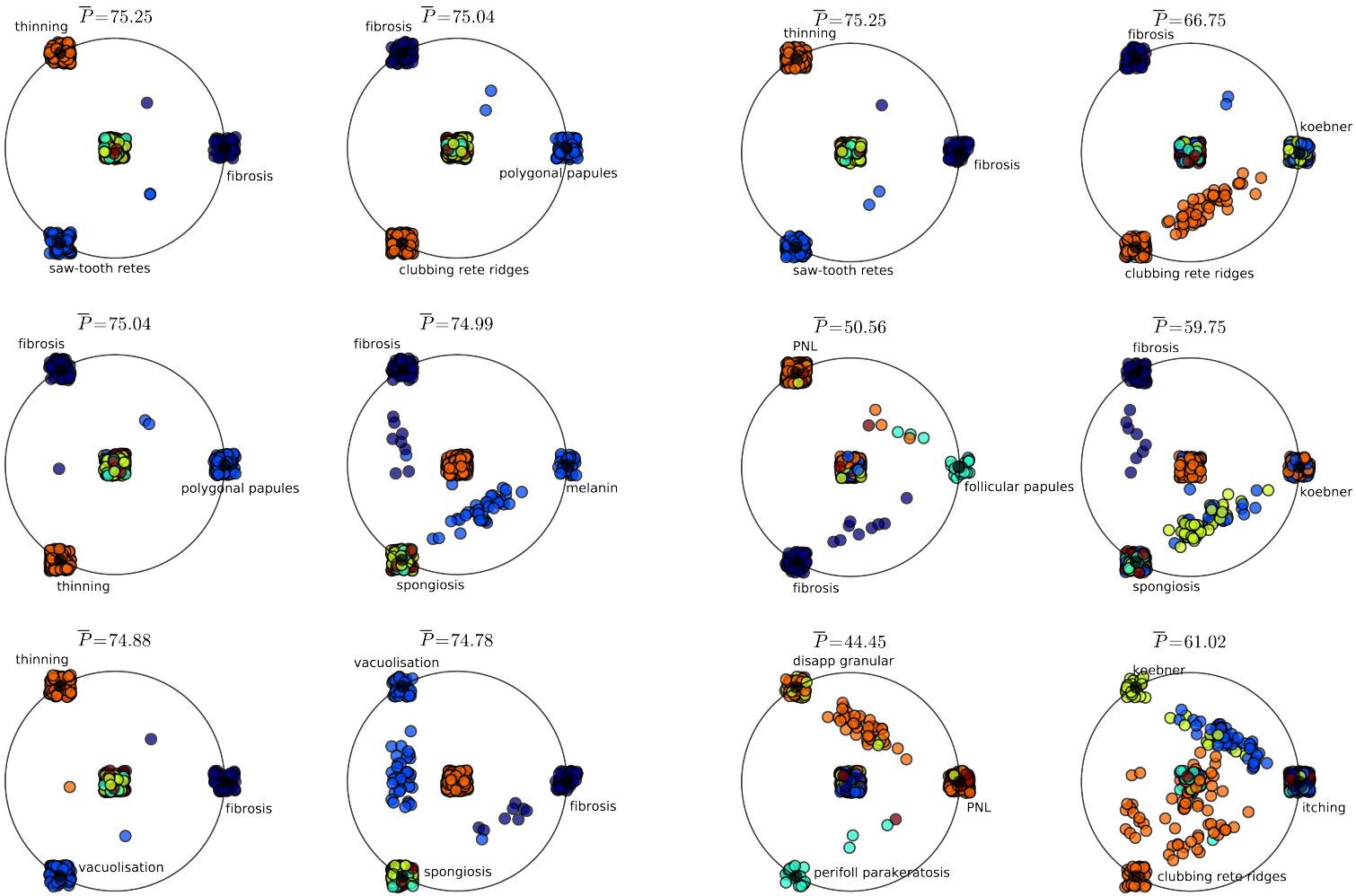
Primary Tumor	VizRank	0.589	0.245	0.175
	K-means	0.480	**0.030**	**0.015**
	K-means iterative	0.368	0.051	0.036
	Hierarchical (average)	0.508	0.081	0.066
	Hierarchical (complete)	**0.244**	0.072	**0.015**
	Hierarchical (single)	0.480	**0.030**	**0.015**
Vehicle	VizRank	0.748	0.231	0.064
	K-means	0.432	0.018	0.002
	K-means iterative	**0.337**	**0.006**	**0.000**
	Hierarchical (average)	0.371	0.013	**0.000**
	Hierarchical (complete)	0.394	0.009	**0.000**
	Hierarchical (single)	0.432	0.018	0.002
Voting	VizRank	0.121	0.121	0.083
	K-means	0.118	0.043	**0.009**
	K-means iterative	**0.101**	0.057	0.039
	Hierarchical (average)	0.113	0.051	0.036
	Hierarchical (complete)	0.120	**0.029**	0.017
	Hierarchical (single)	0.118	0.043	**0.009**
WDBC	VizRank	0.212	0.114	0.075
	K-means	**0.147**	**0.055**	0.026
	K-means iterative	0.226	0.080	**0.015**
	Hierarchical (average)	0.213	0.082	0.026
	Hierarchical (complete)	0.198	0.064	0.033
	Hierarchical (single)	**0.147**	**0.055**	0.026

Table 5.6: The remaining level of uncertainty about the original class when 5, 10, and 15 Radviz projections of 4 attributes are given.

		$H(Y\|P_1...P_5)$	$H(Y\|P_1...P_{10})$	$H(Y\|P_1...P_{15})$
Adult Sample	VizRank	0.562	0.524	0.477
	K-means	**0.530**	0.430	0.313
	K-means iterative	0.560	0.454	0.339
	Hierarchical (average)	0.570	**0.391**	**0.256**
	Hierarchical (complete)	0.538	0.392	0.305
	Hierarchical (single)	**0.530**	0.430	0.313
Breast Cancer Wisconsin	VizRank	0.205	0.089	0.042
	K-means	0.168	0.040	0.012
	K-means iterative	0.193	0.068	0.019
	Hierarchical (average)	0.196	0.043	0.018
	Hierarchical (complete)	**0.158**	**0.030**	**0.007**
	Hierarchical (single)	0.168	0.040	0.012
Dermatology	VizRank	0.112	0.101	0.101
	K-means	0.074	**0.000**	**0.000**
	K-means iterative	0.049	**0.000**	**0.000**
	Hierarchical (average)	0.044	**0.000**	**0.000**
	Hierarchical (complete)	**0.041**	0.013	**0.000**
	Hierarchical (single)	0.074	**0.000**	**0.000**
Glass	VizRank	0.480	0.120	0.040
	K-means	**0.201**	0.025	0.009
	K-means iterative	0.259	**0.019**	**0.000**
	Hierarchical (average)	0.212	0.040	0.020
	Hierarchical (complete)	0.267	0.049	**0.000**
	Hierarchical (single)	**0.201**	0.025	0.009
Marketing	VizRank	2.029	1.640	1.573
	K-means	1.728	0.751	0.472
	K-means iterative	**1.302**	0.591	0.440
	Hierarchical (average)	1.400	0.557	0.459
	Hierarchical (complete)	1.374	**0.545**	**0.385**
	Hierarchical (single)	1.728	0.751	0.472
Mushroom	VizRank	0.757	0.750	0.745
	K-means	0.624	**0.554**	**0.400**
	K-means iterative	0.650	0.600	0.544
	Hierarchical (average)	**0.602**	0.571	0.520
	Hierarchical (complete)	0.611	0.575	0.566
	Hierarchical (single)	0.624	**0.554**	**0.400**

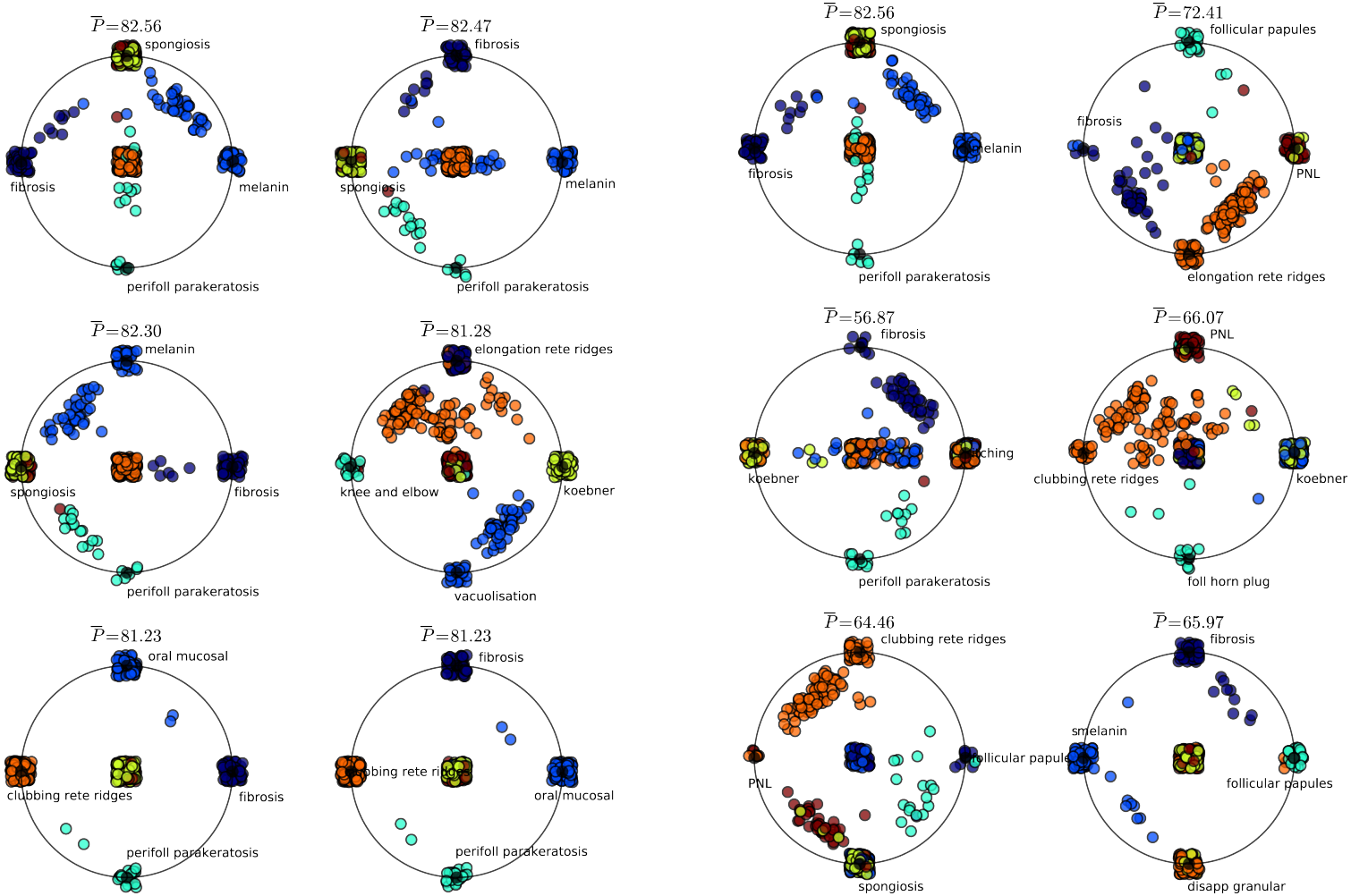
Primary Tumor	VizRank	0.905	0.219	0.168
	K-means	**0.243**	0.082	**0.030**
	K-means iterative	0.427	0.097	0.040
	Hierarchical (average)	0.300	**0.051**	**0.030**
	Hierarchical (complete)	0.351	0.118	0.097
	Hierarchical (single)	**0.243**	0.082	**0.030**
Vehicle	VizRank	0.669	0.315	0.159
	K-means	0.399	0.013	**0.000**
	K-means iterative	**0.383**	0.020	**0.000**
	Hierarchical (average)	0.406	**0.007**	**0.000**
	Hierarchical (complete)	0.427	0.017	**0.000**
	Hierarchical (single)	0.399	0.013	**0.000**
Voting	VizRank	0.092	0.071	0.071
	K-means	0.048	0.032	0.012
	K-means iterative	**0.048**	**0.000**	**0.000**
	Hierarchical (average)	0.095	0.017	**0.000**
	Hierarchical (complete)	0.097	0.031	**0.000**
	Hierarchical (single)	0.048	0.032	0.012
WDBC	VizRank	0.176	0.114	0.065
	K-means	0.145	**0.050**	**0.005**
	K-means iterative	0.174	0.063	0.033
	Hierarchical (average)	0.179	0.050	0.024
	Hierarchical (complete)	**0.129**	0.057	0.020
	Hierarchical (single)	0.145	**0.050**	**0.005**

Examples of the first six Radviz projections of VizRank and *model map* with hierarchical clustering (average linkage) are in Figures 5.9 and 5.10. In Figure 5.9 (Radviz on 3 attributes), for example, the VizRank method lists two groups of essentially identical projections (#1, #2, #3, #5 and #4, #6) that separate only 4 out of 6 classes. *Model map*, on the contrary, includes 4 projections that highlight also the two "missing" classes (*pit rubra pilaris* in projections #3 and #5 and *pytiriasis rosea* in projections #4 and #6). More examples of the Radviz projections are listed in the Appendices C.2 and C.3.



(a) VizRank (b) *Model map (iterative k-means)*

Figure 5.9: The best six Radviz projections (3 attributes) of the Dermatology data set.
Node colors: orange (*psoriasis*), blue (*lichen planus*), light blue (*pit rubra pilaris*), green (*pityriasis rosea*), brown (*seboreic derm*), and violet (*cronic dermatitis*).



(a) VizRank (b) *Model map (iterative kmeans)*

Figure 5.10: The best six Radviz projections (4 attributes) of the Dermatology data set.
Node colors: orange (*psoriasis*), blue (*lichen planus*), light blue (*pit rubra pilaris*), green (*pityriasis rosea*), brown (*seboreic derm*), and violet (*cronic dermatitis*).

5.6 Map of Projections

Experiments show that the *model map* ranking has the potential to assist an analyst to learn more about the problem domain—and learn faster. Diagrams in Figures 5.3, 5.9, and 5.10 suggest that a diverse set of projections is indeed better than a set of the best, yet similar visualizations. *Model map* ranking methods outperformed VizRank in virtually every experiment, listed in Tables 5.3, 5.5, and 5.6. In this section we statistically test the observed differences and compare the performance of different clustering methods.

We compare the conditional uncertainty of the original class attribute $H(Y|P_1,\ldots,P_i)$ in three separate tests, given 5, 10, and 15 projections. Results of the three visualization types from Tables 5.3, 5.5, and 5.6 are merged into 6 testing samples, representing 6 ranking methods, each with 33 (3×11) data set scores (samples).

The equality of the means of several groups is commonly tested with the analysis of variance (ANOVA) [68]. In our case, normality could not be guaranteed due to the small sample size. We will use the non-parametric analogue, the Friedman χ_F^2 test [43, 44]. It ranks the algorithms and compares their average ranks. The null-hypothesis states that all the algorithms are equivalent, so their ranks are equal.

If the null-hypothesis is rejected, we will test for significant differences between the algorithms with the post-hoc Nemenyi test [88]. The differences will be represented with the critical distance diagram [31]. We will connect the non-significant differences with line segments (Figure 5.11). The critical distance is computed by:

$$\mathrm{CD} = q_\alpha \sqrt{\frac{k(k+1)}{N}}, \tag{5.6}$$

where the q_α is the critical value of the Nemenyi test for the selected significance level α, k is the number methods, and N is the number of tested samples (in our case: $\alpha = 0.05$, $k = 6$, and $N = 33$).

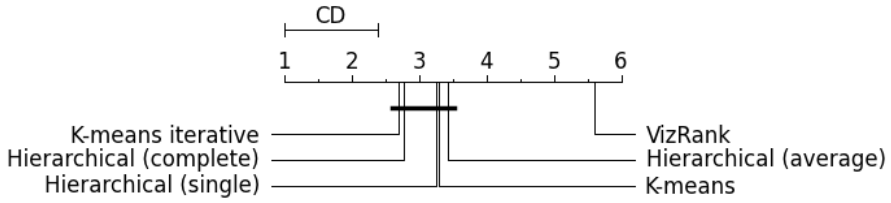
Friedman test rejected the three null-hypotheses with p-value $< 10^{-8}$ (Table 5.7), suggesting some algorithms do differ.

Table 5.7: Results of the Friedman χ_F^2 test. Tables 5.3, 5.5, and 5.6 are merged into 6 testing samples, each with 33 (3×11) data set scores. The conditional uncertainty of the original class attribute $H(Y|P_1,\ldots,P_i)$ is compared in three separate tests for $i \in \{5, 10, 15\}$.

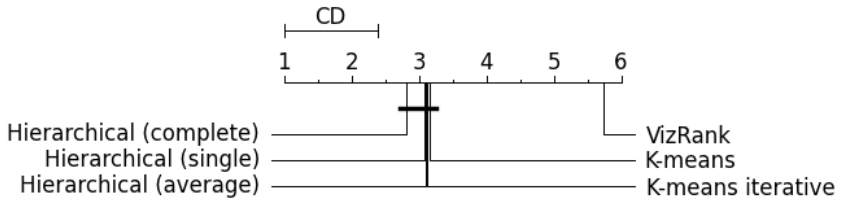
	χ_F^2	p-value
$P_1 ... P_5$	50.668	1.0×10^{-9}
$P_1 ... P_{10}$	54.141	2.0×10^{-10}
$P_1 ... P_{15}$	67.024	4.3×10^{-13}

The results of the post-hoc Nemenyi tests are in Figure 5.11. The *model map* ranking methods perform significantly better than the VizRank method. There are, however, no significant differences among the different clustering techniques.

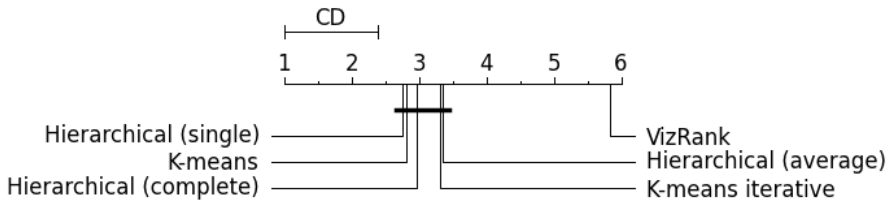
Another interesting application of the model distance measure is the visual representation—a map of the projection space. Instead of scrolling down the list of projections, an analyst could explore an interactive network of organized projections. The example in Figure 5.12 presents the best 15 projections of the three projection types applied on the Dermatology data set, connected into a network. Nodes are colored by the projection type (scatter plots are gray, green and orange are Radviz projections of 3 and 4 attributes respectively) and node size represents the quality of projection (average probability $\overline{P}$ assigned to the correct class). The projected attributes are listed below each node. Apart from the visualization and practical use, the idea of projection graphs has not been thoroughly researched yet. Whether explorative analysis like this will prove useful is left for the future work.



(a) Considering the first 5 projections.



(b) Considering the first 10 projections.



(c) Considering the first 15 projections.

Figure 5.11: Comparison of the 6 ranking methods with the Nemenyi test. Methods that are not significantly different are connected.

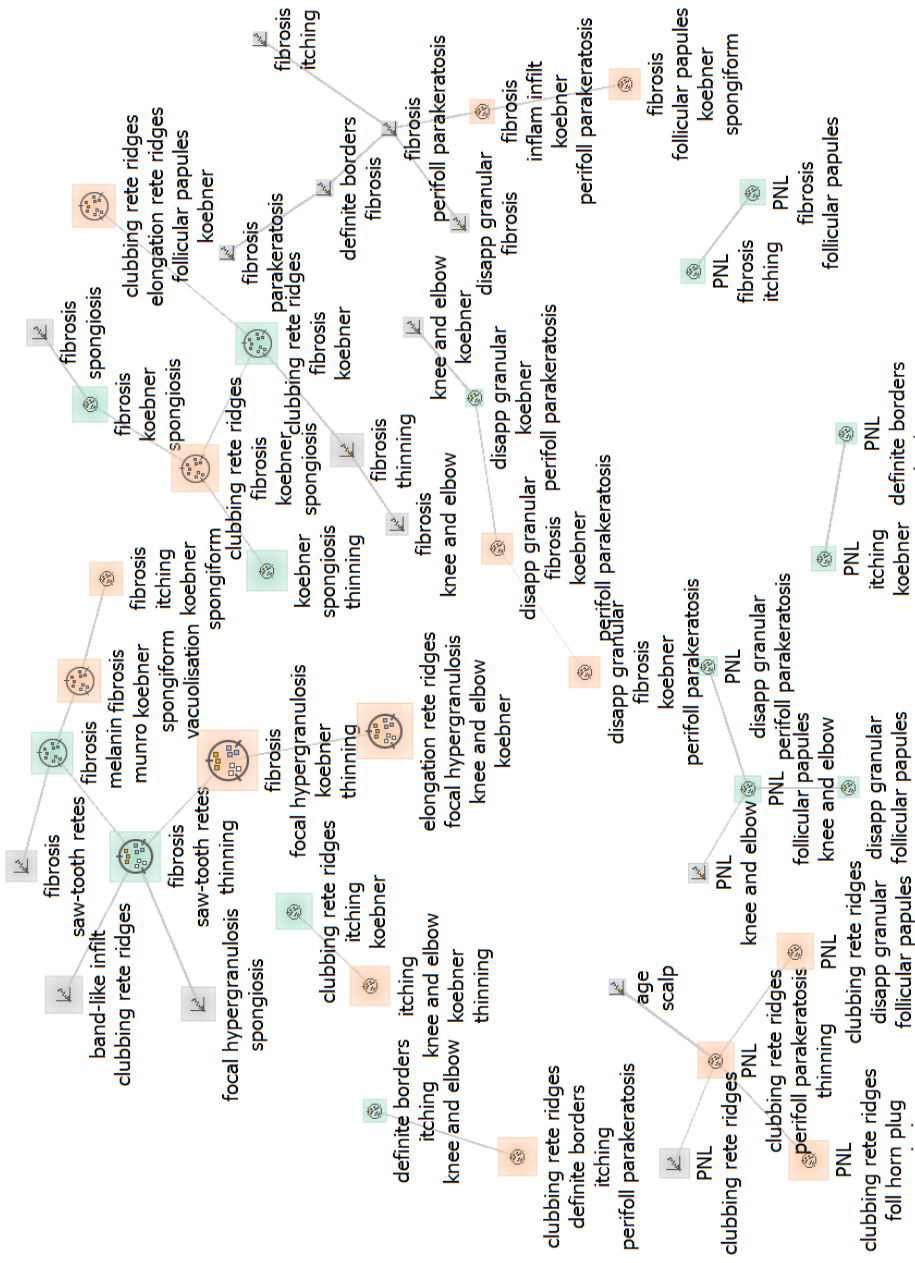


Figure 5.12: The map of projection space. The network consists of the best 15 projections of the three projection types, each connected to the nearest projection, according to the model distance measure. It is laid out with the FragViz layout technique, discussed in Section 2.3, hence similar clusters are positioned together. Nodes are colored by projection type (scatter plots are gray; green and orange are Radviz projections of 3 and 4 attributes, respectively) and node size represents the quality (average probability $\overline{P}$) assigned to the correct class. Corresponding attributes are listed below each projection.

Chapter 6

Exploring Ensembles of Classifiers

Ensemble learning is a strategy that combines a set of different and simple classifiers into an integrated classifier. In other words, an ensemble combines many weak learners to hopefully form a strong learner. Ensembles are supervised learning techniques that are increasingly popular because of their superior predictive performance. Interpreting ensemble methods, though, is not a trivial task due to their complex nature. They are commonly thought of as *black box* learning algorithms.

We describe how to create a *model map* of the Random Forest [18] ensemble learning technique. In Section 6.2 we explore the forests of two data sets: Zoo and Marketing, and show how different parts of the map "specialize" on the prediction of different classes. We discover some trees are redundant and use the model distance-based algorithm to build the forest with a smaller number of trees, but with equally good prediction performance.

6.1 Ensemble Learning

Empirically, ensembles perform best when the models are significantly diverse [104, 79]. Many ensemble methods therefore seek to promote diversity among the models they combine [20, 1], although non-intuitive, more random algorithms tend to produce a stronger ensemble. Common types of ensemble methods are:

- **Bootstrap aggregating (bagging)** [17] generates a number of new training sets by sampling instances uniformly and with replacement. Models are fit to the new training sets (bootstrap samples) and combined by averaging the output. The random forest algorithm is a special case of the bagging idea combined with the random selection of features. It assembles a collection of decision trees and outputs the class with the most votes over the classification of individual trees.

- **Boosting** developed from the original idea of Kearns to create a single strong learner out of a set of weak learners [71]. It incrementally learns weak classifiers and add them to a final strong classifier. After each new weak learner, the misclassified instances are weighted higher, giving focus to the future weak learners. The first algorithm (and the most significant historically) is Adam boost [40].

- **Bayesian model averaging** is an approximation of the Bayes Optimal Classifier by sampling hypotheses and combining them using the Bayes' law [60]. Hypotheses are typically sampled using a Monte Carlo sampling technique such as MCMC. It tends to over-fit the data and does not perform as well as other (simpler) ensemble techniques.

- **Bucket of models** is a model selection algorithm that chooses the best model for the given data set out of a collection of models. It performs as well as the best algorithm in the set. However, when used on multiple data sets it will produce better results (on average) than any model from the set.

- **Stacking** combines the predictions from multiple models [118]. It works best when the models included are of very different types. Stacking applies a two-level structure: the base-level classifiers output their own predictions and the meta-level classifier takes the outputs as its input to generate a final decision.

The *model map* technique could be used to explore any kind of ensemble method. We limit ourselves on the random forest algorithm and leave other ensemble types for future work.

6.2 Ensemble Visualization

Many existing visualization techniques are adapted to visualize *black box* models or the random forest in particular. Variable importance plot shows the difference of the correct votes between the original classifier and one with the permuted values of an attribute. Marginal or partial dependence plots [58] provide another approach to visualizing results. They visualize the effects of small numbers of attributes on the predictions of classification or regression tools, and highlight the relationship between the given attribute and the response averaged over the joint values of the other attributes. In addition to the relationship between predictors and responses, we can examine a confusion matrix of predictive performance. We can inspect the out-of-bag error rate as a function of the number of trees in order to determine how many trees to grow.

Random forest produces measures of similarity of data points called proximities. After the tree is grown, all of the data instances are put down the tree. The proximity of two points is increased by one if they are in the same terminal node. We normalize

proximities in the end by dividing them with the number of trees. A distance matrix of proximities is commonly visualized with the multidimensional scaling (MDS) technique [26], an optimization procedure which assigns a location to each instance in the (usually) two-dimensional space, where the distance between points in the plane reflects their similarity.

Breiman suggested using linked visualizations to explore random forests. Points visualized with the multidimensional scaling can be colored and brushed. The brushed data points can be highlighted in an associated parallel coordinate displays or heat maps. Brushing allows us to easily determine attribute importance in specific regions of the data, and to look at the input attributes and attribute importance in the parallel coordinate plot for potential subgroups of data. Heat maps are used to represent the votes for each class, and for interactions and proximity matrices.

Nomograms are another technique which highlight the attribute importance. In the words of Hankins: "they are not an uncertain novelty, but a milestone in the history of visualization [54]." Nomograms were successfully applied to visualize naive Bayesian classifiers [87], support vector machines [66], and such. Štrumbelj proposed a method for visualizing and explaining *black box* classification and regression models and their predictions for individual instances [113, 108]. The method successfully reveals how individual features influence the model and can be used with any type of a *black box* regression model in a uniform way.

Existing methods are focused on the indirect "black-box-like" visualization of the random forest. There are cases, however, where showing how trees are built and how similar they are might be of interest, especially for the researcher or a domain expert who is not used to machine learning techniques.

6.3 Visualization of Random Forests

With *model map* techniques we can visualize the relations between individual trees. Different regions of the *model map* (clusters of trees) or individual trees can be explored in an interactive way within the `Model Map` widget. The map puts similar trees together, so different regions of the forest make different predictions.

6.3.1 Zoo Data Set

Figure 6.1 shows a screen capture of the `Model Map` widget. A map of a random forest on the Zoo data set is on the right side. Half of the data was used to learn the trees, whereas *model map* distances and various performance measures were estimated from the other half. The ensemble consists of 100 classification trees, which more than suffices for the optimal classification performance (Figure 6.2) on this domain.

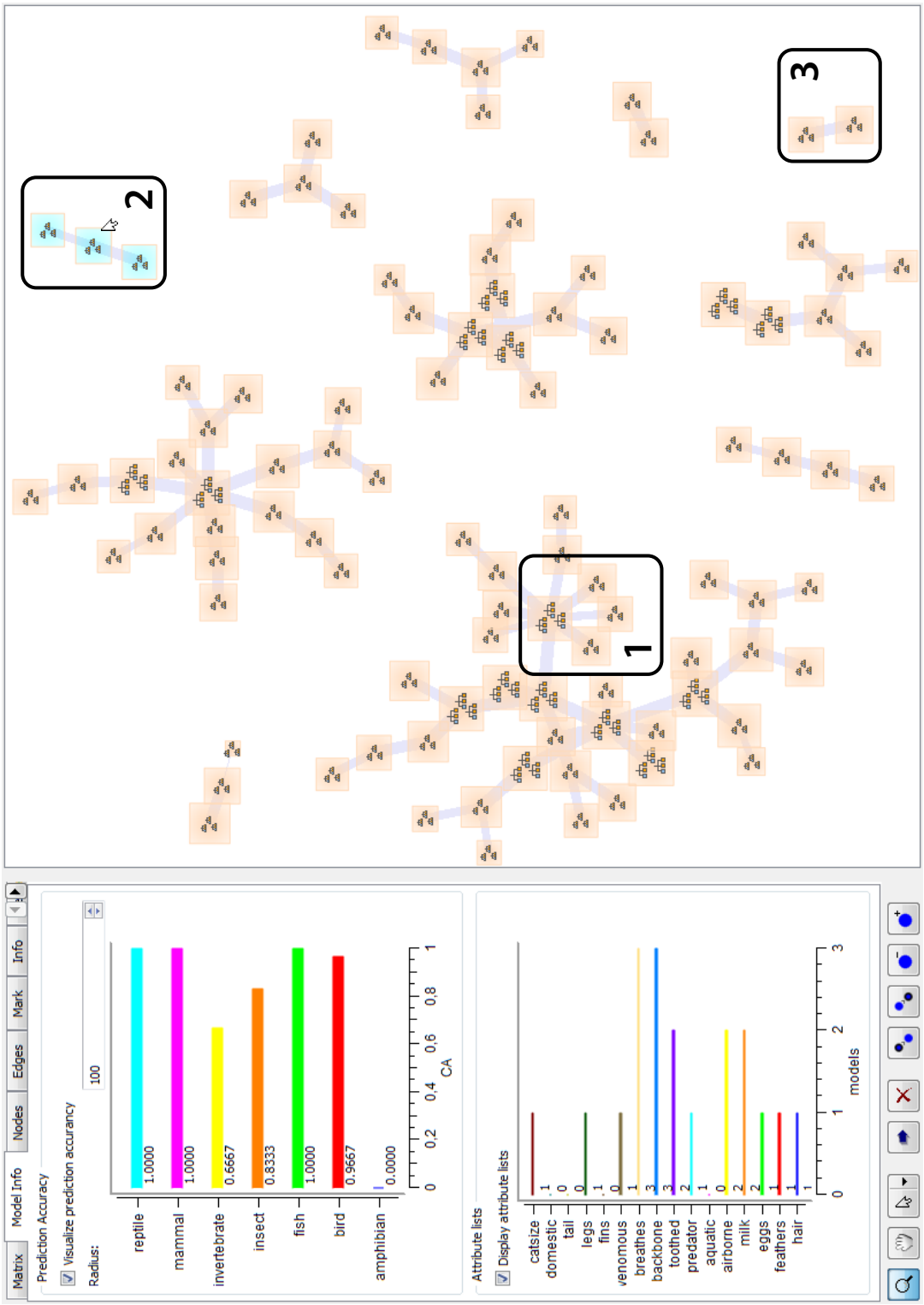


Figure 6.1: A screen capture of the `Model Map` widget, displaying a map of random forest on the Zoo data.

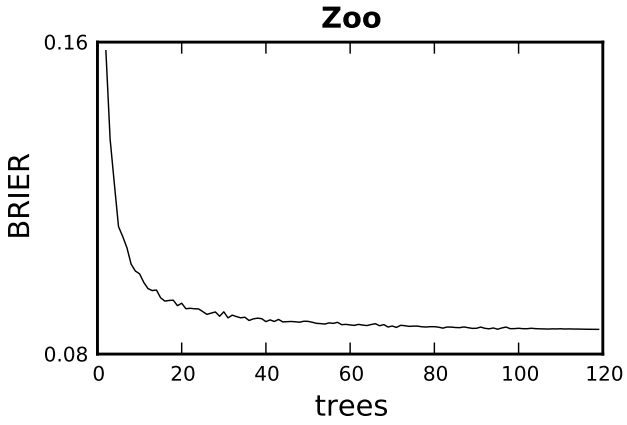


Figure 6.2: The Brier score for the random forest classifier on the Zoo data as a function of the number of trees grown.

We assessed the performance of models by the Brier score [19], which is applicable on the multi-valued class attribute as well:

$$BS = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \sum_{c \in \mathcal{C}} (p_c(e) - o_e)^2, \tag{6.1}$$

in which $\mathcal{C}$ is the set of possible classes, $p_c(e)$ is the probability that was forecast for class c and instance e, and o_e indicates whether e indeed belongs to the class c (1 if it does and 0 if not).

Each tree is connected with the most similar one according to the Manhattan model distance measure and edge width is proportional to the similarity. Node size is inversely proportional to the Brier score, evaluated on the test data set. The network is visualized with the FragViz method.

Moving the mouse around the map interactively marks nodes in the vicinity. In the example in Figure 6.1 three nodes are marked with light blue color. A summary of the marked trees is in the left part of the image. At the top, there is the average classification accuracy of a particular class, whereas the influence of individual attributes is displayed below. A simple count of the occurrence of the attribute in the nodes of marked trees was used as the measure of attribute's influence.

The classes *bird*, *fish*, and *mammal* are well predictable all around the map. This particular set of trees is specialized in *reptiles*, but never predicts an *amphibian*. Not at all surprising, as the "tail" and "aquatic" attributes—which separate the two classes—are not used in this part of the forest.

To illustrate how we can analyze different parts of the forest using the proposed methods, we selected three regions of the forests. Average classification accuracies (CA) and attribute importance of models in regions one to three are listed in Tables 6.1(a) and 6.1(b). The CA scores are given for each of the seven classes. Region 1 is focused on *invertebrates*. The "feathers" and "toothed" attributes are

Table 6.1: Average classification accuracy and attribute importance for the classification models in the selected regions in Figure 6.1.

(a) classification accuracies

	Region 1	Region 2	Region 3
reptile	**0.25**	**1.00**	**0.25**
mammal	1.00	1.00	0.95
invertebrate	**1.00**	0.67	0.60
insect	0.69	0.83	**1.00**
fish	1.00	1.00	1.00
bird	1.00	0.97	1.00
amphibian	0.75	**0.00**	1.00

(b) attribute importance

	Region 1	Region 2	Region 3
catsize	3	1	0
domestic	0	0	0
tail	0	0	0
legs	2	1	**2**
fins	2	0	0
venomous	0	1	0
breathes	**4**	**3**	0
backbone	1	**3**	1
toothed	**4**	2	1
predator	0	1	1
aquatic	1	0	1
airborne	1	2	1
milk	2	2	0
eggs	1	1	**2**
feathers	**4**	1	1
hair	1	1	0

widely used in this models, as they are the key attributes to separate *invertebrates* and *fishes* from other animals.

Region 3 is specialized in *amphibians*—the smallest class with only four instances —and pays for it by doing worst for the other two less common classes (*reptiles* and *invertebrates*). Key attributes in this case are "'eggs" and "legs," as all *amphibians* lay eggs and have 4 legs.

In addition to reviewing the model we can play an active role in the model construction. Regions of trees can be selected to form a new random forest classifier which can be sent forward to other widgets for further analysis of a specific sub-domain. Trees can be excluded or included in the forest to fine tune the classifier by a single mouse click. The `Model Map` widget is a versatile tool to explore and play with ensembles of classifiers.

6.3.2 Marketing Data Set

The Marketing data was extracted from the survey, in which a total of $9,409$ questionnaires containing 502 questions were filled out by shopping mall customers in the San Francisco Bay area. It consists of 13 demographic attributes. The data set is a good mixture of categorical and continuous variables. The class attribute ("Annual Income of Household") has 9 values, ranging from 1 (less than $\$10,000$) to 9 ($\$75,000$ or more).

A *model map* is in Figure 6.4(a). The 100 classification trees suffice for the optimal classification performance (Figure 6.3). Similarly to the Zoo case study, half of the data was used to learn the trees, whereas the *model map* distances and various performance measures were estimated on the other half. The Manhattan model

distance measure was used to connect each tree with the most similar one. Edge width is proportional to the similarity, node size represents the Brier score, and the network is visualized with the FragViz method.

Tables 6.2(a) and 6.2(b) list an overview of classification accuracies and attribute importance of the three exposed regions. In Region 1 as in most of the map, attributes "years in SF" and "education" are used the most. Segments of people who earn less than $\$10,000$ and from $\$50,000$ to $\$74,999$ are equally well predictable in all of the model space. Region 2 has a rather high accuracy also for the $\$20,000$ to $\$24,999$ segment, which is most likely the consequence of extensive use of the "ethnic class" attribute. Region 3 is specialized in the $\$30,000$ to $\$39,999$ segment and, among the two most used attributes in the map, extensively uses the "household members" attribute.

In a random forest, the classification score typically improves with the number of trees grown until it stabilizes at its limit. For the Marketing domain, having more than 100 trees will not increase the accuracy of the forest. We observe next how the

Table 6.2: Average classification accuracy and attribute importance for the classification models in the selected regions in Figure 6.4(a).

(a) classification accuracies

	Region 1	Region 2	Region 3
1	**0.81**	**0.81**	**0.81**
2	0.53	0.53	0.53
3	0.55	0.55	0.53
4	0.49	**0.56**	0.51
5	0.49	0.50	0.44
6	0.51	0.49	**0.57**
7	0.47	0.44	0.48
8	**0.59**	0.57	0.55
9	0.51	0.52	0.57

(b) attribute importance

	Region 1	Region 2	Region 3
language	108	73	46
ethnic class	308	197	134
type of home	314	172	119
household status	199	61	78
under 18	254	147	107
household members	225	166	151
dual income	182	117	71
years in sf	**361**	**224**	**135**
occupation	269	177	2
education	**362**	**202**	**152**
age	63	15	117
material status	270	132	84
sex	303	184	116

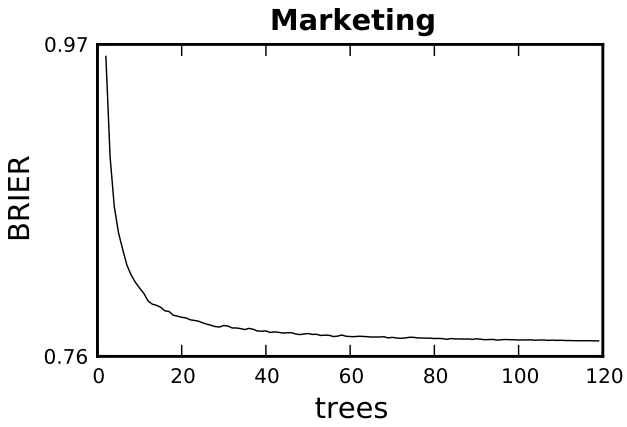
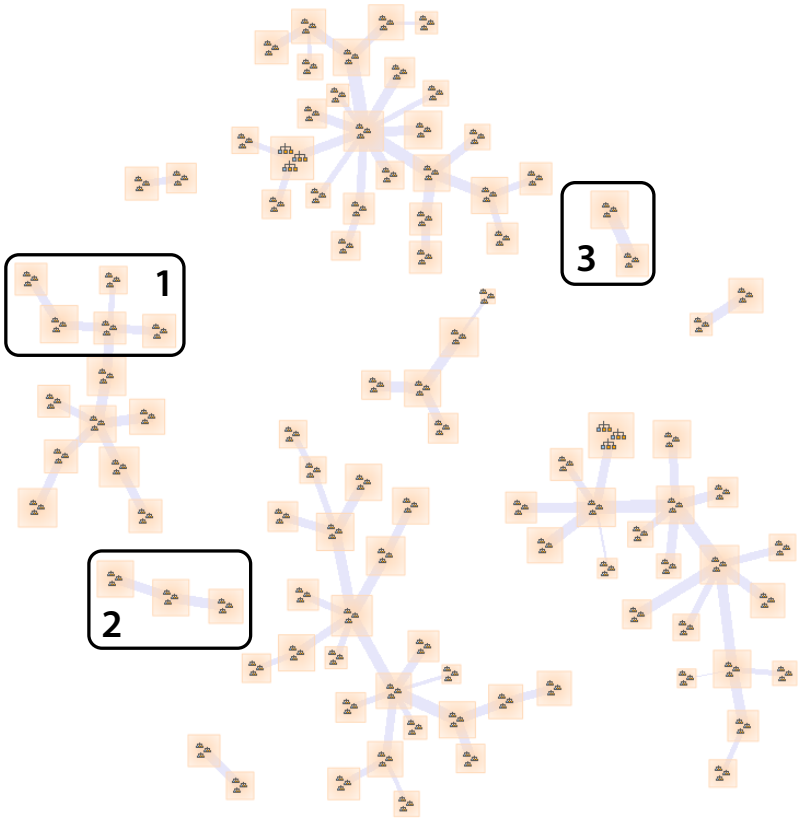
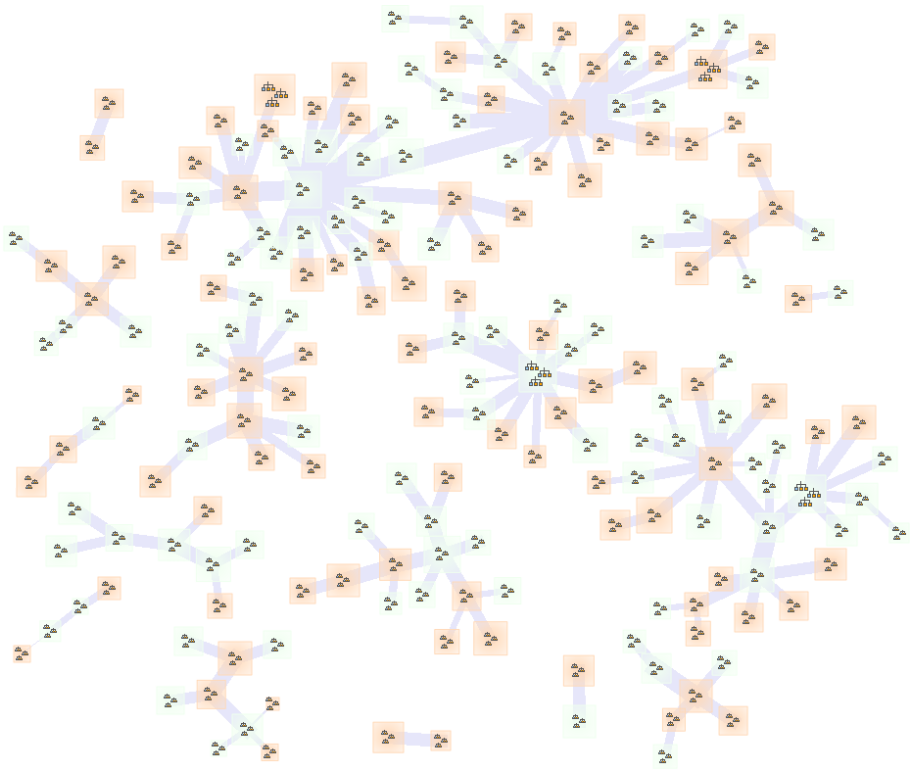


Figure 6.3: The Brier score performance measure of the random forest classifier on the Marketing data as a function of the number of trees grown.



(a) 100 trees



(b) 200 trees

Figure 6.4: Maps of random forests on the Marketing data set.

network changes when more trees are added. A map with a double number of trees is shown in Figure 6.4(b). It seems that new trees (colored light green) are added only to existing clusters. This indicates a possible heuristic for the construction of random forests: if we knew how to randomly select trees from the clusters of the whole model space (*i.e.* randomly select diverse trees), we could increase the learning speed, that is, we could have the same prediction performance with less trees, or even increase the performance, with the right blend of decision trees.

6.4 Model-map-based Forest

The purpose of this study is to explore the possibility of a heuristic for the random forest algorithm; not to propose the heuristic itself. Whether or not it is possible to guide a construction of decision trees, to resemble the random selection of trees from the *model map* clusters, is left for future work.

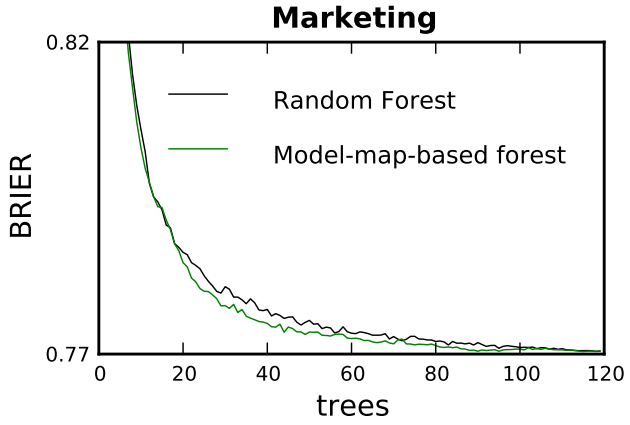
We compare the learning rate of the two forest algorithms (random and the *model-map-based* forest). For each of the increasing number of trees (from 2 to 120), we estimate the average Brier score of 500 randomly generated forests of each kind (*model-map-based* and Random Forest).

The *model-map-based* forest algorithm first clusters the trees learned with a random forest. In our case, 120 classification trees are pre-learned on the first third of the data. The second third of the data set is used to compute the model distances and cluster the trees into k clusters, where k equals the number of trees parameter of the algorithm. The k-means algorithm with the Manhattan model distance measure is used for clustering. A tree classifier is randomly chosen from each cluster to build a forest of k trees. Finally, the model is evaluated on the last third of the data with the Brier score.

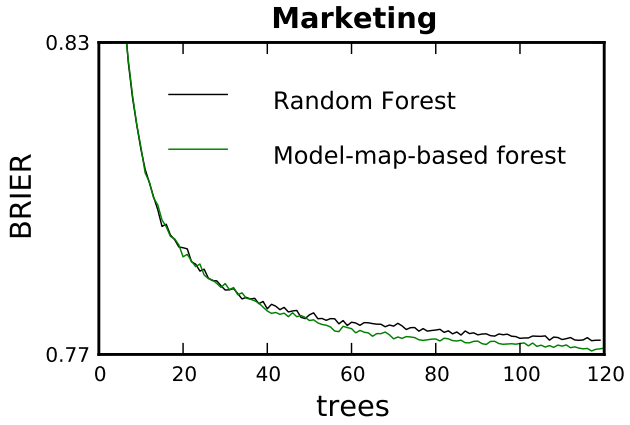
In Figure 6.5(a), the *model-map-based* forest outperforms the random forest algorithm, not in terms of the final prediction score, but in terms of the learning speed. Note that the same classification performance actually follows from the algorithm design: clustering 120 trees into 120 clusters and then "randomly" selecting one tree from each results in the same set of trees that is used by the random forest.

The results are even better if we increase the number of trees (the experiment on $1,000$ trees is shown in Figure 6.5(b)). If we knew the rules that govern the model space, it seems we could substantially increase the learning speed just by generating the "right" trees.

We test whether the learning rate of the *model-map-based* forest is significantly higher than the learning rate of the random forest. Figure 6.5(a) shows no difference of the two algorithms in the beginning (small number of trees) and in the end (large number of trees). We compare the average Brier scores on all tree sizes, to test if the difference in the middle is significant. Since the sample size is rather small



(a) 100 trees



(b) 1000 trees

Figure 6.5: The Brier score of the random forest and the *model-map-based* forest classifiers on the Marketing data as a function of the number of trees grown. The score is averaged over 500 iterations.

(119 samples) and the average Brier scores at different number of trees are not commensurate, we use the Wilcoxon signed-ranks test [116], a non-parametric version of the t-test which ranks the differences in performance of the two classifiers on each data set and compares the ranks. Small p-value (< 0.01, see Tables 6.3 and 6.4) rejects the null-hypothesis and suggests the *model-map-based* forest is significantly better than the random forest. The difference, in practice, is small and it is probably better to build a few tens of extra trees on simple domains like Marketing.

Table 6.3: Average ranks of the random and *model-map-based* forests. Methods are tested on the Marketing data.

	rank
random forest	1.746
model-map-based forest	1.254

Table 6.4: Results of the Wilcoxon signed-rank test. We compare average Brier scores, evaluated on the *model-map-based* and random forests, with different number of trees parameter, and learned on the Marketing data set.

	z	p-value
Marketing	1417.0	1.89×10^{-8}

To further explore these intriguing results, we designed a more comprehensive test setting. We compare the two algorithms on the 12 data sets introduced in Tables 4.1 and 5.1.

Results on other data sets (in Figure 6.6) are less encouraging. Our method seems to be better on two data sets (Marketing and WDBC) and competitive on four (Breast Cancer Wisconsin, Glass, Primary Tumor, and Vehicle).

The Wilcoxon signed-rank test rejected the null-hypothesis in favor of the random forest method (see Tables 6.5 and 6.6), confirmed that ensemble methods favor randomness.

Table 6.5: Average ranks of the random and model-map-based forests.

	rank
random forest	1.235
model-map-based forest	1.765

Table 6.6: Results of the Wilcoxon signed-rank test. We compare average Brier scores, evaluated on the *model-map-based* and random forests of varying number of trees parameter and learned on the 12 data set.

	z	p-value
12 data sets	137774.0	6.85×10^{-107}

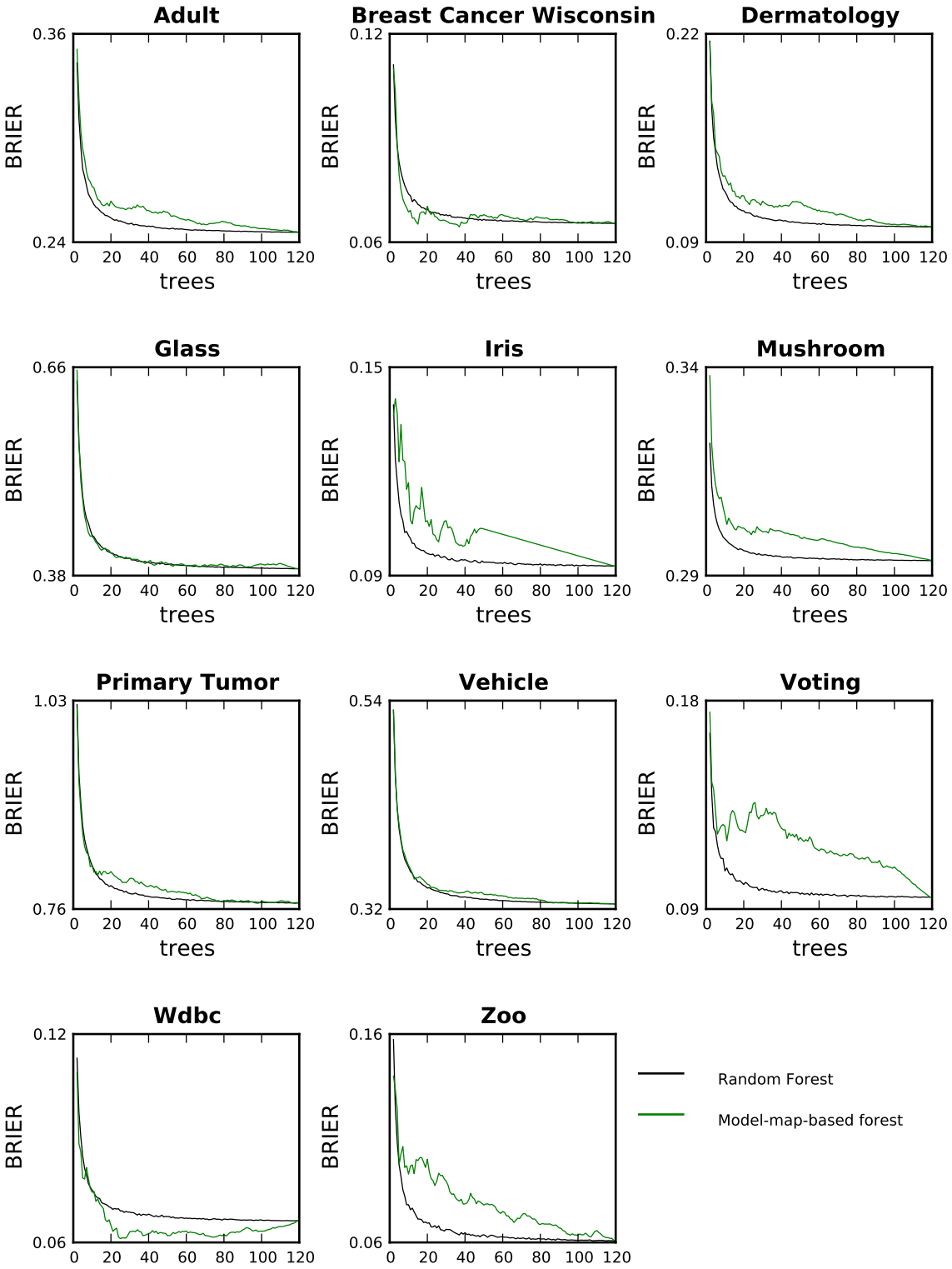


Figure 6.6: The Brier score of the random forest and the *model-map-based* forest classifiers on the 12 data sets as a function of the number of trees grown. The score is averaged over 500 iterations.

Chapter 7

Conclusion

We are witnessing the emergence of large data repositories, requiring methods to allow the domain experts to sieve through these data sets and gather information, reason on the patterns, and form plausible hypotheses to be tested in subsequent studies. Visualization combined with visual data analysis plays a major role, as it can reveal hidden patterns and allow experts to explore the data.

Visualizations require the development of dedicated algorithms that craft the proper placement of the object under consideration. Explorative data analysis requests these to be fast in order to construct responsive interfaces. We developed a layout optimization technique FragViz that meets these requirements and specifically addresses the visualization of fragmented networks, where standard algorithms do not consider similarities between unconnected components.

FragViz is neither faster than all existing algorithms nor more accurate in terms of the match between the given and the projected distances. It is slower than the Fructherman-Reingold algorithm, which is a direct consequence of considering more information. The resulting vertex distances may match the given distance matrix worse than in multidimensional scaling; a consequence of fixing the layout of the components. This is a matter of design decision: the goal of FragViz is to provide a sensible local picture and a global overview, hence the two-level optimization. For instance, in a chain-like component the two vertices on the edge may be weakly related to a common third vertex not belonging to the component. While other layout optimization algorithms would bend the chain, FragViz keeps it straight.

We responded to the lack of interactive and user friendly tools for visual network analysis with Orange Network, an add-on for Orange data mining suite. Orange Network complements existing tools with methods for local exploration of the graph, such as: highlighting subgraphs, using the network for data filtering, observing how networks evolve in time, analyzing networks with data mining algorithms, and exploring how connections relate to network data. Orange Network is embedded in a powerful analytic toolkit, giving it an edge over other network software.

The need for better predictions drives the development of complex models, ensembles of many simple models, or even algorithms for stacking them all (complex models and ensembles) into a single super-model. Methods are needed to organize and explore such super-models. We proposed a *model map* method that constructs the map of the model space. The rationale behind is that two models are not similar—from the expert perspective—if they *look* similar (*e.g.* both are trees or use the same attributes) but rather if they give similar predictions. The *model map* method defines the foundations for the comprehensive study of a mixture of models.

Model predictions are random variables (either discrete, predicted class; or continuous, predicted probability). To compare models we must thus compare the two random variables. We proposed five distance measures and found little difference between them. The idea—to compare models based on the predictions they yield—is crucial, as the *model map* method is robust regarding the distance metric. In the case studies we used the Manhattan norm model distance for its superior computational performance, and because we prefer the Euclidean-like squared differences over other measures of distribution (dis)similarity (such as the Kullback-Leibler divergence), since we draw the *model map* in the Euclidean plane.

We present two applications of the *model map* concept. The first is a technique for organizing a large number of projections: the *model map* ranking. It considers projection diversity and outperforms existing methods in the amount of information it offers. It helps an analyst to faster learn about the problem domain. Our method yields more information about the problem domain in general and also about the class than VizRank when viewing the same number of projections.

Projections can also be arranged into a map of projection space. Instead of scrolling down the list of projections, an analyst can explore an interactive network of organized projections. We suggest this research direction for the future work.

Another application is a map of the Random Forest. Existing methods focus on the indirect (*black-box-like*) visualization of the random forest. We complement them with a novel approach to visualize the forest directly. With *model map* technique we can show the relations between individual trees. We built an Orange Model Maps add-on to explore local properties of the *model map* (different regions of the map (clusters of trees) or individual trees) in an interactive way.

In addition to reviewing ensembles, our method allows the user to play an active role in the model construction. Regions of trees can be selected to form a new random forest classifier which can be sent forward to other widgets for further analysis of a specific sub-domain. Trees can be excluded or included in the forest to fine-tune the classifier by a single mouse click. We leave this open for the future work.

Orange (`http://orange.biolab.si`), Orange Network add-on, Orange Model Maps add-on, and all required third party libraries are available under the GNU GPL license for MS Windows, Linux, and Mac OS X. Source code, networks, and data sets that were used in case studies are included in Orange or in add-ons.

Appendix A

Orange and Orange Canvas

Orange is an open source Python library for machine learning and data mining. The library includes a selection of popular machine learning methods as well as many various preprocessing methods (filtering, imputation, or categorization), sampling techniques (bootstrap or cross-validation), and related methods.

Python is a popular scripting language, featuring clean syntax, a powerful yet non-obtrusive object oriented model, built-in high level data structures, complete run-time introspection, and elements of functional programming. It ships with a comprehensive library of routines for string processing, system utilities, Internet-related protocols, data compression, and many others. These can be complemented by freely available libraries like SciPy [67], which includes a library for matrix manipulation and linear algebra Numpy [4], the statistical library stats.

Similar to R, where some libraries are written in low-level languages (Fortran, C) and others in R itself, the fast core of Orange is written in C++, while other modules are written in Python.

The code below uses the Orange library from Python. It loads the Fisher's Iris data set and selects only the instances of *versicolor* and *virginica*. It randomly splits the data into training and testing subsets, constructs the logistic regression model from the training set, and counts the correctly classified instances from the test set.

```
import Orange
data = Orange.data.Table("iris")
data = data.filter(iris=["Iris-versicolor", "Iris-virginica"])
data = data.translate(Orange.data.Domain(data.domain.features,
     Orange.data.preprocess.RemoveUnusedValues(data.domain.class_var, data)))

folds = Orange.data.sample.SubsetIndices2(data, 0.7)
train = data.select(folds, 0)
test = data.select(folds, 1)
```

```
lr = Orange.classification.logreg.LogRegLearner(train)
corr = 0.0
for inst in test:
    if lr(inst) == inst.get_class():
        corr += 1
print "Accuracy:", corr / len(test)
```

To continue the example, we compared the classification accuracy (implemented in module Orange.scoring) of several algorithms (logistic regression with a stepwise variable selection, pruned classification trees, and k-nearest neighbors model) using cross-validation sampling technique from module Orange.testing.

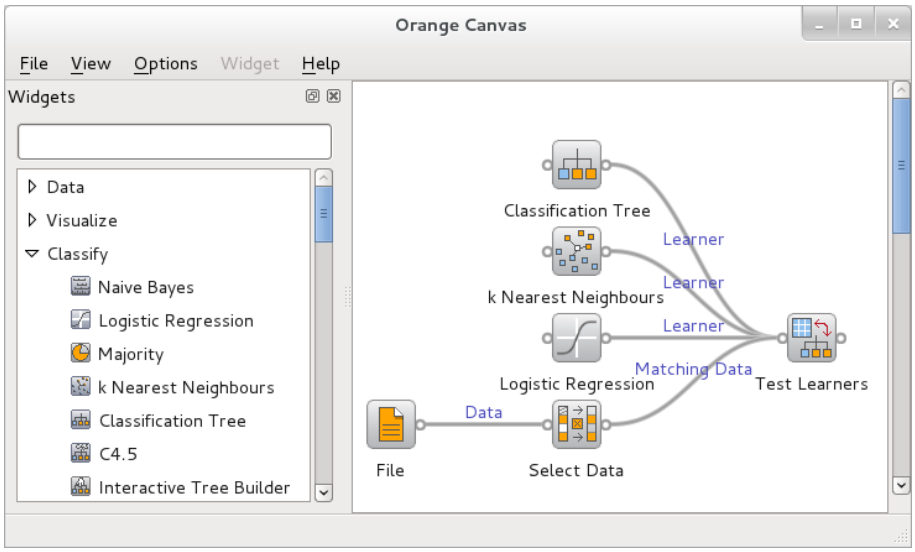
```
from Orange.classification import logreg, tree, knn
from Orange.evaluation import scoring, testing

models = [logreg.LogRegLearner(stepwiseLR = True),
          tree.TreeLearner(min_instances=2, m_pruning=2),
          knn.kNNLearner(k=3)]

res = testing.cross_validation(models, data)
cas = scoring.CA(res)
print "Logistic regression:", cas[0]
print "Classification tree:", cas[1]
print "K-nearest neighbor:", cas[2]
```

Using such scripts is only suitable for experts with enough programming skills and does not allow for visual exploration and manipulation of the data.

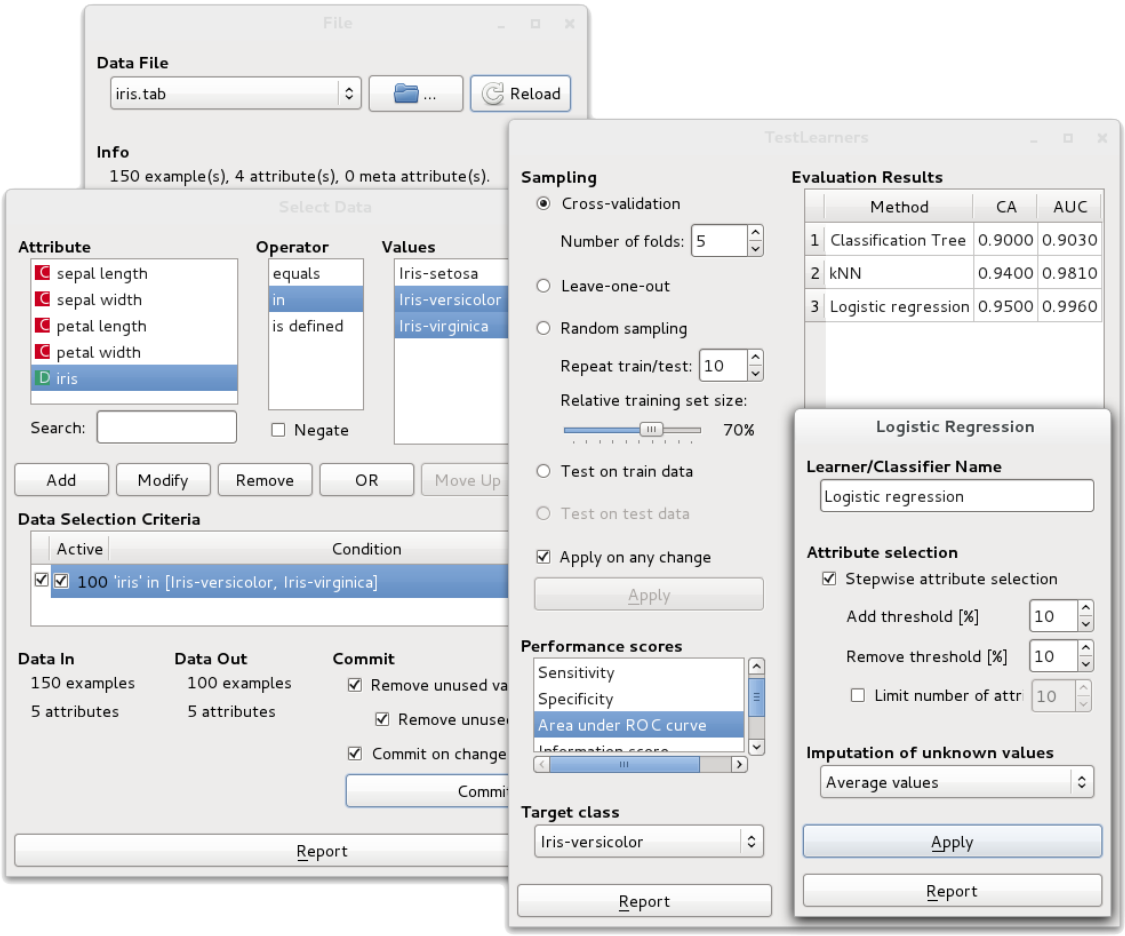
Orange Canvas provides a graphical interface for these functions. Its basic ingredients are widgets. Each widget performs a basic task, such as reading the data from a file in one of the supported formats or from a data base, showing the data in tabular form, plotting histograms and scatter plots, constructing various models and testing them, clustering the data, and so on.

Widgets can require data for performing their function and can output data as a result. The most common data type are sets of data instances; other types include models, model constructors, and many others. Types are organized into a hierarchy, so a widget may require, for instance, a general prediction model or a specific one, such as logistic regression.

The user can set set the data flow between the widgets by connecting them into a scheme: she can put the widgets onto the canvas and connect their inputs and outputs. Figure A.1(a) shows a scheme that performs the same procedure as the last code snippet. The `File` widget reads the data, `Select Data` selects instances of *versicolor* and *virginica*, and gives them to the `Test Learners`. The latter also gets learning algorithms from `Logistic Regression`, `Classification Tree`, and `K-nearest neighbors`, performs the cross validation, and shows the results.



(a) An example of the Orange Canvas scheme.



(b) A few widgets from the above scheme.

Figure A.1: Widgets exchange data in Orange Canvas: File widget loads the data which is then filtered; Test Learners widget constructs and tests three classification models.

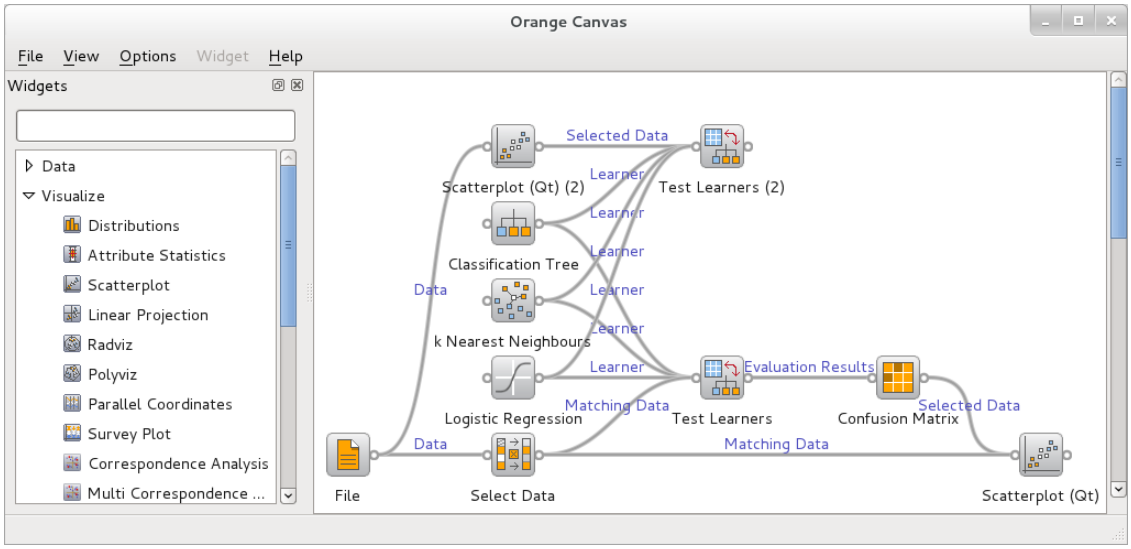


Figure A.2: An example of a scheme for interactive data exploration.

Double clicking the widget brings up a dialog with the widget's settings and, depending upon the widget type, its results. Some widgets from the scheme are shown in Figure A.1(b).

Orange Canvas is interactive: any change in a single widget—loading another data set, changing the filter, modifying the logistic regression parameters—instantly propagates down the scheme (unless the propagation is explicitly disabled, or the change needs a confirmation by the user).

The user can add a `Scatter Plot` widget to the above scheme and give it the filtered Iris data from `Select Data` (Figure A.2). Then she can connect the `Test Learners` widget to a `Confusion Matrix` that shows how many data instances from each class (corresponding to rows) were (mis)classified to another class (corresponding to columns). If the user selects one or more cells, the `Confusion Matrix` gives the corresponding instances to any widgets connected to its output.

The user can connect the `Confusion Matrix` to another of `Scatter Plot`'s inputs, the one for a subset of instances. Wired like this, the `Scatter Plot` widget shows the entire data set and marks the instances selected in the `Confusion Matrix`.

Appendix B

Distance Measure Correlation

Table B.1: The Spearman rank correlation coefficients measuring correlation between different model distance matrices of models of the Zoo and the Breast Cancer Wisconsin data sets and corresponding p-values.

		Spearman ρ				p-value			
		D4.2	D4.3	D4.4	D4.5	D4.2	D4.3	D4.4	D4.5
Breast Cancer Wisconsin	D4.1	0.941	0.918	0.918	0.790	5.7×10^{-5}	5.0×10^{-190}	5.0×10^{-190}	2.5×10^{-28}
	D4.2		0.909	0.909	0.875		1.2×10^{-69}	1.2×10^{-69}	0
	D4.3			1.000	0.851			0	1.9×10^{-117}
	D4.4				0.851				1.9×10^{-117}
Dermatology	D4.1	0.861	0.768	0.825	0.806	8.4×10^{-4}	1.5×10^{-3}	1.3×10^{-3}	8.8×10^{-4}
	D4.2		0.755	0.812	0.838		1.6×10^{-3}	1.7×10^{-3}	1.3×10^{-3}
	D4.3			0.966	0.830			2.6×10^{-4}	2.1×10^{-3}
	D4.4				0.883				7.8×10^{-4}
Iris	D4.1	0.848	0.852	0.852	0.749	3.7×10^{-5}	4.5×10^{-5}	4.3×10^{-5}	4.8×10^{-3}
	D4.2		0.807	0.810	0.767		1.8×10^{-4}	1.7×10^{-4}	6.3×10^{-4}
	D4.3			1.000	0.917			1.6×10^{-26}	1.7×10^{-5}
	D4.4				0.918				1.7×10^{-5}
Voting	D4.1	0.815	0.838	0.838	0.820	8.2×10^{-22}	7.3×10^{-6}	7.3×10^{-6}	2.8×10^{-39}
	D4.2		0.860	0.860	0.828		1.9×10^{-5}	1.9×10^{-5}	1.0×10^{-80}
	D4.3			1.000	0.856			0	3.1×10^{-4}
	D4.4				0.856				3.1×10^{-4}
Zoo	D4.1	0.867	0.893	0.907	0.826	4.6×10^{-9}	5.7×10^{-4}	9.0×10^{-5}	5.9×10^{-6}
	D4.2		0.874	0.887	0.841		4.0×10^{-4}	1.7×10^{-8}	8.9×10^{-13}
	D4.3			0.989	0.849			1.8×10^{-4}	5.3×10^{-4}
	D4.4				0.872				8.1×10^{-4}

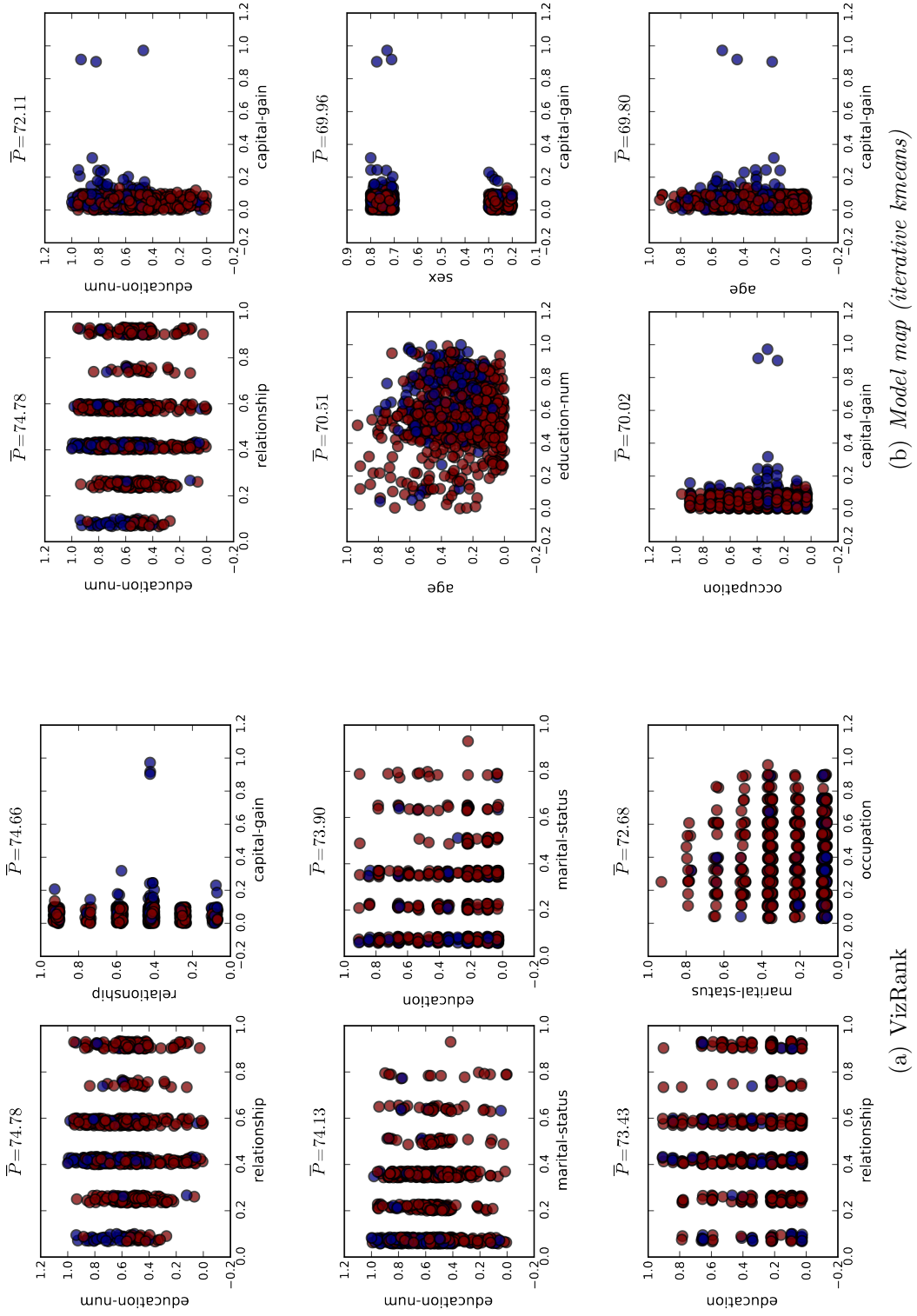
Appendix C

Projection Sets

C.1 Scatter plot

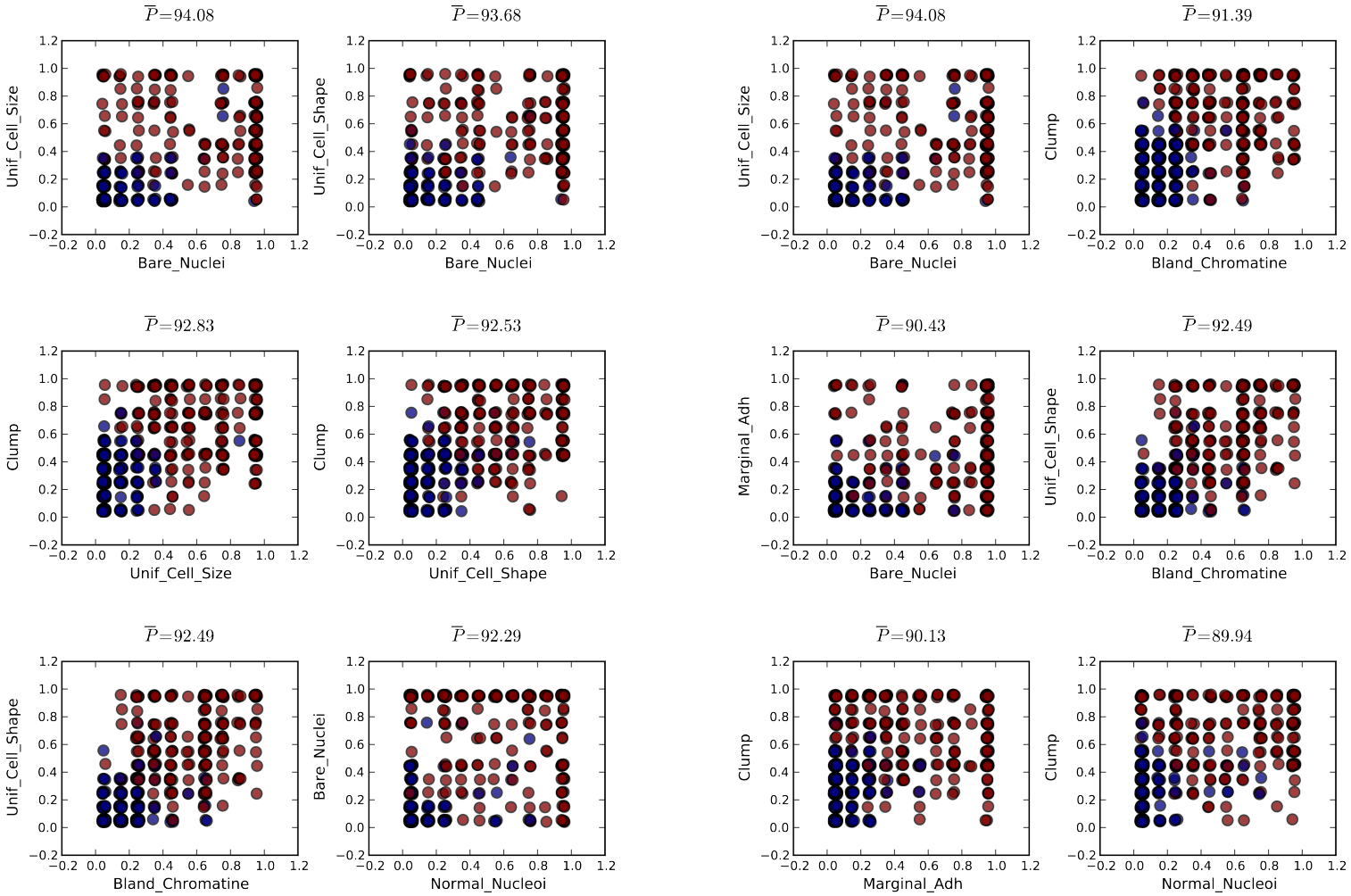
B.2 Radviz on Three Attributes

B.3 Radviz on Four Attributes



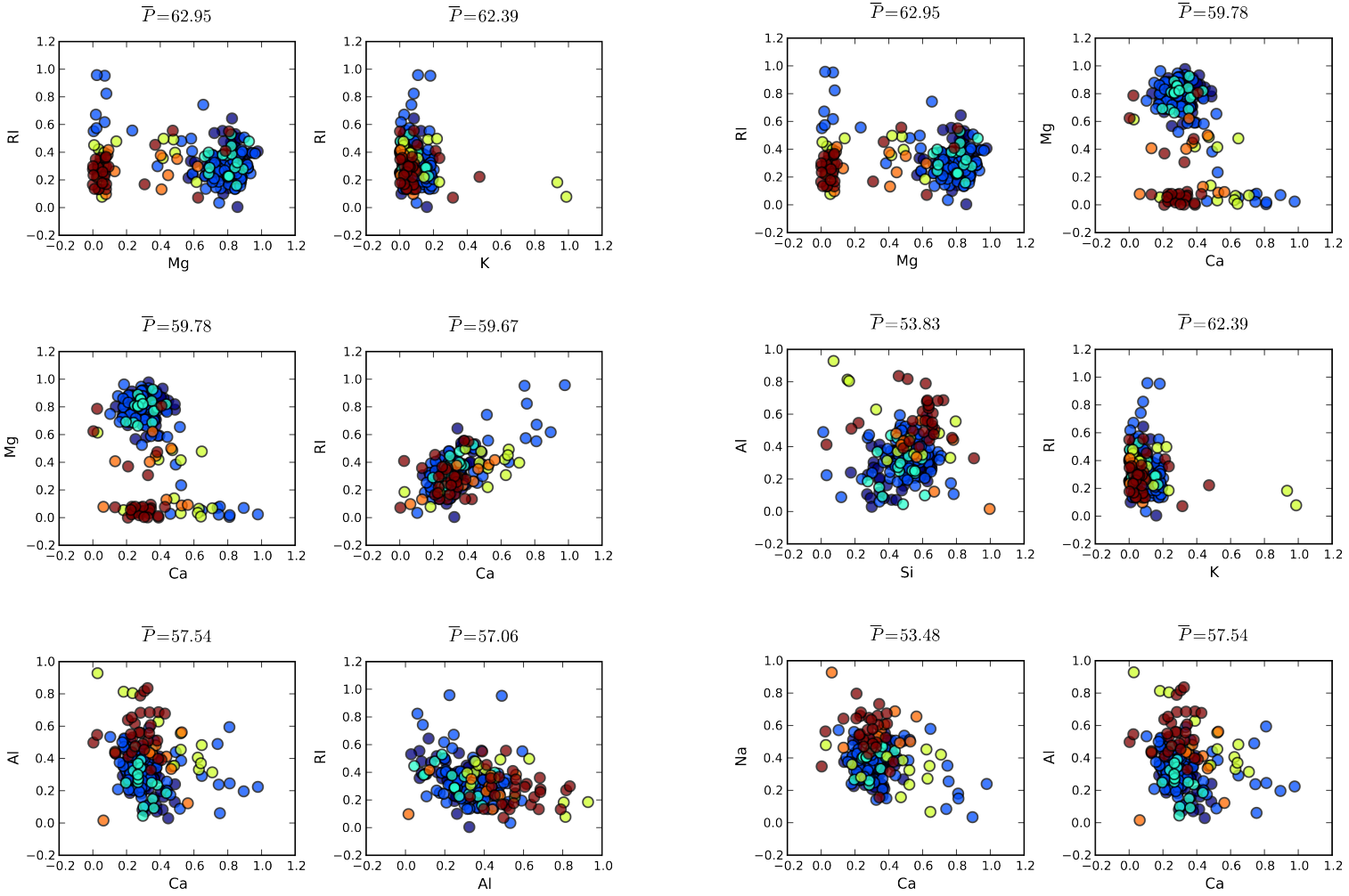
(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.1: The best six scatter plot projections of the Adult data set.
Node colors: brown ($<= 50K$) and violet ($> 50K$).



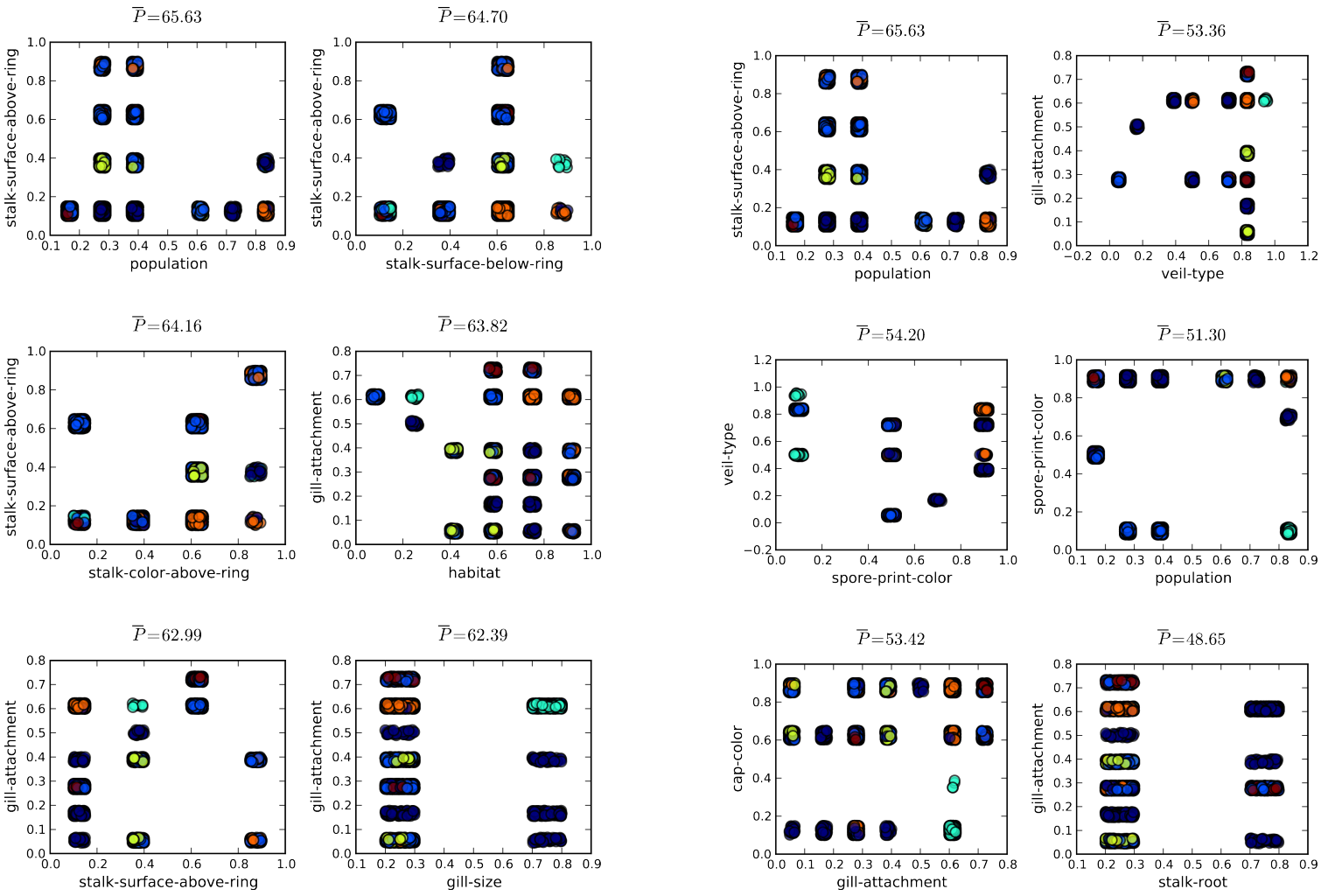
(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.2: The best six scatter plot projections of the Breast Cancer Wisconsin data set.
Node colors: brown (*malign*) and violet (*benign*).



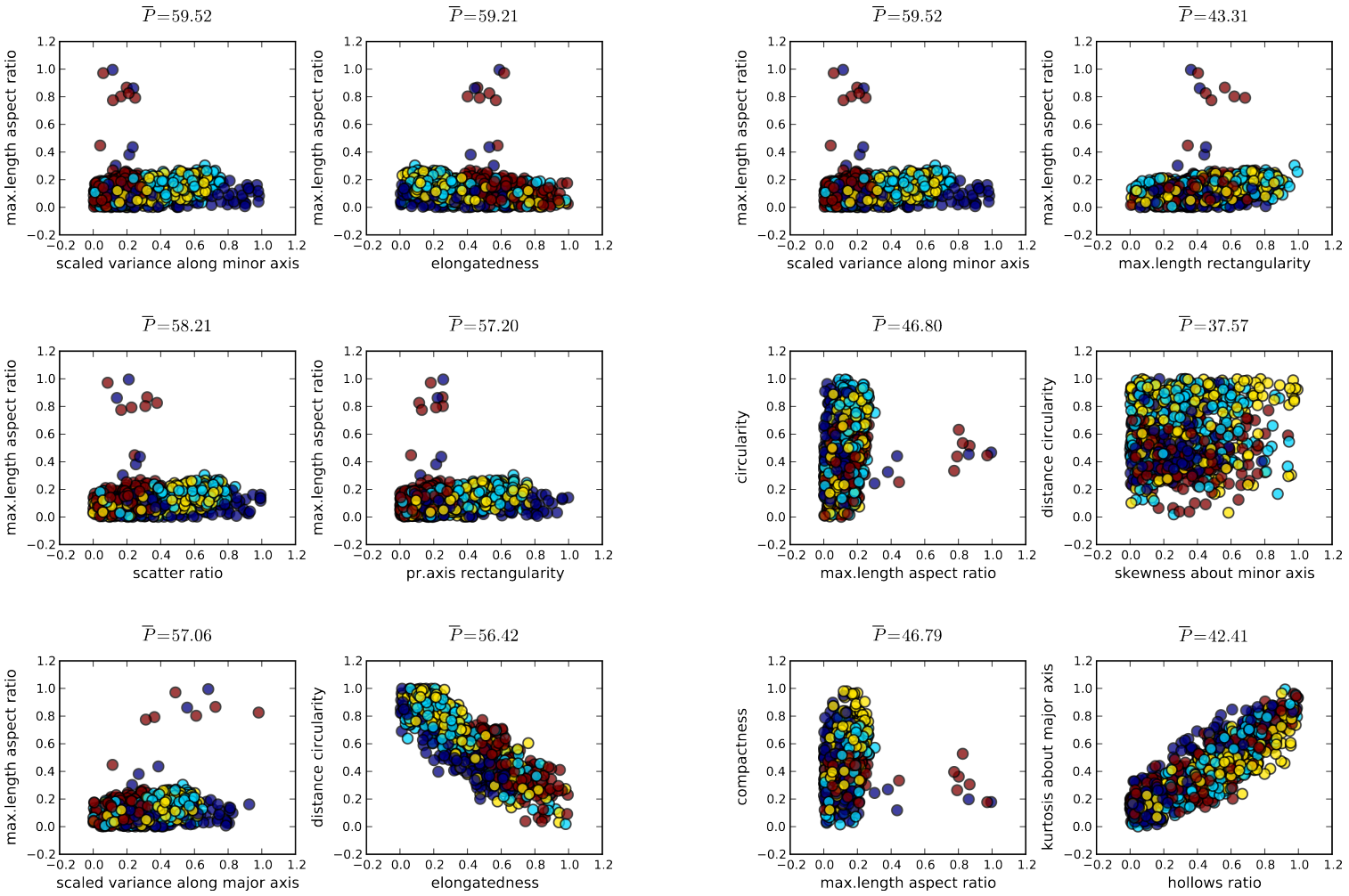
(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.3: The best six scatter plot projections of the Glass data set.
Node colors: violet (*1*), blue (*2*), light blue (*3*), green (*5*), orange (*6*), and brown (*7*).



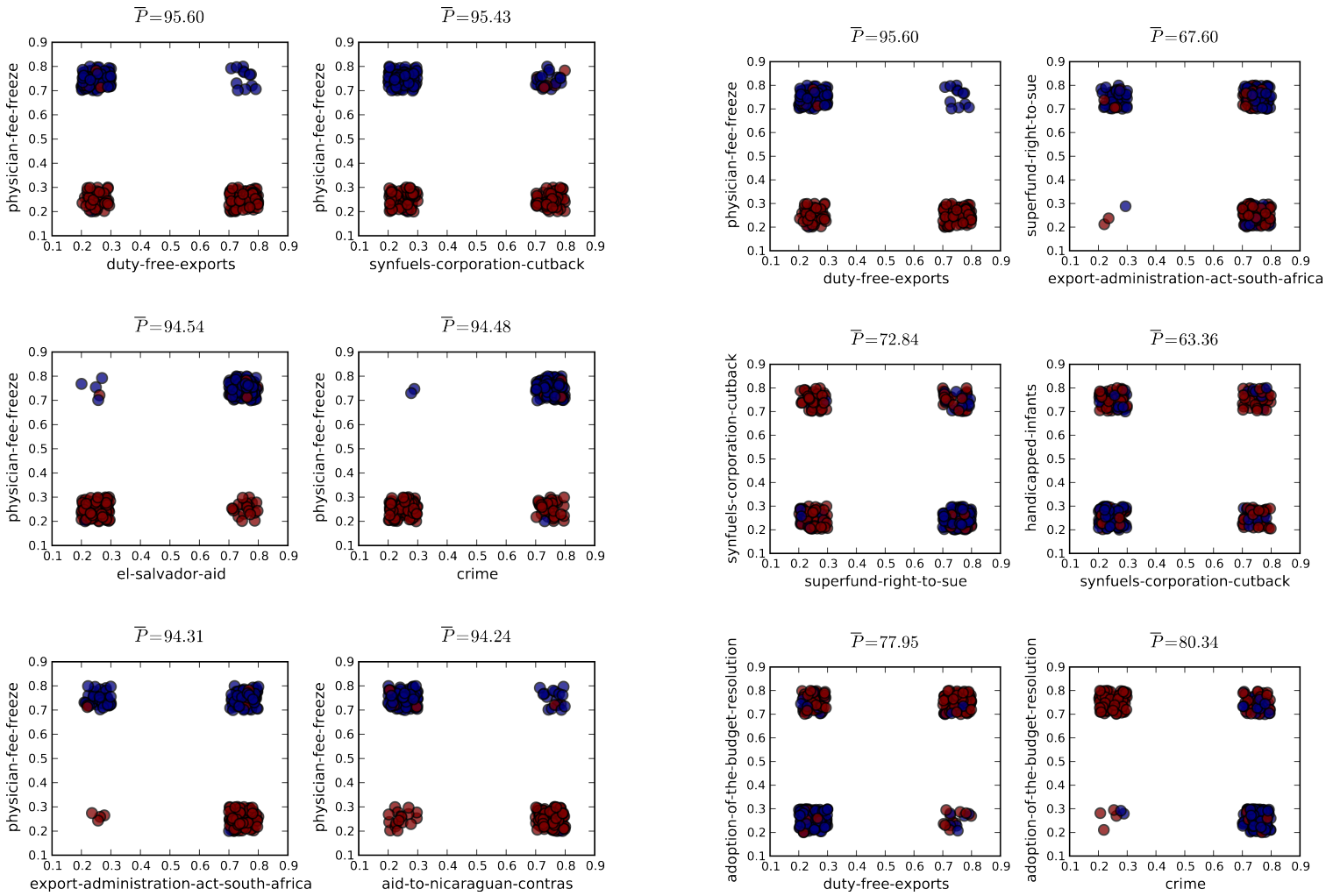
(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.4: The best six scatter plot projections of the Mushroom data set.
Node colors: violet (d), blue (g), light blue (l), orange (p), brown (u), green (m), and red (w).



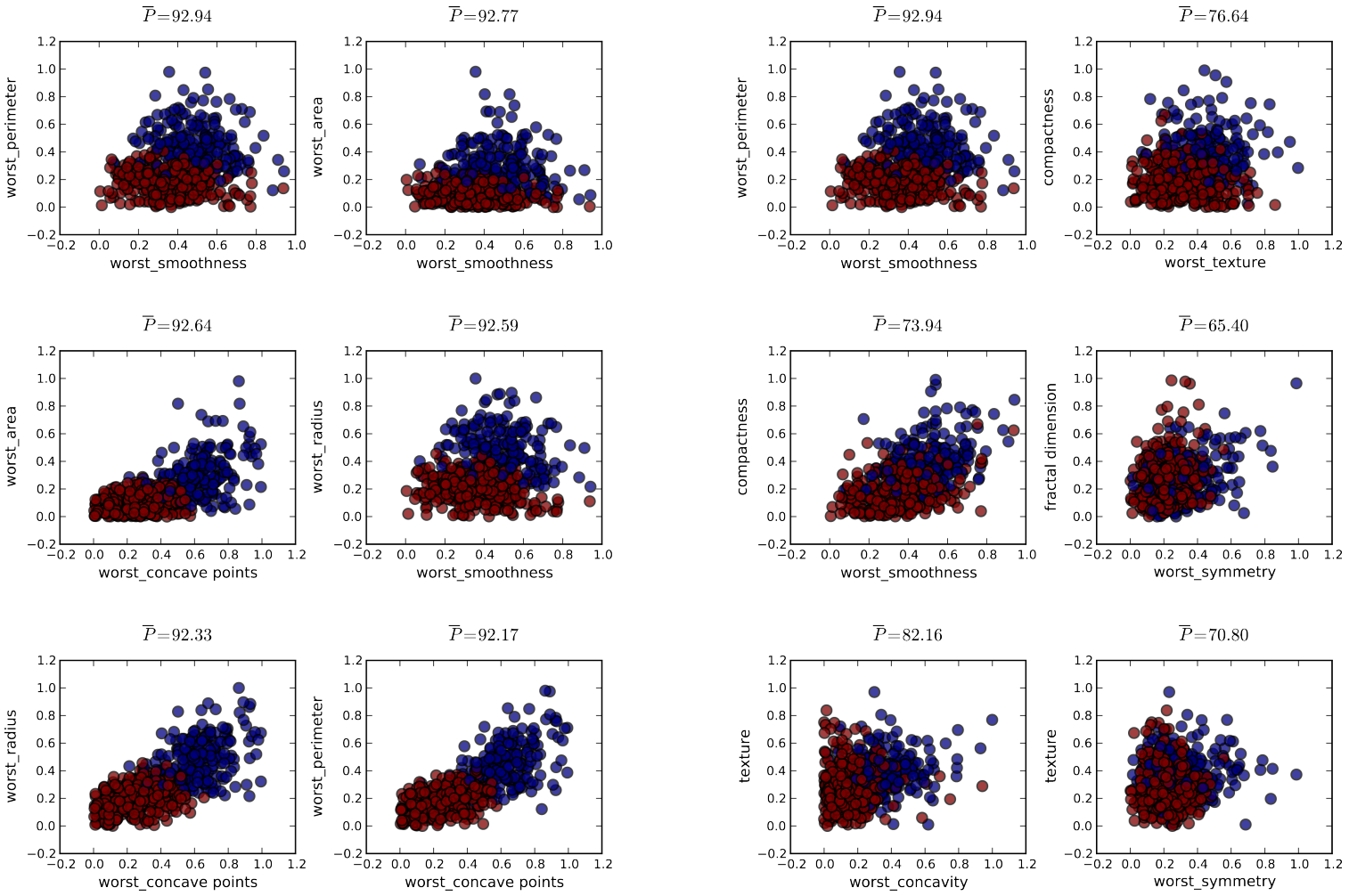
(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.5: The best six scatter plot projections of the Vehicle data set.
Node colors: violet (*bus*), brown (*van*), light blue (*opel*), and yellow (*saab*).



(a) VizRank (b) *Model map (iterative kmeans)*

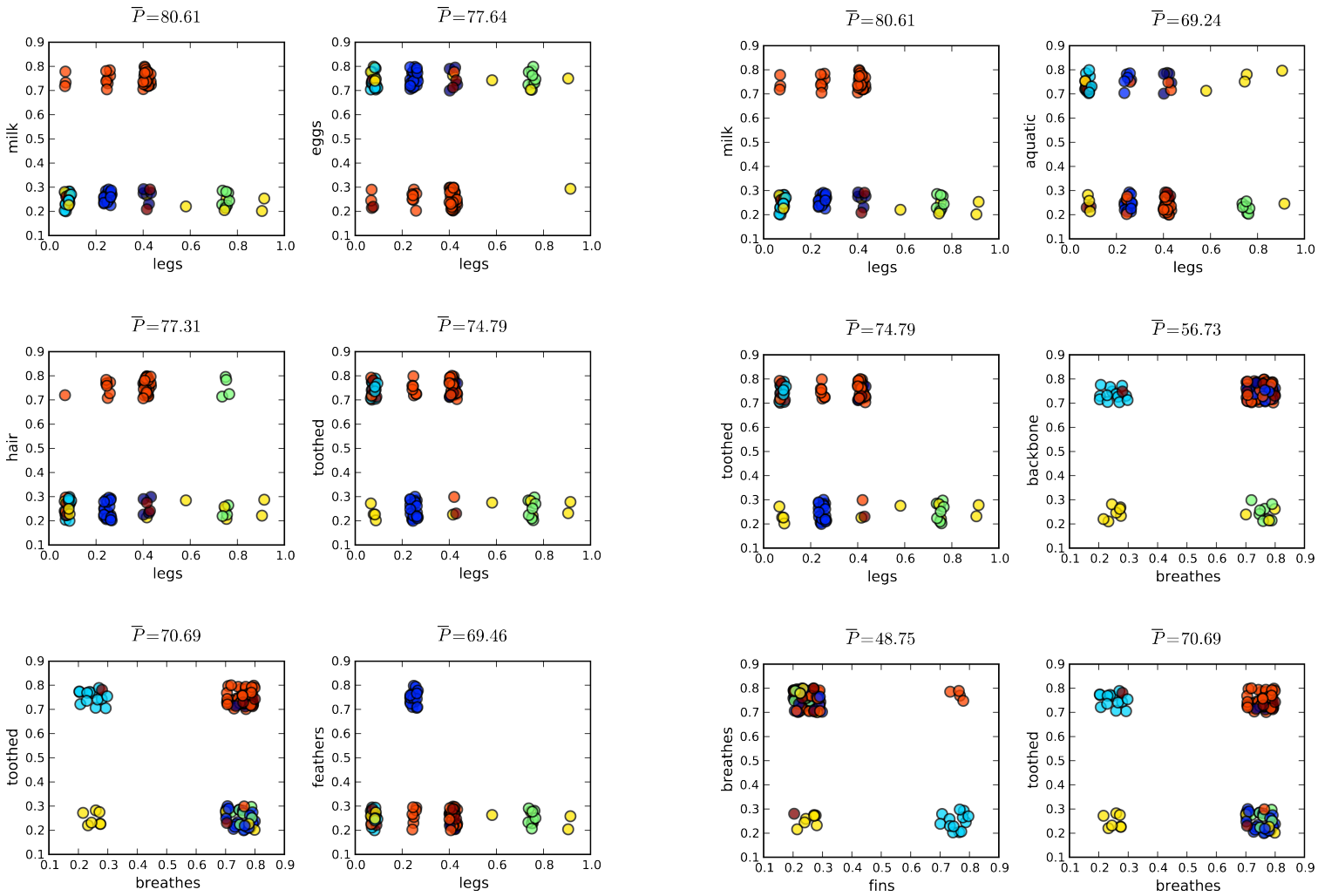
Figure C.6: The best six scatter plot projections of the Voting data set.
Node colors: brown (*republican*) and violet (*democrat*).



(a) VizRank

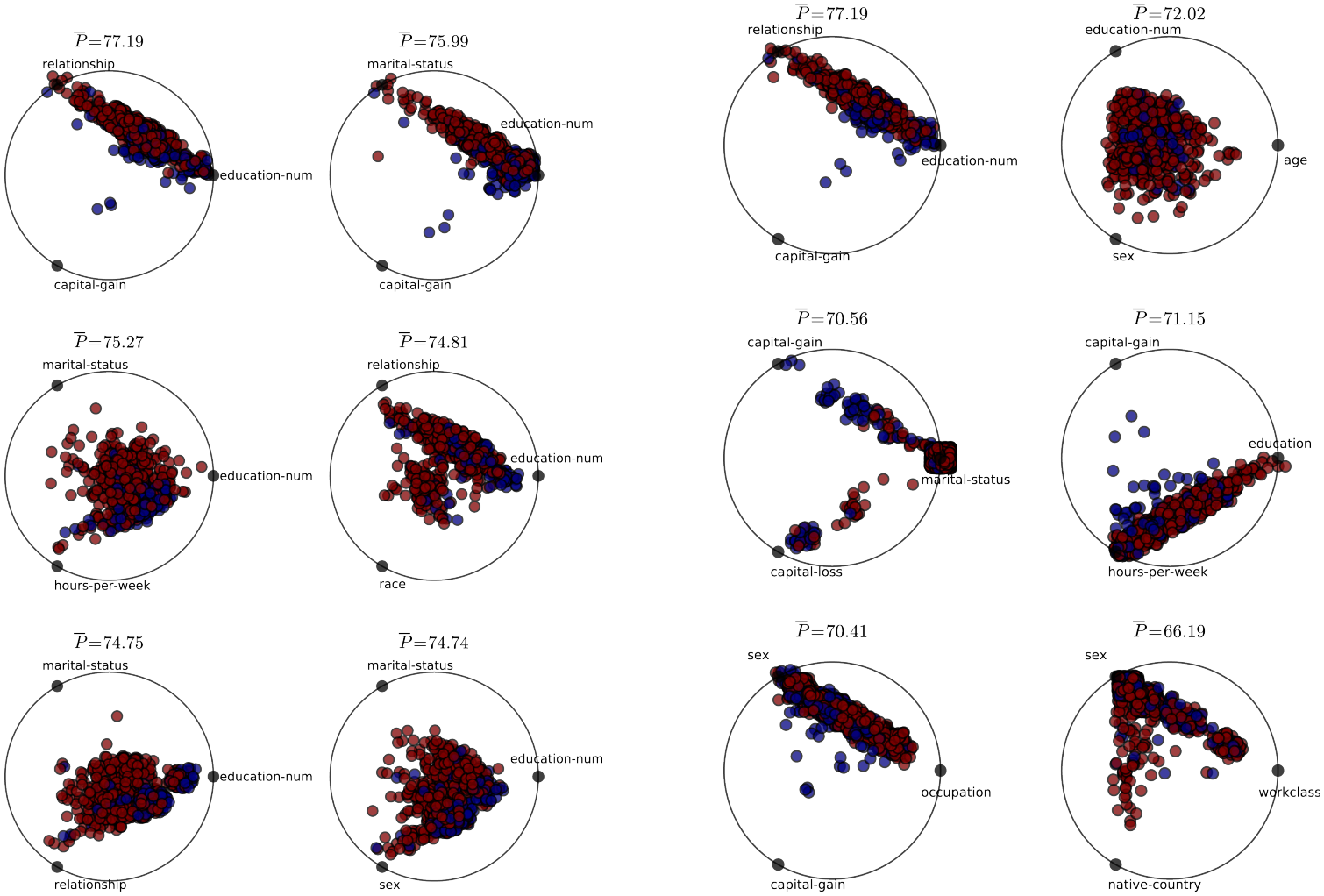
(b) *Model map (iterative kmeans)*

Figure C.7: The best six scatter plot projections of the WDBC data set.
Node colors: brown (G) and violet (M).



(a) VizRank (b) *Model map (iterative kmeans)*

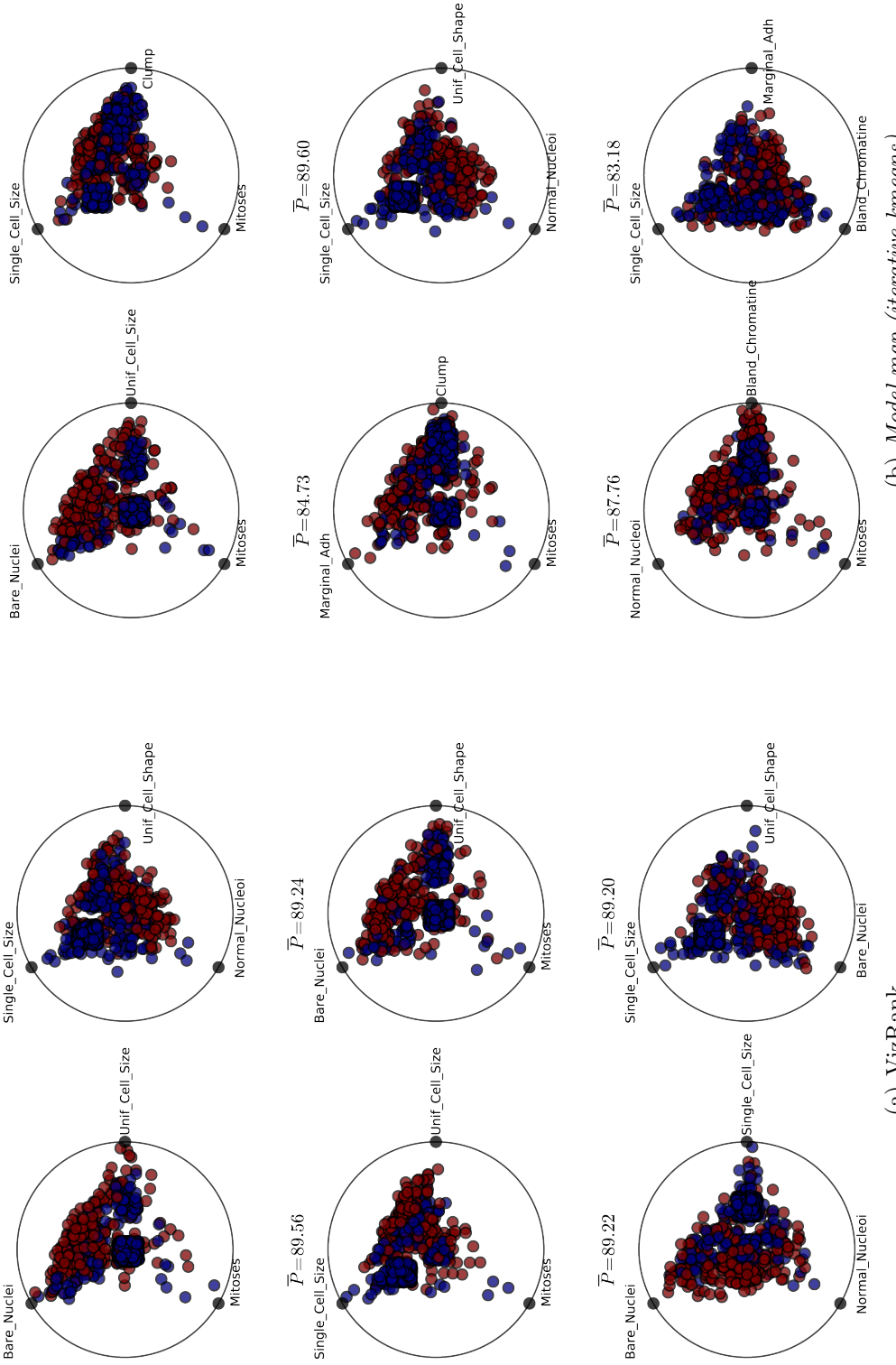
Figure C.8: The best six scatter plot projections of the Zoo data set.
Node colors: violet (*amphibian*), brown (*reptile*), blue (*bird*), light blue (*insect*), green (*insect*), orange (*mammal*), and yellow (*invertebrate*).



(a) VizRank

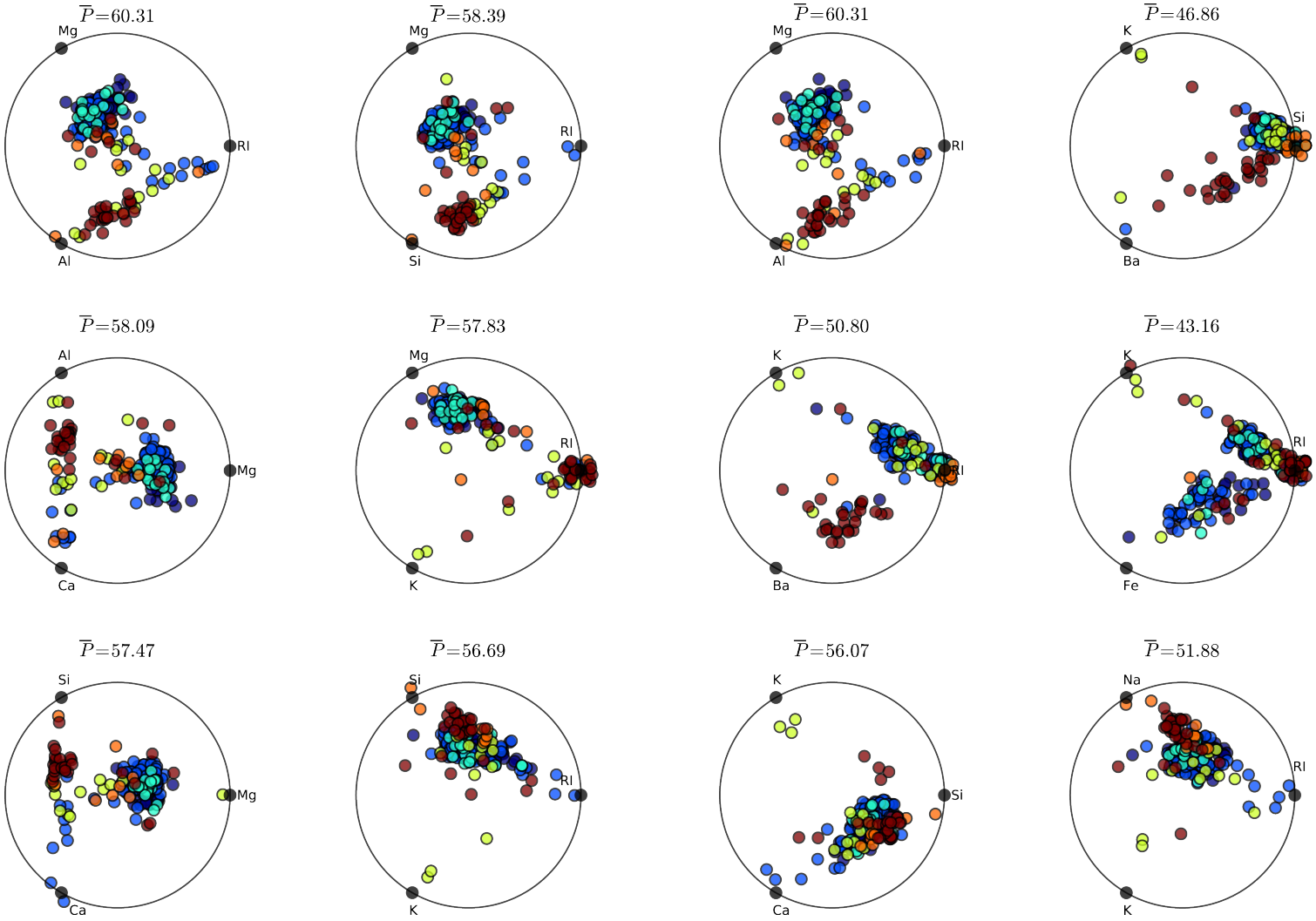
(b) *Model map (iterative kmeans)*

Figure C.9: The best six Radviz projections (3 attributes) of the Adult data set.
Node colors: brown ($<= 50K$) and violet ($> 50K$).



(a) VizRank (b) *Model map (iterative kmeans)*

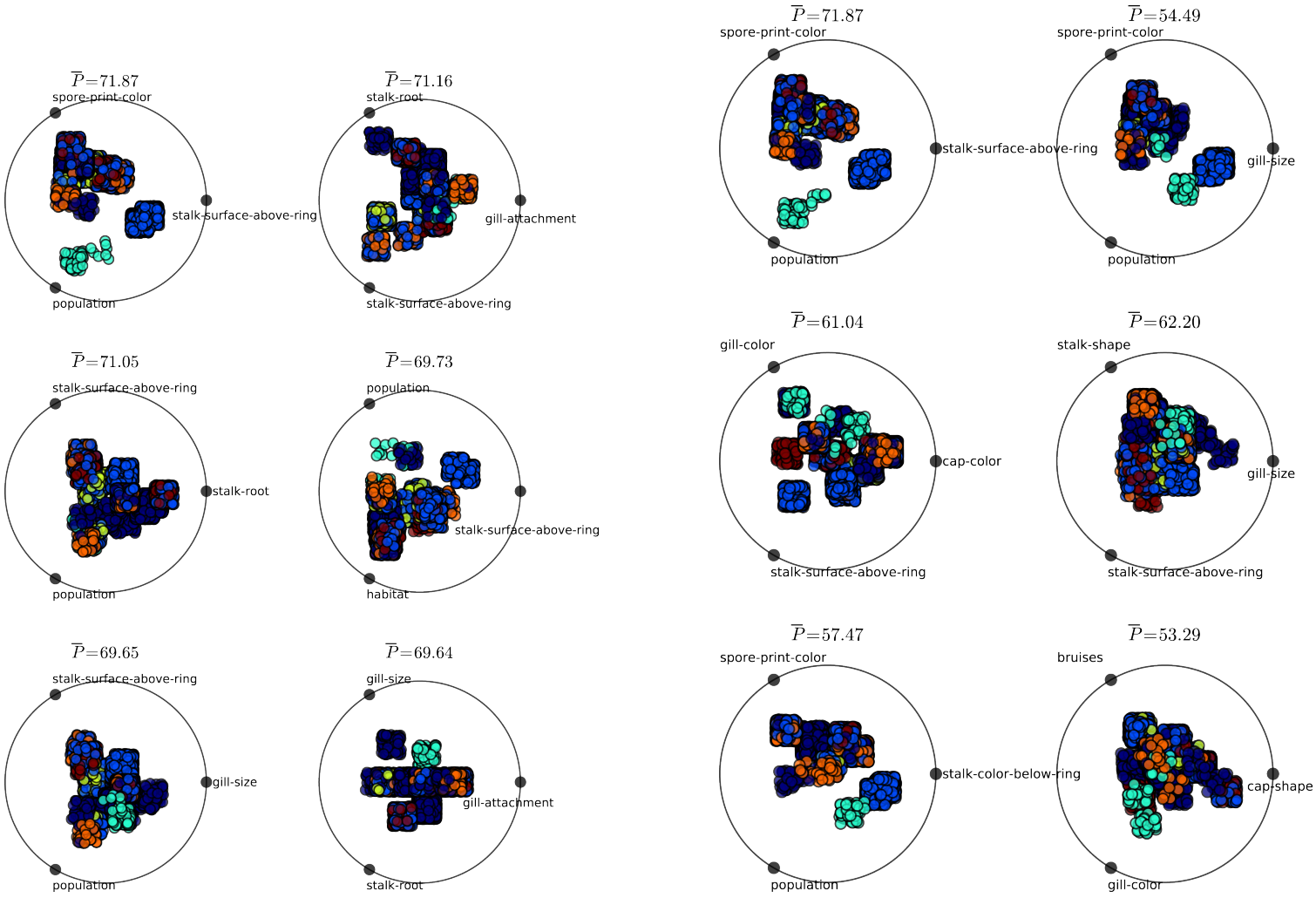
Figure C.10: The best six Radviz projections (3 attributes) of the Breast Cancer Wisconsin data set.
Node colors: brown (*malign*) and violet (*benign*).



(a) VizRank

(b) *Model map (iterative kmeans)*

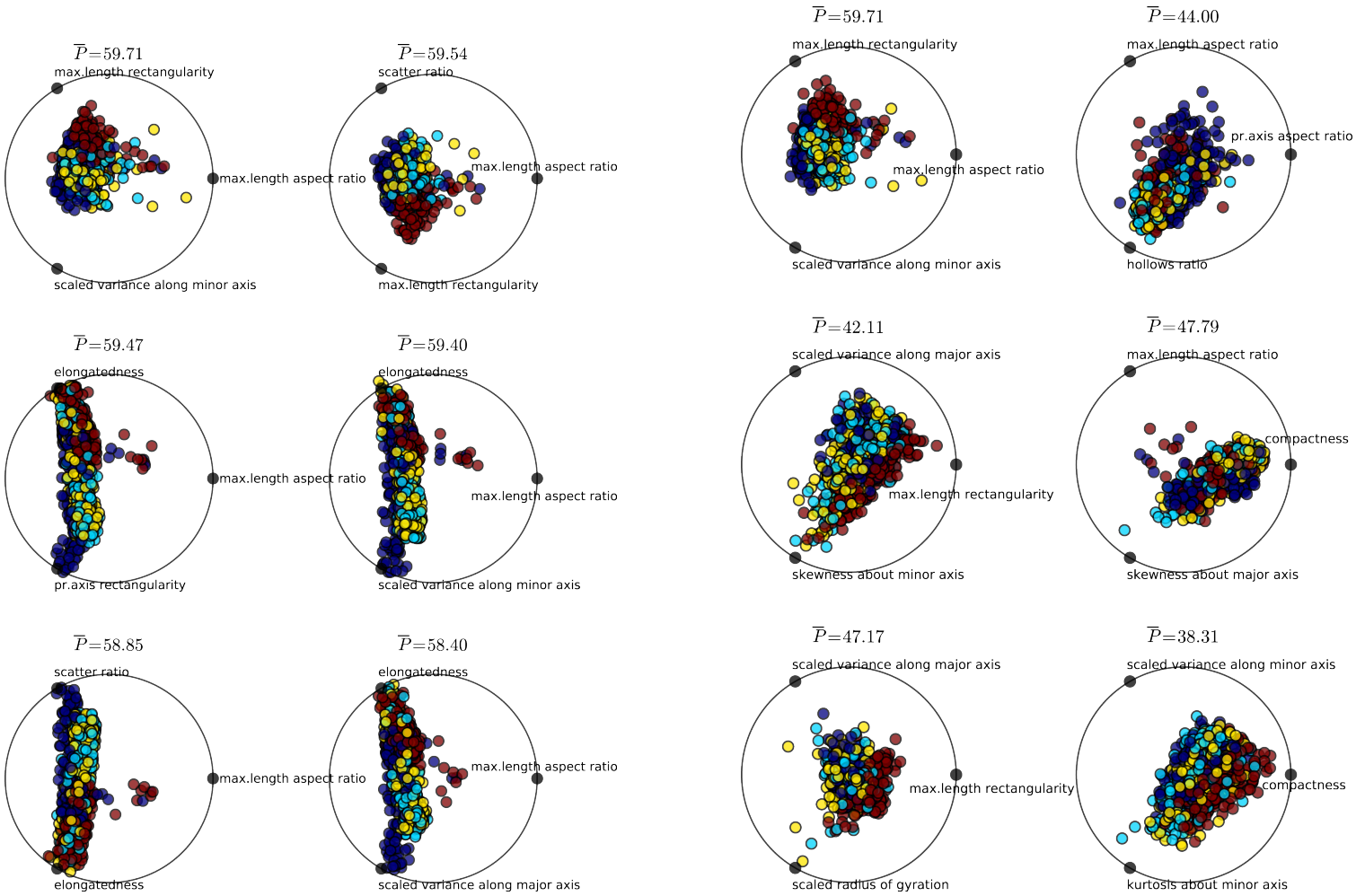
Figure C.11: The best six Radviz projections (3 attributes) of the Glass data set.
Node colors: violet (*1*), blue (*2*), light blue (*3*), green (*5*), orange (*6*), and brown (*7*).



(a) VizRank

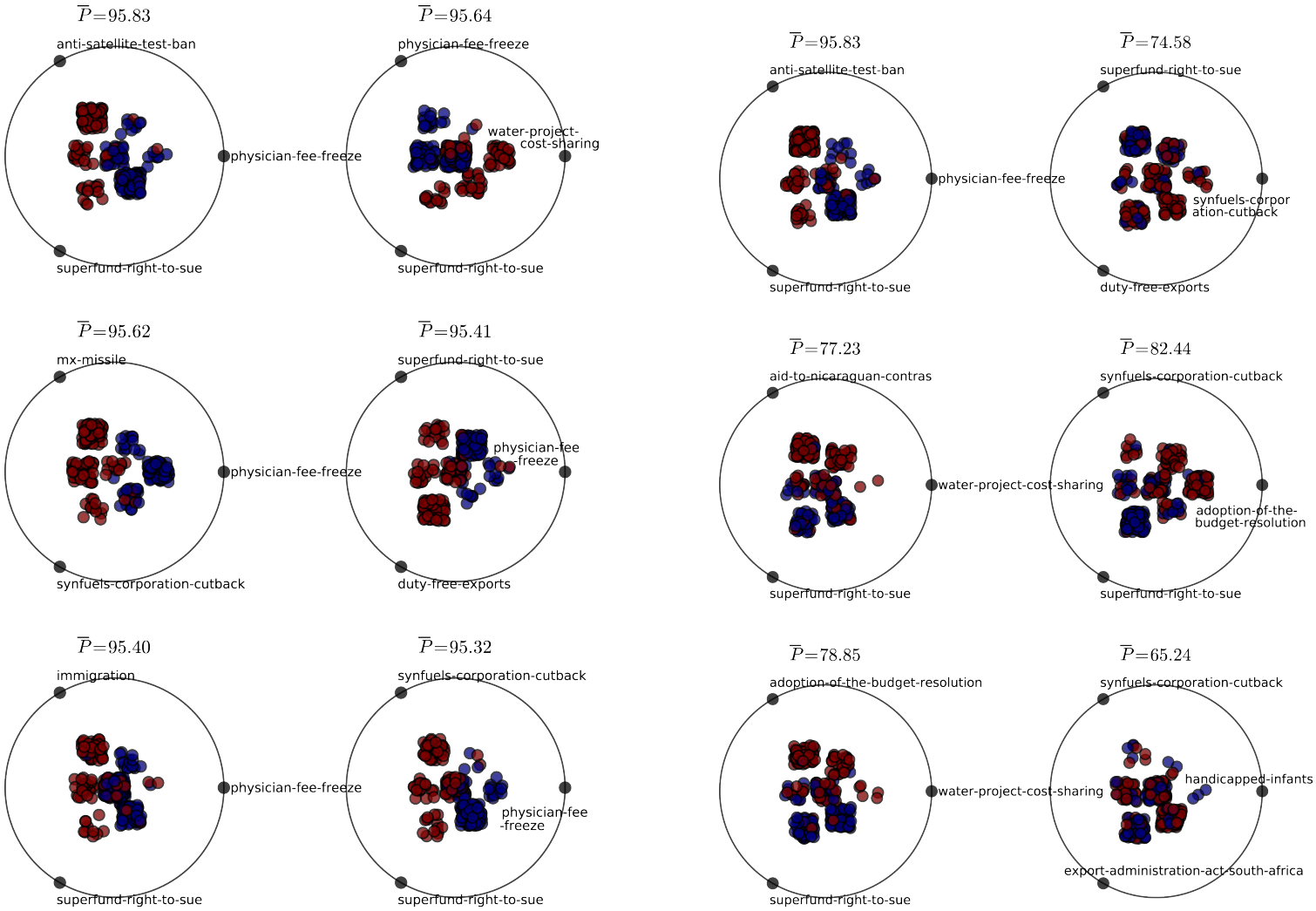
(b) *Model map (iterative kmeans)*

Figure C.12: The best six Radviz projections (3 attributes) of the Mushroom data set.
Node colors: violet (d), blue (g), light blue (l), orange (p), brown (u), green (m), and red (w).



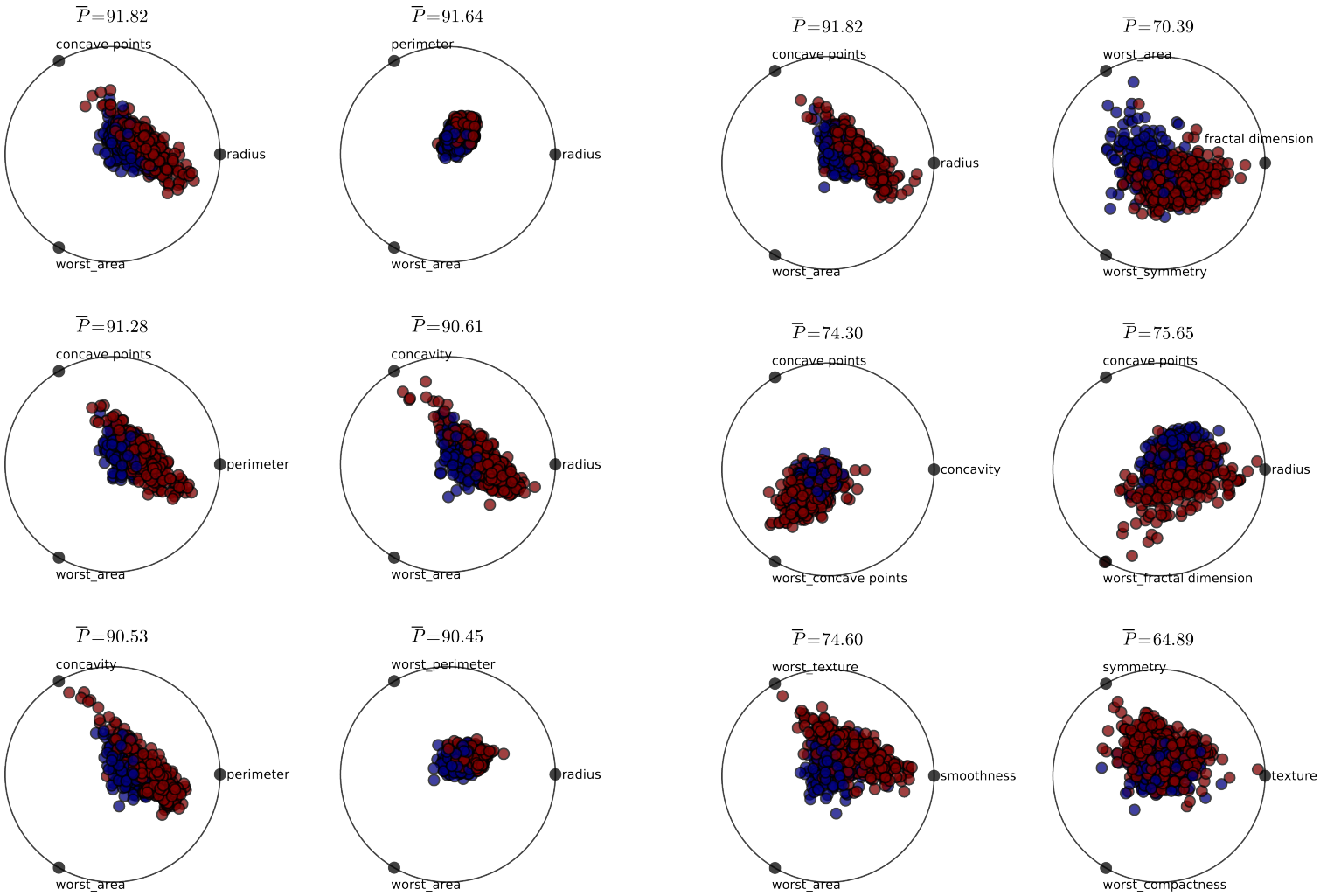
(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.13: The best six Radviz projections (3 attributes) of the Vehicle data set.
Node colors: violet (*bus*), brown (*van*), light blue (*opel*), and yellow (*saab*).



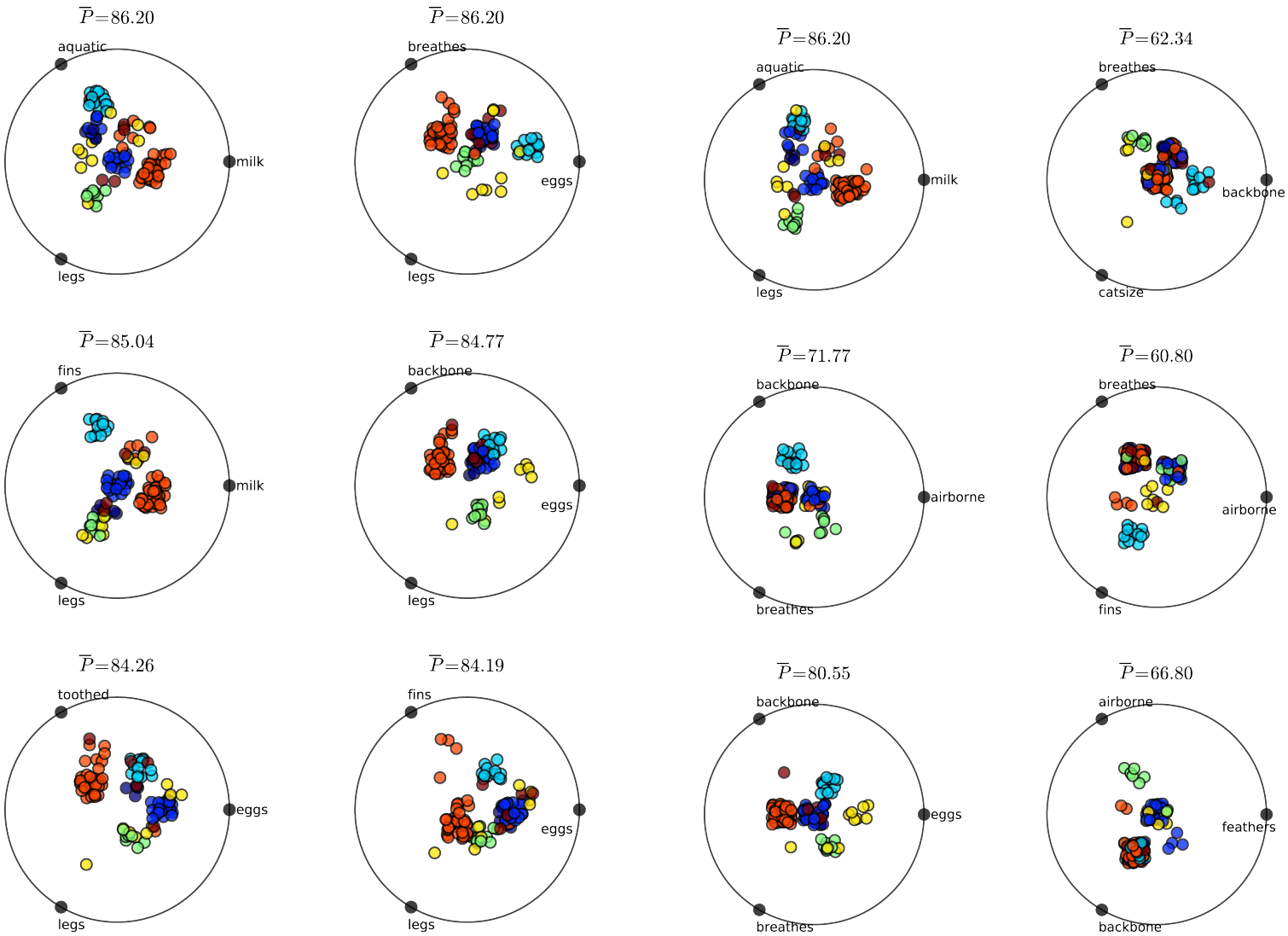
(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.14: The best six Radviz projections (3 attributes) of the Voting data set.
Node colors: brown (*republican*) and violet (*democrat*).



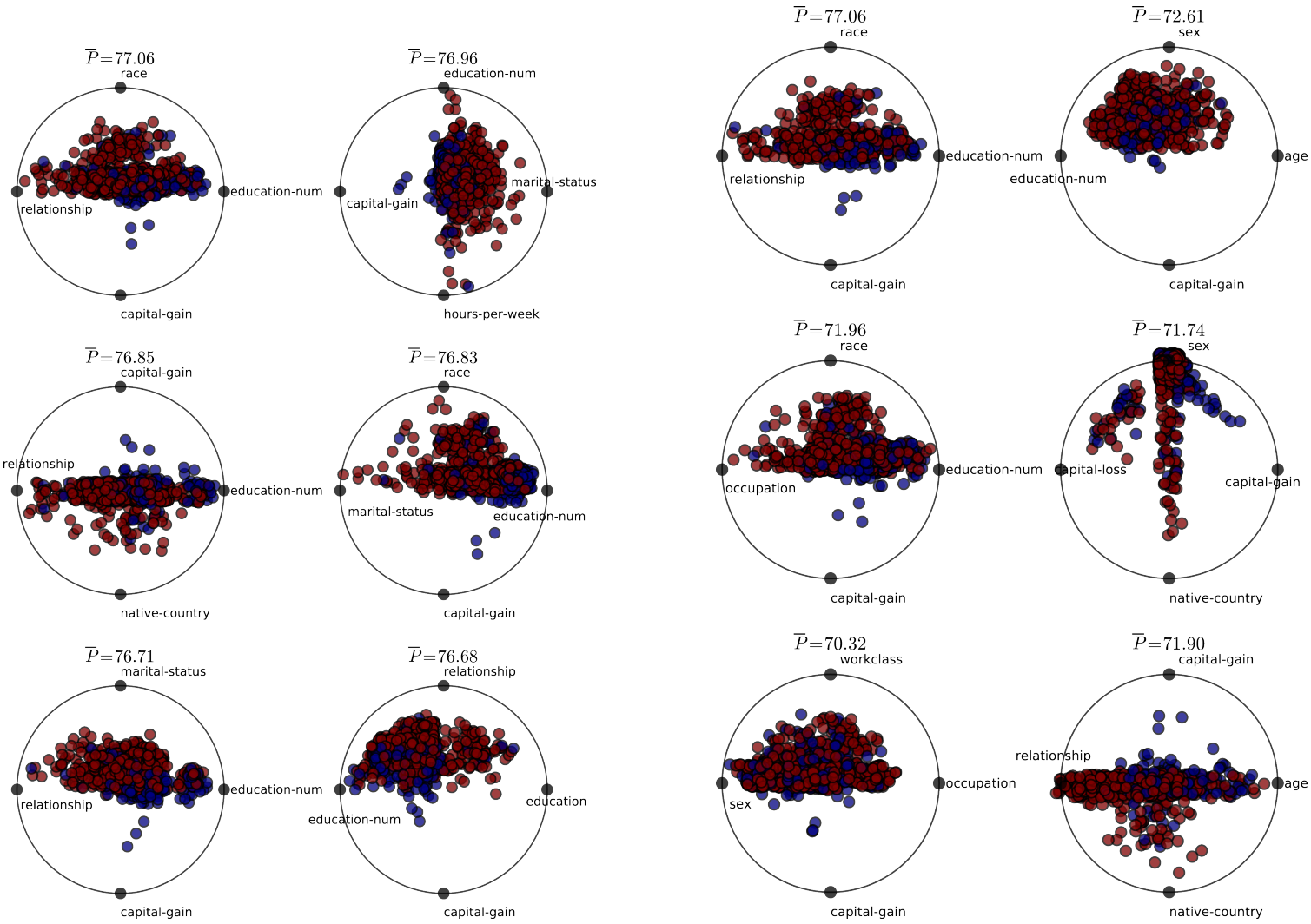
(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.15: The best six Radviz projections (3 attributes) of the WDBC data set.
Node colors: brown (G) and violet (M).



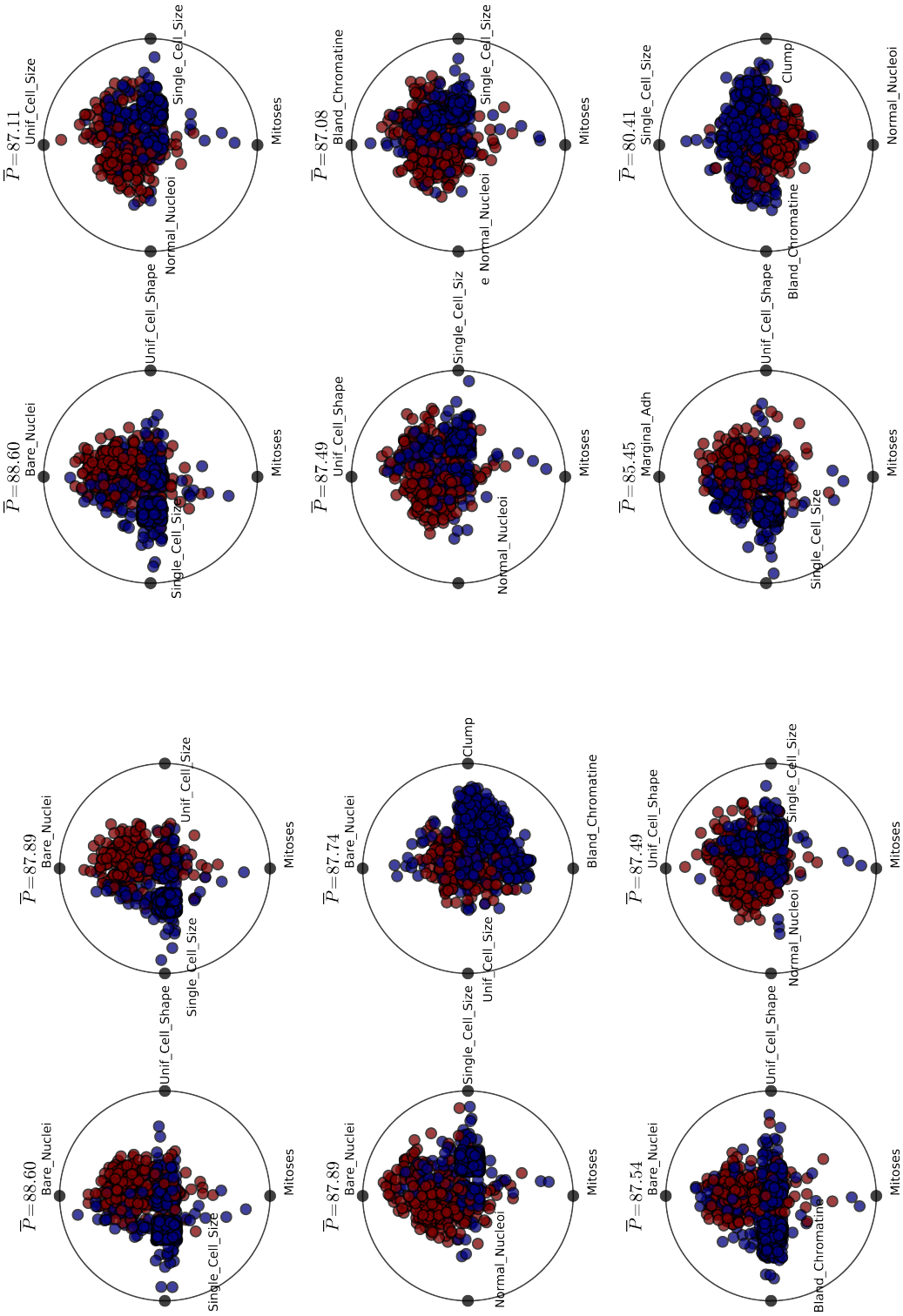
(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.16: The best six Radviz projections (3 attributes) of the Zoo data set.
Node colors: violet (*amphibian*), brown (*reptile*), blue (*bird*), light blue (*insect*), green (*insect*), orange (*mammal*), and yellow (*invertebrate*).



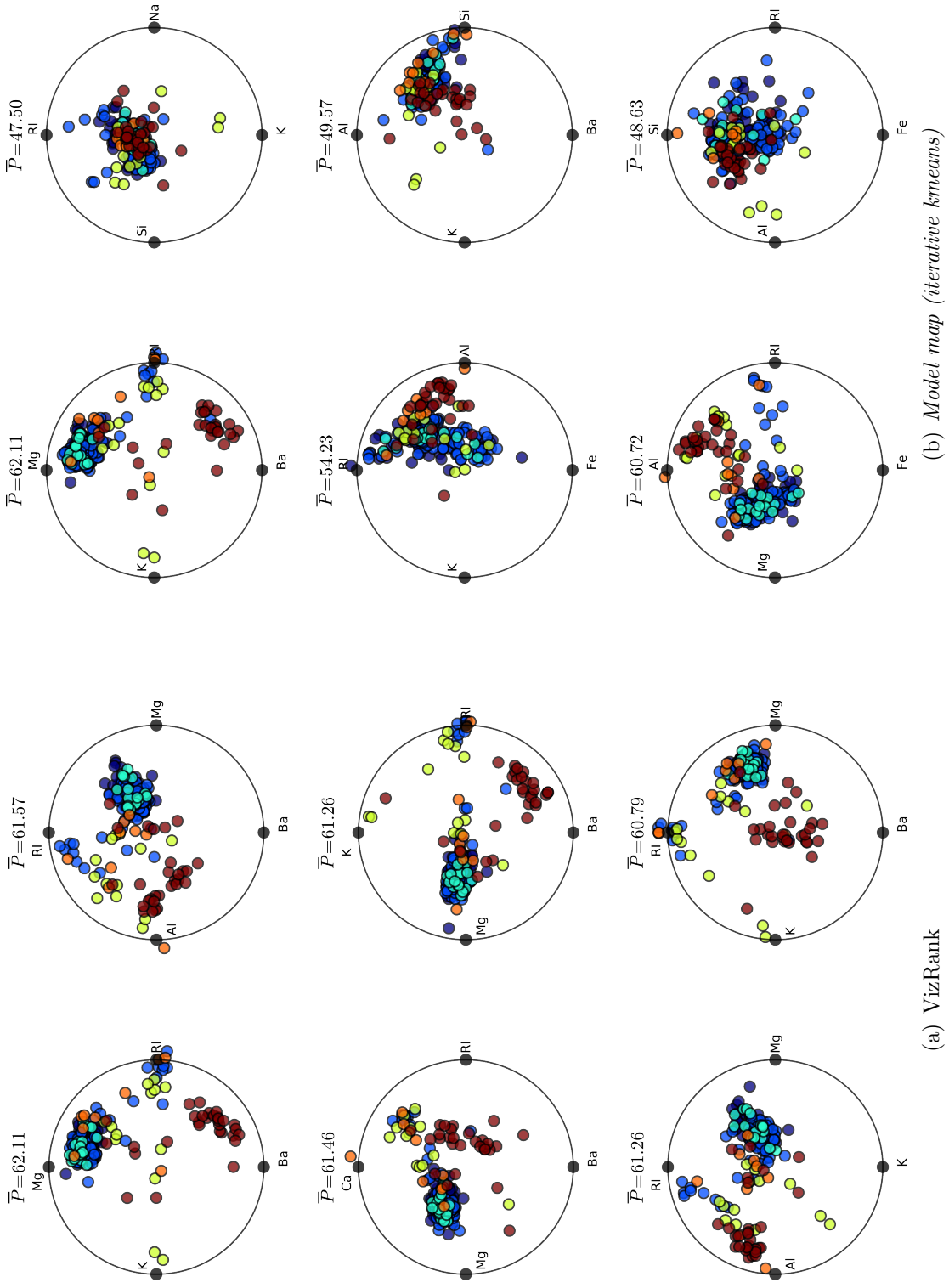
(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.17: The best six Radviz projections (4 attributes) of the Adult data set.
Node colors: brown ($<= 50K$) and violet ($> 50K$).



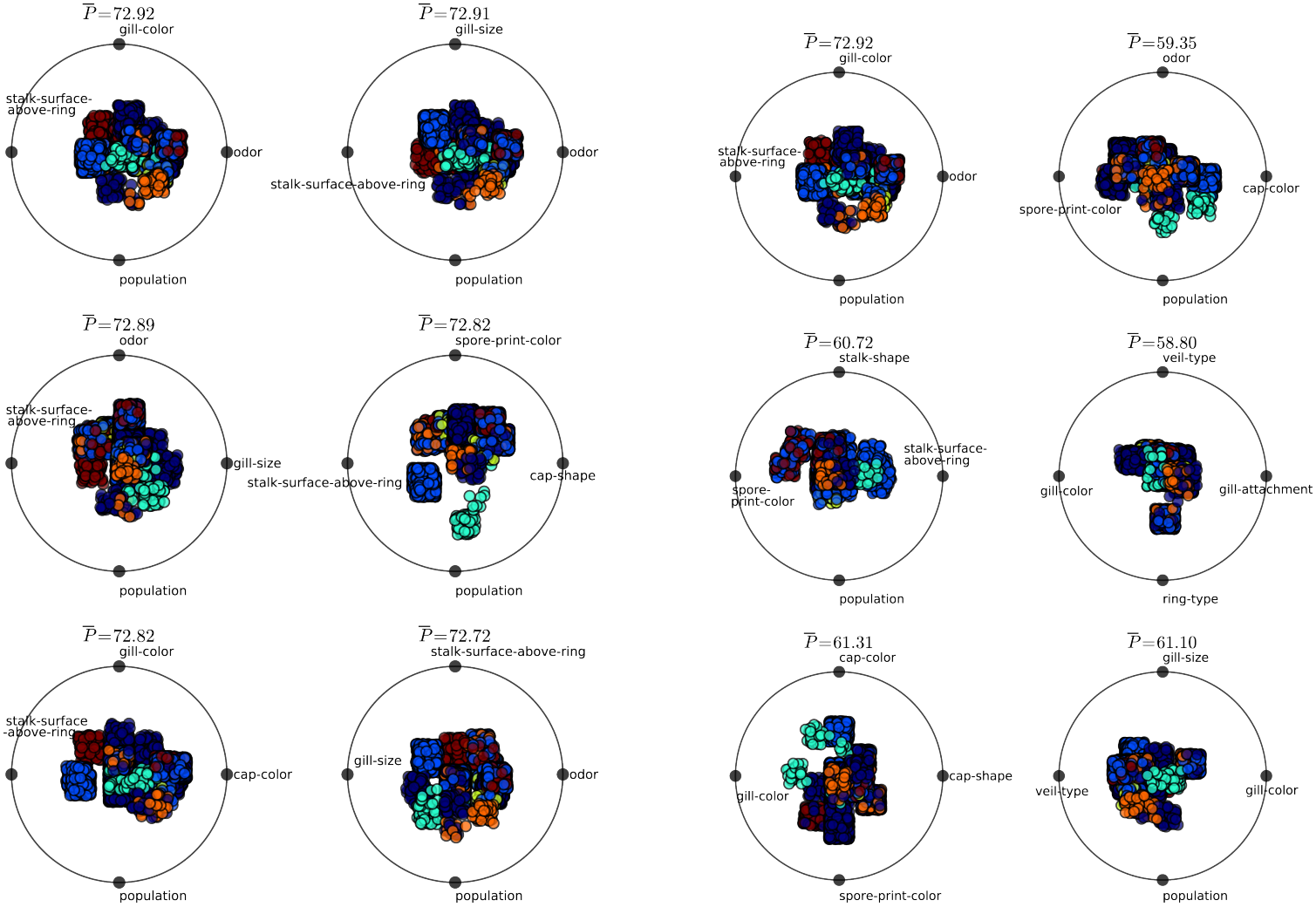
(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.18: The best six Radviz projections (4 attributes) of the Breast Cancer Wisconsin data set.
Node colors: brown (*malign*) and violet (*benign*).



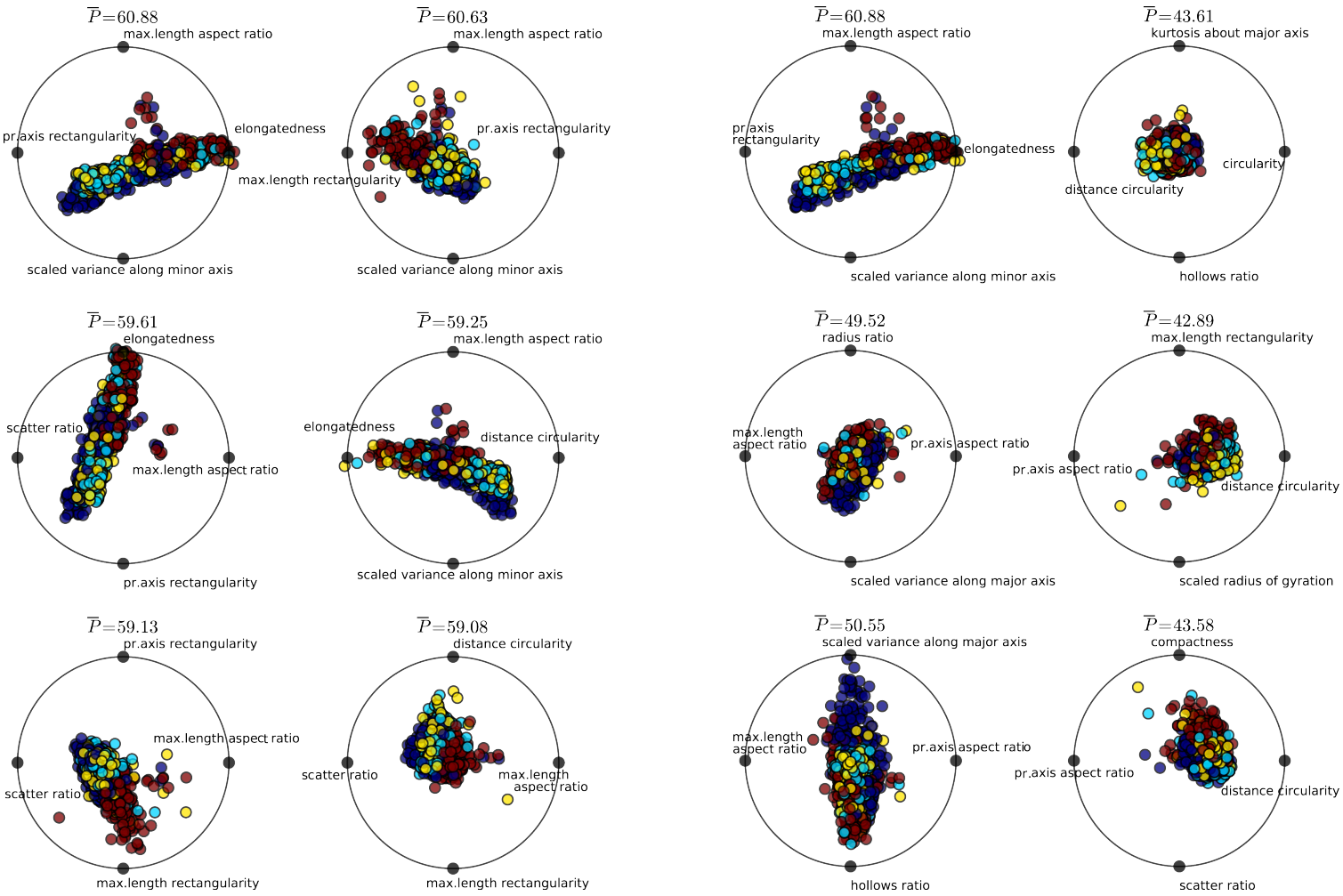
(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.19: The best six Radviz projections (4 attributes) of the Glass data set.
Node colors: violet (*1*), blue (*2*), light blue (*3*), green (*5*), orange (*6*), and brown (*7*).



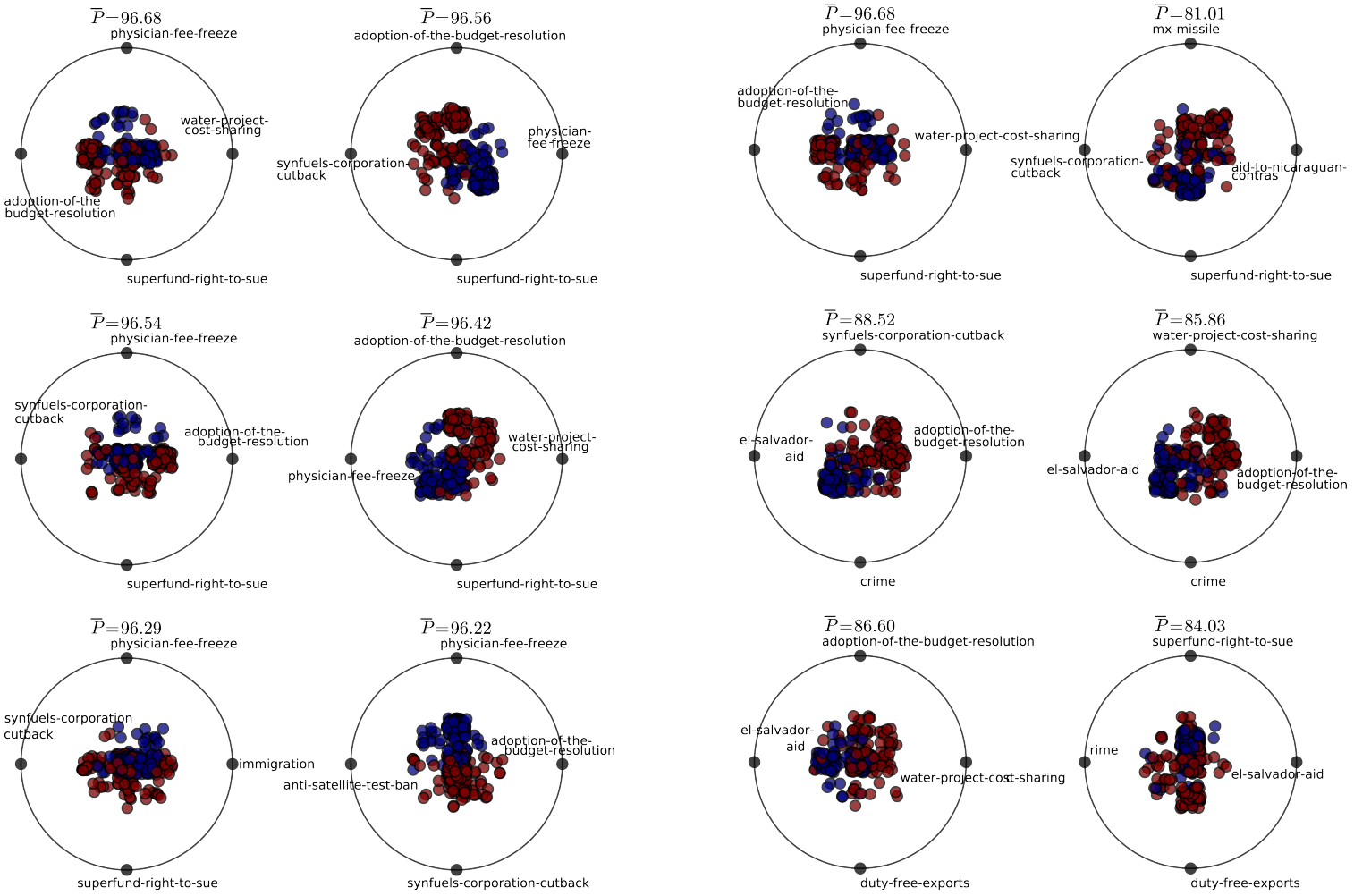
(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.20: The best six Radviz projections (4 attributes) of the Mushroom data set.
Node colors: violet (d), blue (g), light blue (l), orange (p), brown (u), green (m), and red (w).



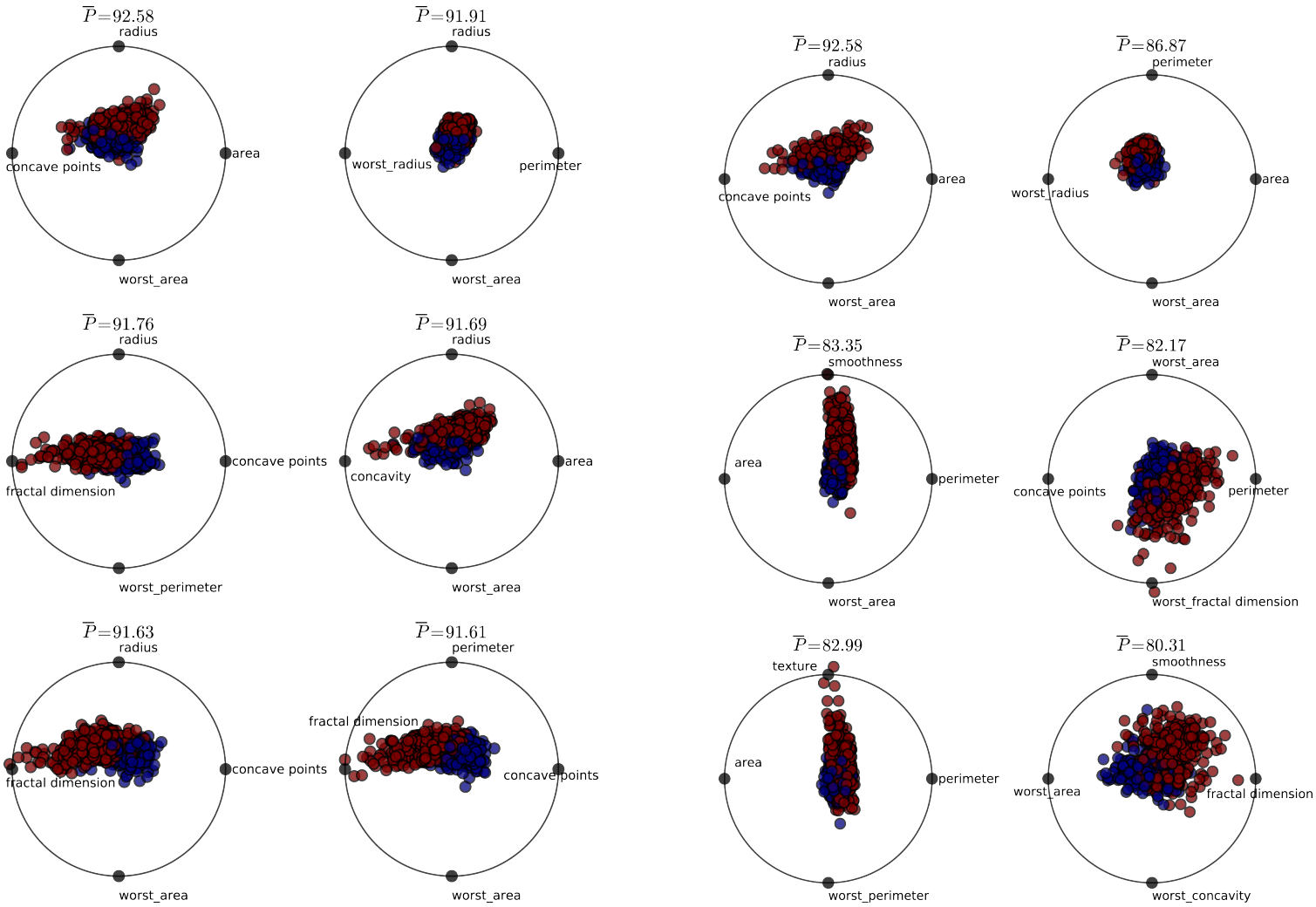
(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.21: The best six Radviz projections (4 attributes) of the Vehicle data set.
Node colors: violet (*bus*), brown (*van*), light blue (*opel*), and yellow (*saab*).



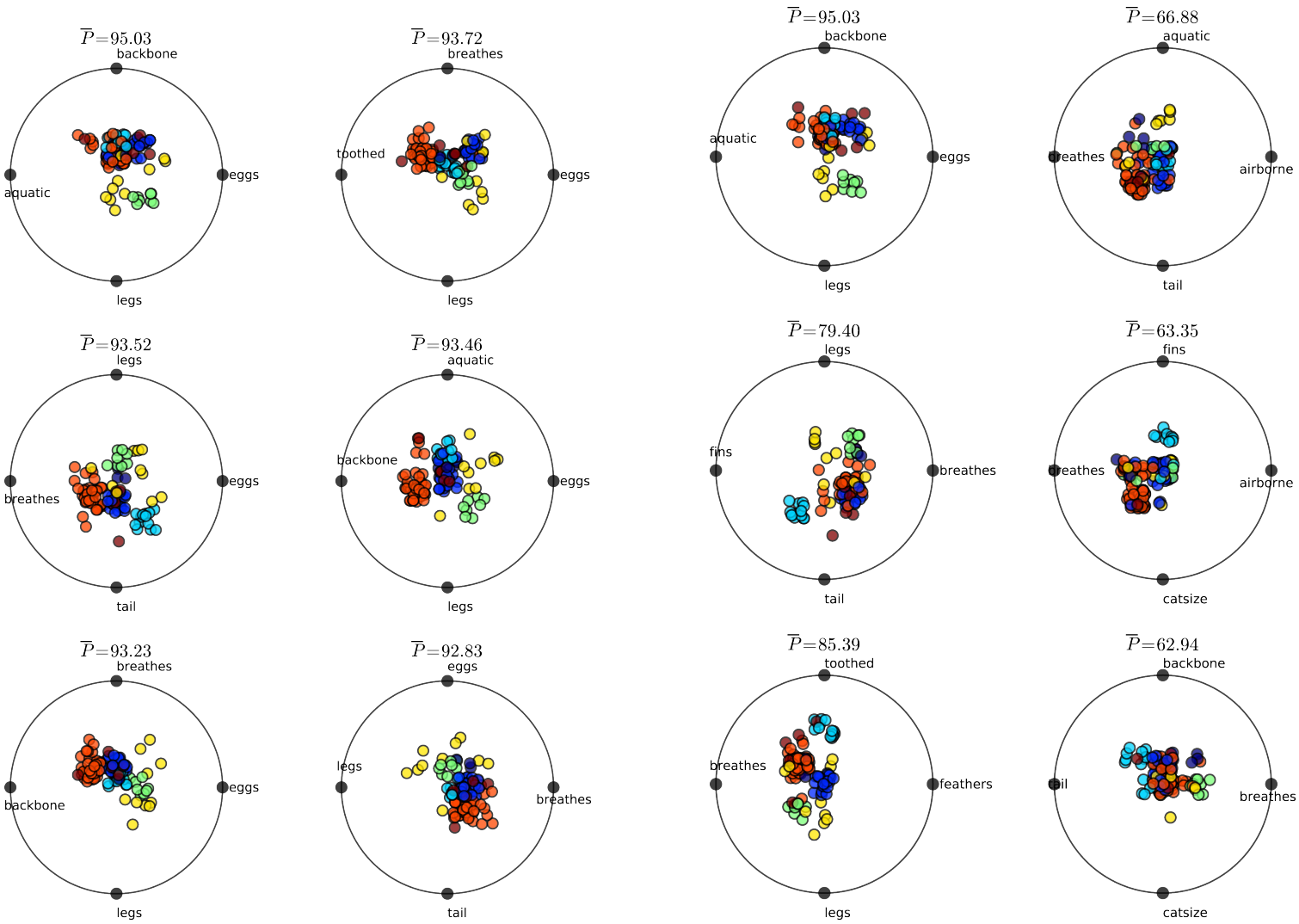
(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.22: The best six Radviz projections (4 attributes) of the Voting data set.
Node colors: brown (*republican*) and violet (*democrat*).



(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.23: The best six Radviz projections (4 attributes) of the WDBC data set.
Node colors: brown (G) and violet (M).



(a) VizRank (b) *Model map (iterative kmeans)*

Figure C.24: The best six Radviz projections (4 attributes) of the Zoo data set.
Node colors: violet (*amphibian*), brown (*reptile*), blue (*bird*), light blue (*insect*), green (*insect*), orange (*mammal*), and yellow (*invertebrate*).

Dodatek D

Razširjeni povzetek v slovenskem jeziku

(Extended Abstract in Slovene Language)

Vizualizacija in analiza prostora napovednih modelov

D.1 Uvod

Pri odkrivanju znanja iz podatkov običajno zgradimo več modelov, ki razložijo zanimive vzorce v podatkih. Modeli morajo biti dovolj preprosti, da jih domenski strokovnjaki razumejo. Drevo z več sto vozlišči ali razsevni diagram, ki slika nekaj deset značilk v dvodimenzionalno ravnino, imata lahko odlično napovedno točnost, a sta morda povsem nerazložljiva. Po drugi strani pa preprost model ne more razložiti zapletenih podatkov z veliko zanimivimi relacijami.

Problem lahko rešimo z večjim številom preprostih modelov, kjer vsak predstavlja vpogled v manjši del problemske domene, skupaj pa tvorijo celotno sliko. Izziv, ki ga takšna predstavitev znanja predstavlja, je, kako množico modelov prestaviti uporabniku. Četudi zmorejo računalniki danes zgraditi na tisoče modelov, si te obsežne zbirke domenski strokovnjak ne more ogledati brez pomoči računalnika.

Zapletene podatke lahko predstavimo z več ponazoritvami, od katerih vsaka izpostavi določen, manjši del podatkovne domene. Na primer, predstavimo jih z linearnimi preslikavami, kjer vsaka prikazuje manjše število značilk. Podobno lahko dobre napovedne modele zgradimo iz zbirke preprostih (šibkih) modelov, na primer z naključnimi gozdovi klasifikacijskih dreves, kjer se vsako osredotoči na svoj del podatkovne domene. Primanjkuje pa metod za raziskovanje takšnih zbirk modelov.

Zanima nas, ali obstaja omejen izbor ponazoritev, ki vsebuje celotno sliko problema in ali se lahko sprehodimo po naključnem gozdu in opazujemo skupne lastnosti dreves v različnih območjih. V disertaciji predstavimo nov pogled na napovedne modele. Gre za postopek ponazoritve in organizacije prostora klasifikacijskih modelov, ki ga lahko raziskujemo z interaktivnim orodjem.

V tem doktorskem delu definiramo napovedni model in različne mere podobnosti med njimi. Opišemo metode za sestavo grafa iz zbirke napovednih modelov ter nekaterih vrst vizualizacij; te prav tako obravnavamo kot napovedni model. Predstavimo izgradnjo, vizualizacijo in analizo omrežij napovednih modelov, ki jih pravimo *zemljevid modelov*.

Najprej smo opisali orodja za analizo, vizualizacijo in interaktivno preiskovanje omrežij, kot so igraph, statnet, Gephi, Network Workbench, GUESS in podobna. Izdelali smo poglobljeno primerjavo z orodjem Orange Network. Ta vsebuje tudi metode, ki smo jih razvili v okviru doktorskega dela. Osredotočili smo se predvsem na razlike in prednosti, ki jih naše orodje nudi v primerjavi z obstoječimi orodji.

Omrežje lahko vizualiziramo na mnogo načinov, ki ponazarjajo različne povezave (relacije) med točkami. V doktorskem delu smo predstavili metode za risanje omrežij. Posebno pozornost smo namenili metodi FragViz [107], ki smo jo razvili za vizualizacijo razdrobljenih omrežij. To so omrežja z velikim številom majhnih nepovezanih podgrafov. Metoda FragViz se je izkazala kot najprimernejša za vizualizacijo omrežja napovednih modelov.

Praktično uporabo *zemljevida modelov* smo prestavili na več primerih. Najprej za razširitev metode VizRank [81, 82], ki oceni vizualizacije glede na njihovo sposobnost ločevanja primerov glede na razred. Metoda VizRank večinoma obravnava linearne projekcije, spremenjene v napovedne modele, ki jih uporabniku ponudi v obliki seznama. Ta je urejen glede na kakovost projekcije. Pomanjkljivost omenjene metode je, da išče najboljše projekcije, a ne upošteva njihove različnosti. To je sicer uporabno, kadar želimo poiskati eno (najboljšo) vizualizacijo izmed desetine podobnih na vrhu seznama. Težko pa na seznamu najdemo ostale zanimive projekcije z drugačnim pogledom na podatke.

Drug primer uporabe *zemljevida modelov* je vizualizacija in raziskovanje naključnih gozdov [18]. Naključni gozd je napovedni model, ki sestoji iz večjega števila naključnih napovednih dreves. Običajno so ta majhna (vsebujejo do 10 vozlišč), zato vsako pokrije le del prostora napovednega problema. Naključni gozd pogosto zelo dobro napoveduje, njegovi pomanjkljivosti, v primerjavi z enim samim napovednim drevesom, pa sta pomanjkanje metod za vizualizacijo in nerazumljivost.

Zemljevid modelov uporabniku pomaga pri raziskovanju gozda. Izpostavi, na primer, različna drevesa, ki dajejo podobne napovedi ali tista, ki napovedujejo drugače. Poleg interaktivnega raziskovanja gozda lahko z omrežjem dreves uporabniku pojasnimo posamezne napovedi. Točke v omrežju, ki predstavljajo posamezna drevesa, si uporabnik, na primer, lahko obarva glede na njihovo napoved.

Raziskali smo tudi uporabnost omrežja modelov za učenje naključnih gozdov. Različni deli omrežja modelov predstavljajo različne poglede na podatke. Zato domnevamo, da bodo modeli, izbrani iz različnih področij omrežja, tvorili dober sestavljen model.

D.2 Vizualizacija in analiza omrežij

Hitra rast interneta, nova socialna omrežja in druge oblike elektronskih komunikacij ustvarjajo ogromne količine podatkov, ki niso zgolj zbirke osamljenih podatkov. Danes so podatki med seboj povezani, te povezave pa vsebujejo veliko skritih informacij, ki razlagajo kompleksne lastnosti. Problemi, ki so povezani z njimi, spodbujajo raziskave zapletenih mrež, njihove vizualizacije in odkrivanja zakonitosti v omrežjih, ki so postala osrednja tema na mnogih znanstvenih področjih.

Razvili smo nov algoritem za optimizacijo položaja točk grafa, FragViz, ki je primeren za "razdrobljena" omrežja (sestavljena iz večjega števila nepovezanih pod-

grafov) in pomaga pri reševanju omenjenih problemov. Najprej opišemo področje vizualizacije omrežij s poudarkom na tehnikah za optimizacijo položaja točk. Nato predstavimo še metodo FragViz, ki jo uporabljamo pri ponazoritvi primerov omrežij.

D.2.1 Analiza omrežij

Raziskovanje kompleksnih omrežij in odkrivanje splošnih zakonitosti, ki v njih veljajo, je danes osrednja tema mnogih znanstvenih disciplin. Številne podatkovne zbirke poleg osnovnih lastnosti o objektih hranijo tudi različne vrste relacij (odnosov ali interakcij) med njimi. Te vsebujejo veliko informacij in razkrivajo kompleksne lastnosti, ki bi sicer pogosto ostale skrite. Analiza omrežij [90, 91] preučuje relacije med objekti.

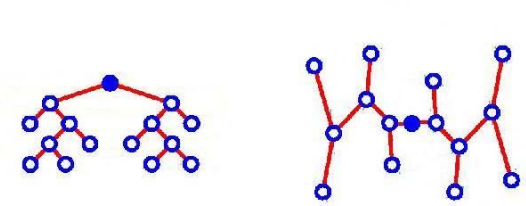
Analiza omrežij je široko področje, ki vključuje (pod-)področja, kot so analiza socialnih omrežij [29], analiza povezav, odkrivanje znanja nad grafi, odkrivanje skupnosti [39] in druga. Področje je prav tako tesno povezano z matematično teorijo grafov, odkrivanjem znanja iz podatkov, strojnim učenjem, statistiko, spektralno analizo in nekaterimi drugimi področji.

Teorija grafov [13] proučuje grafe – matematične strukture, ki jih uporabljamo za upodobitev odnosov med objekti. Graf G je množica točk $V(G)$ (objektov) in povezav $E(G)$ (relacij), ki povezujejo pare točk, torej $G = (V(G), E(G))$. Omrežje je graf z dodatnimi podatki o točkah ali povezavah in je orodje za abstrakcijo sveta. Teorija omrežij je predmet proučevanja uporabne matematike in fizike ter se medsebojno dopolnjuje s teorijo grafov. Uporablja se na mnogih znanstvenih področjih, med drugim v računalniški znanosti, biologiji, ekonomiji in sociologiji. Teorija omrežij proučuje grafe kot predstavitev relacij med diskretnimi objekti.

Na področju analize omrežij so bile opravljene številne raziskave. Področji eksperimentalne analize in odkrivanje znanja iz podatkov sta pri analizi omrežij rodili nove metode in veliko zanimivih rezultatov. Pojavljajo se študije dinamičnih omrežij, ki se s časom spreminjajo, in dinamičnih sistemov na omrežjih, kot je pojav sinhronizacije. Interdisciplinarni projekti združujejo ideje iz statistike, strojnega učenja, algoritmov in drugih področij s hitro rastočo znanostjo o omrežjih.

D.2.2 Optimizacija položaja točk

Graf lahko prikažemo na mnogo načinov; dober prikaz množice točk in povezav med njimi pa je tisti, ki izraža več pomenov. Cilj risanja grafov je torej prikazati graf na takšen način, da je iz slike razvidno čim več. Različni načini risanja so primerni za različne tipe grafov in različne namene predstavitev. Recimo, na sliki D.1 sta prikazana matematično identična grafa. Levi nam poda informacijo o drevesni strukturi grafa s korenom v pobarvani točki, v desnem je drevesna struktura skrita.



Slika D.1: Sliki istega grafa imata lahko različen pomen.

Pri risanju grafov oziroma sestavljanju algoritmov za risanje grafov iščemo takega, ki bo prikazal strukturo grafa tako, da ga bo uporabnik najbolje razumel. Uporabniki imajo subjektivne zahteve, kljub temu pa obstaja nekaj splošno priznanih in merljivih kriterijev. Rezultat naj bo torej slika grafa s čim manj križanj povezav, narisana na majhnem območju, z enakomerno razporeditvijo točk, brez dolgih povezav in z ohranjeno vsebovano simetrijo.

Nekateri od naštetih ciljev se medsebojno izključujejo. V splošnem je problem iskanja najboljše risbe v okviru danih ciljev NP-poln problem.

Naš algoritem FragViz temelji na algoritmu na osnovi sil, ki sta predstavila Thomas M. J. Fruchterman in Edward M. Reingold [45]. Algoritem je prilagoditev Eadesovega sistema vzmeti [35] za risanje neusmerjenih grafov z ravnimi povezavami. Algoritem Fruchterman-Reingold temelji na dveh vodilih:

1. Povezane točke naj si bodo blizu.

2. Točke naj ne bodo narisane preveč skupaj.

Najlažje ga je razložiti z analogijo. Točke se vedejo kot atomi ali nebesna telesa, ki vplivajo druga na drugo s privlačnimi in odbojnimi silami. Te povzročajo gibanje točk. Tako kot pri Eadesovi metodi tudi tukaj računamo odbojne sile med vsemi pari točk, privlačne pa le med sosedi.

Čeprav je algoritem zasnovan po naravnem sistemu, kot sta sistema vzmeti in gravitacije, "sila" ni pravilen izraz. Sile v tem primeru uporabljamo za izračun hitrosti v časovni enoti in s tem premik točke, prave sile pa izražajo pospešek. Ta razlika je ključnega pomena, ker resnična definicija vodi k dinamičnemu ravnotežju, mi pa iščemo statično ravnotežje.

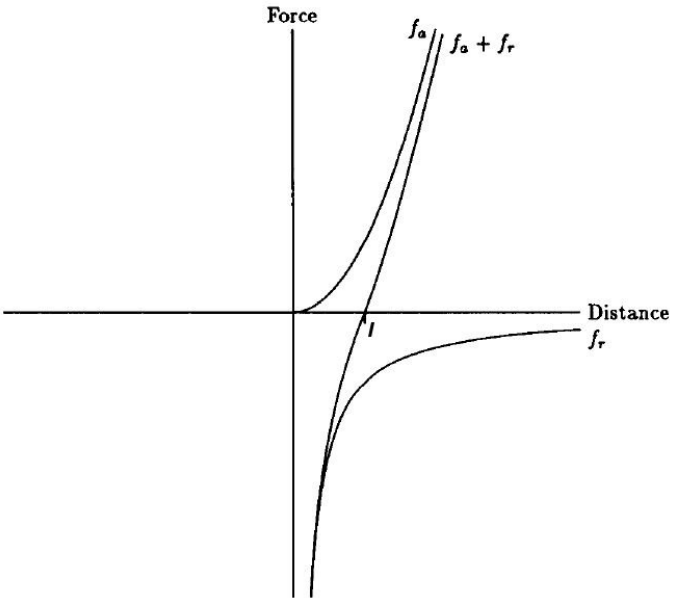
Algoritem je iterativen in vsaka ponovitev vsebuje tri korake. V prvem izračunamo efekt privlačne sile na vsako točko, v drugem izračunamo odbojne sile in na koncu omejimo premik točk s temperaturo, ki predstavlja največji možen premik točke v nekem koraku. Ideja je, da omejimo premik na poljubno izbrano največjo vrednost in da se ta vrednost v vsaki ponovitvi zmanjša. Ko se razporeditev izboljšuje, postajajo prilagoditve vedno bolj "fine".

Želimo, da bi bile točke čim bolj enakomerno razporejene znotraj okvirja, kjer je k radij praznega prostora okrog točke. Formuli za izračun privlačne in odbojne sile izberemo tako, da sta sili v ravnotežju ravno pri k. Če sta točki bolj narazen od k, prevladuje privlačna sila, sicer pa odbojna. Med nepovezanimi točkami privlačna sila ne deluje. Enačba D.1 prikazuje privlačno silo, D.2 pa odbojno.

$$f_a(d) = \frac{d^2}{l} \tag{D.1}$$

$$f_r(d) = \frac{-l^2}{d} \tag{D.2}$$

Slika D.2 prikazuje sili in njuno vsoto v primerjavi z razdaljo. V točki, kjer vsota sil seka vodoravno os, sta privlačna in odbojna sila v ravnotežju. To je ravno pri k, ki je tudi idealna razdalja med dvema točkama.



Slika D.2: Odbojno in privlačno silo primerjamo z razdaljo med dvema točkama.

D.2.3 Vizualizacija razdrobljenih omrežij

Omrežje je pogosto sestavljeno iz več nepovezanih delov – komponent. Običajne tehnike za ponazoritev omrežja, kot so metode Fruchterman-Reingold, Kamada-Kawai [70] in Frick (s sodelavci) [41], postavijo komponente naključno. To lahko povzroči napačno interpretacijo sicer nepovezanih komponent.

Primer je slika D.3, ki predstavlja štiri komponente iz mreže napovednih modelov. Točke predstavljajo napovedni model, ki je naučen na podatkih Dermatology, povezave pa podobnost med njimi. V tem primeru so modeli, ki so zgrajeni z istimi

značilkami, podobni in zato povezani. Iz njihove postavitve na sliki D.3a lahko napačno sklepamo, da sta modela z značilkami "itching" in"definite borders" bolj podobna modeloma z značilkami "PNL" in "fibrisis" kot ostalim modelom v grafu. Takšnim napačnim razlagam se lahko izognemo, če najprej narišemo največjo komponento, ki ji sledi seznam drugih.

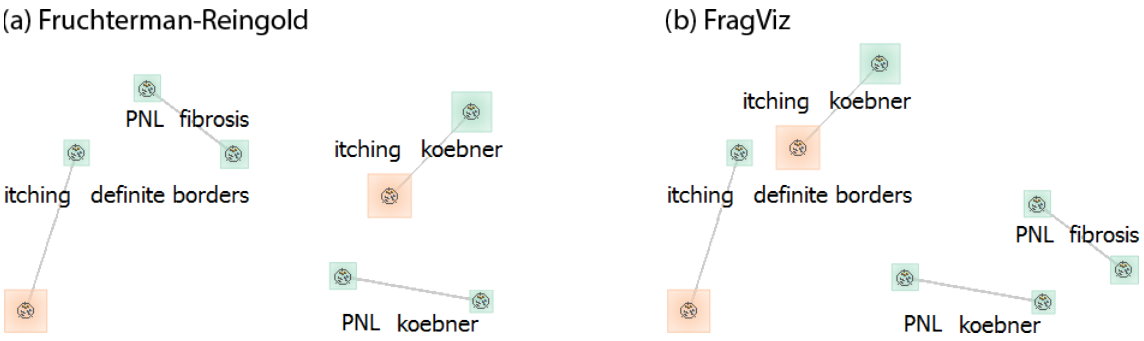
Razvili smo metodo FragViz, ki razporedi komponente glede na predznanje o njihovi podobnosti. Na primer, če točke razporedimo z algoritmom FragViz, dobimo sliko D.3b, iz katere lahko pravilno razberemo podobnost štirih modelov, ki vključujejo značilke "backbone", "breathes", "toothed" in "feathers".

Ta algoritem sprejme seznam komponent in matriko podobnosti med točkami. Metoda FragViz ima dva koraka. Najprej uporabi metodo Fruchterman-Reingold na vsaki komponenti posebej, potem pa ustvari globalno sliko. Komponente razporedi s fizikalnim modelom, v katerem so "sile" med komponentami sorazmerne njihovi podobnosti. Glavni prispevek metode FragViz je drugi korak, ki ga predstavimo spodaj.

Naj bo graf $\mathcal{G} = (V, E)$ sestavljen iz p disjunktnih komponent $V = \bigcup_{k=1}^{p} V_k$ in matrika podobnosti D oblike $|V| \times |V|$. Razporeditev točk znotraj komponente V_k je fiksna, položaj točk pa dan v internem koordinatnem sistemu vsake komponente z izhodiščem v težišču točk. Položaj točke v_i bomo označili z $\mathbf{v_i}$.

Cilj je poiskati položaj komponent $\mathbf{c_k}$ in usmerjenost njenega koordinatnega sistema $\boldsymbol{\phi_k}$, da kar najbolje odraža matriko razdalj D.

Metodo si lahko predstavljamo z metaforo: komponente so plošče, na katere so privarjeni obročki, ki predstavljajo točke. Na njih so vpete vzmeti (njihova trdota odraža razdaljo iz matrike D), ki povezujejo različne plošče in predstavljajo povezave. Narava (v našem primeru računalniška simulacija) postavi plošče tako, da so "sile" v vzmeteh v ravnovesju.



Slika D.3: Štiri komponente iz mreže modelov na podatkih Dermatology. Na levi so točke razporejene z algoritmom Fruchterman-Reingold, na desni pa z algoritmom FragViz, ki upošteva tudi dodatno informacijo o razdaljah med točkami.

Privzemimo, da imajo komponente enako maso m. Masa komponente V_k je

$$m_k = |V_k|m, \tag{D.3}$$

njen vztrajnostni moment pa

$$I_k = m \sum_{v_i \in V_k} ||\mathbf{v_i}||^2. \tag{D.4}$$

Silo med paroma točk (v_i, v_j) definiramo s Hookovim zakonom,

$$\mathbf{F_{ij}} = (d_{ij} - ||\mathbf{g_i} - \mathbf{g_j}||)\frac{\mathbf{g_i} - \mathbf{g_j}}{||\mathbf{g_i} - \mathbf{g_j}||}, \tag{D.5}$$

kjer sta $\mathbf{g_i}$ and $\mathbf{g_j}$ poziciji točk v globalnem koordinatnem sistemu

$$\mathbf{g_i} = \mathbf{v_i} + \mathbf{c_k}, \tag{D.6}$$

za takšen k, da velja $v_i \in V_k$. Naj bo $\mathbf{F_i}$ vsota sil na točko v_i:

$$\mathbf{F_i} = \sum_{v_j \in V} \mathbf{F_{ij}}. \tag{D.7}$$

Sile povzročijo linearni pospešek in kotni pospešek komponente:

$$\mathbf{a_k} = \frac{\sum_{v_i \in V_k} \mathbf{F_i}}{m_k} \tag{D.8}$$

$$\boldsymbol{\alpha_k} = \frac{\sum_{v_i \in V_k} \mathbf{F_i} \times \mathbf{v_i}}{I_k} \tag{D.9}$$

Predpostavimo neskončno trenje, zato komponenta ne obdrži nobenega momenta. Za vsako točko se komponenta premakne za razdaljo, ki je sorazmerna linearnemu pospešku $\Delta\mathbf{c_k} \sim \mathbf{a_k}$ in se zavrti za kot, ki je sorazmeren kotnemu pospešku $\boldsymbol{\Delta}\phi_\mathbf{k} \sim \alpha_\mathbf{k}$ tako, da velja:

$$\Delta\mathbf{c_k} \sim \frac{\sum_{v_i \in V_k} \mathbf{F_i}}{|V_k|} \tag{D.10}$$

in

$$\Delta\phi_\mathbf{k} \sim \frac{\sum_{v_i \in V_k} \mathbf{F_i} \times \mathbf{v_i}}{\sum_{v_i \in V_k} ||\mathbf{v_i}||^2}. \tag{D.11}$$

Po teh enačbah smo naredili simulacijo fizikalnega procesa, ki naključno določi koordinate komponent in iterativno računa sile $\mathbf{F_i}$ ter premika in vrti komponente, dokler ne doseže stanja z minimalno energijo, kjer so vse sile $\mathbf{F_i}$ zanemarljivo majhne.

Računalniška simulacija opisanega sistema je počasna. Pospešimo jo lahko, če najprej postavimo komponente na pravo mesto in jih šele nato zavrtimo. Približek metode računa razdaljo med komponentami namesto razdalje med točkami. Razdaljo med komponentama V_k in V_l definiramo kot povprečno razdaljo med njunimi točkami, podobno kot je definirana povprečna povezanost pri hierarhičnem razvrščanju [103]:

$$\delta_{kl} = \frac{1}{|V_k||V_l|} \sum_{\substack{v_i \in V_k \\ v_j \in V_l}} d_{ij}. \tag{D.12}$$

Iščemo razporeditev, kjer razdalja med težišči parov komponent $\mathbf{c_k}$ in $\mathbf{c_l}$ odraža dano razdaljo δ_{kl}. Postopek je precej hitrejši, ker smo računanje razdalj med vsemi točkami v vsaki iteraciji zamenjali z enim računanjem (po enačbi D.12) na začetku optimizacije. Problem smo prevedli na znani problem večdimenzionalnega lestvičenja, za katerega obstaja veliko učinkovitih rešitev, na primer SMACOFF [84].

Za vrtenje uporabimo izvorno definicijo sile (D.5), ki jo izračunamo na končni razporeditvi. Uporabimo enak postopek kot pri simulaciji, toda računamo le kot vrtenja. Lokalnim minimumom se izognemo s simuliranim ohlajanjem. Postopek v praksi ne porabi veliko časa (čeprav računa razdalje med vsemi pari točk), ker je ponovitev malo.

Uporabnost metode smo predstavili na štirih primerih bioloških mrež. Vse temeljijo na omrežju genov, izraženih pri levkemiji. Naša metoda izpostavi nekatere značilne povezave med geni in skupinami genov, ki bi sicer ostale skrite. Poleg znanih dejstev smo prišli do novih spoznanj o bolezni, ki jih morajo klinične študije še potrditi.

D.3 Orange in Net Explorer

Z novim orodjem za raziskovanje omrežij odgovarjamo na nove trende kot sta odzivna analiza omrežja v realnem času in interaktivnost. Večina orodij za raziskovanje omrežij je namreč omejenih na vizualizacijo grafov in računanje osnovnih (splošnih) značilnosti, ki v njih veljajo. Omrežje lahko z njimi analiziramo kot celoto. Manj je raziskav na področju interaktivnosti, na primer: lokalno omejeno raziskovanje omrežja, samostojna obdelava podomrežja, testiranje odvisnosti strukture grafa od vhodnih parametrov itd.

D.3.1 Sorodna orodja

Veliko orodij za risanje omrežij in računanje njihovih značilnosti je na voljo zastonj (Pajek [10], NetworkX [51], Graphviz [37], Gephi [8], Network Workbench [92]) ali v komercialnih paketih (na primer Net Miner [27]). Pregled nekaterih je v tabeli D.1.

Tabela D.1: Pregled orodij za analizo omrežij.

	Odprto-kodno	Interaktivno	Skriptni vmesnik (v Pythonu)
Pajek			
NetMiner		×	
NetworkX	×		× (×)
Graphviz	×		× (×)
igraph	×		× (×)
statnet	×		×
Gephi	×	×	
Network Workbench	×	×	
Orange Network	×	×	× (×)

Večina podaja znanje o grafu kot celoti. Orodje Pajek je, na primer, usmerjeno v vizualizacijo velikih omrežij, stanet [53] pa v simuliranje naključnih eksponentnih modelov grafov in statističnih analiz. Manj raziskav je na področju interaktivnost. Recimo, da imamo omrežje genov, kjer povezave predstavljajo medsebojno izraženost dveh genov ali omrežje raziskovalnih člankov, kjer so povezani tisti s podobnimi ključnimi besedami. Želeli bi orodje, kjer se sprememba parametrov pri konstrukciji grafa na vizualizaciji odrazi takoj, ker se zanimivi vzorci pogosto pokažejo šele ob pravi kombinaciji vhodnih parametrov.

Podoben problem je filtriranje točk, ki mora biti možno bodisi s skripto bodisi interaktivno v programu. Interaktivnost nam koristi tudi pri raziskovanju lokalnih zakonitosti omrežja. Vprašanje "Kateri geni so najbolj podobni izraženim pri levkemiji?" je gotovo bolj zanimivo kot "Kako dobro potenčni zakon odraža porazdelitev stopenj točk v grafu?".

Rezultat raziskovanja je pogosto pod-omrežje, ki ga želimo raziskati bolj podrobno. Ko najdemo neko skupnost v omrežju socialne mreže, bi radi izvedeli, kako se ta razlikuje od ostale populacije. Če v omrežju najdemo zdravila s podobnim učinkom na telo, nas zanima, kaj imajo skupnega. Na takšna vprašanja lažje odgovorimo, če je naše orodje del večje zbirke orodij za statistično analizo, strojno učenje ali odkrivanje zakonitosti v podatkih.

D.3.2 Gradniki za vizualizacijo in raziskovanje omrežja

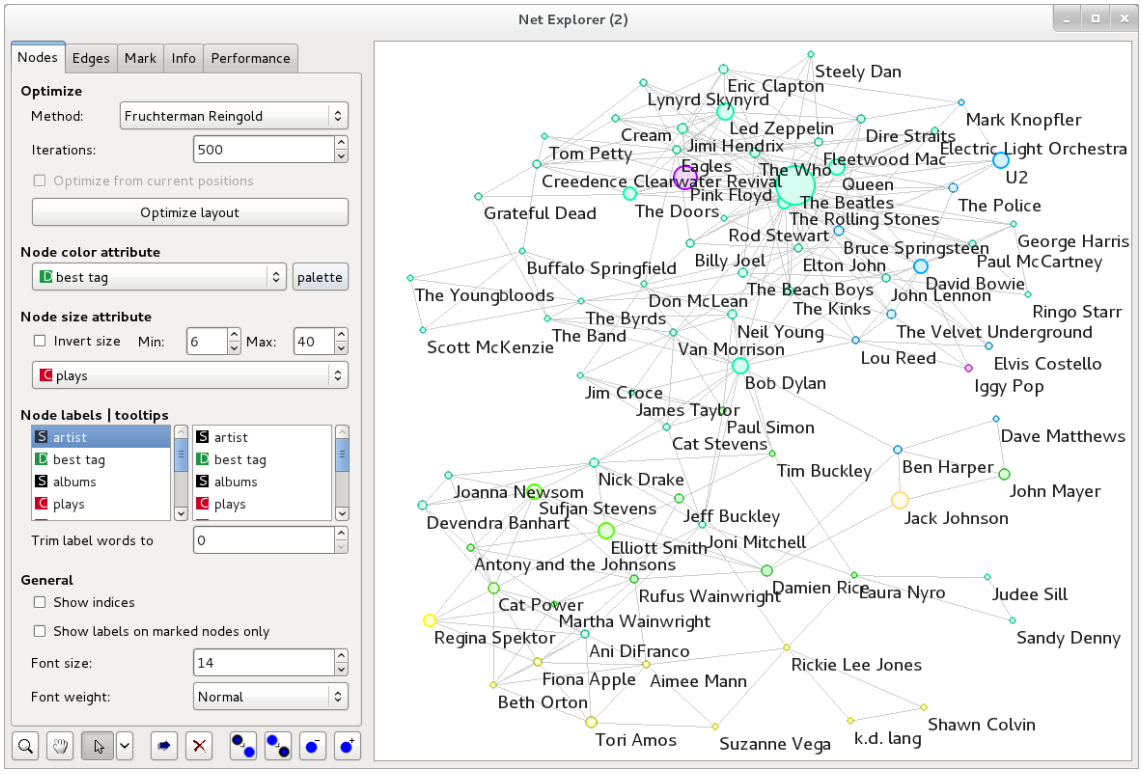
Razvili smo gradnike za delo z omrežji Orange Network, ki so na voljo kot dodatek (angl. *add-on*) k orodju za učenje iz podatkov Orange [30]. Naše orodje je tudi edino orodje za analizo omrežij v programskem jeziku Python, ki omogočajo interaktivno vizualizacijo. Zbirka gradnikov za raziskovanje omrežij vključuje:

- `Net File:` gradnik, ki prebere podatke iz datotek. Te so lahko tipa Pajek, Graph Modeling Language (GML [59]) ali kot objekt NetworkX. Uporabnik lahko omrežju pripne dodatne podatke o točkah in povezavah.

- `Net from Distances:` gradnik, ki zgradi omrežje iz matrike razdalj. Točke poveže glede na različna merila: razdalja med točkama naj bo znotraj predpisanega intervala, točko poveži s k najbližjimi sosedi ali poveži točke, med katerimi so razdalje znotraj določenega percentila. Del gradnika je tudi histogram razdalj. Poleg njega vključuje še nekatere filtre: vrni celotno omrežje, največjo komponento v omrežju, točke z najmanj eno povezavo in podobne.

- `Net Explorer`: gradnik za vizualizacijo in interaktivno predstavitev omrežja. Z gradnikom lahko izvajamo operacije v realnem času in brez motečih zamikov nad grafi velikosti do nekaj deset tisoč točk. Velikost grafa sicer ni omejena, a se delovanje gradnika precej upočasni pri grafih z več sto tisoč točkami. Primer glavnega okna gradnika je na sliki D.4. Z ostalimi gradniki se lahko poveže z več vhodnimi signali: *Network* – omrežje, *Items* – dodatni podatki o omrežju, *Item Subset* – podmnožica točk, ki naj jo izpostavi, *Distances* – razdalje med točkami in *Net View* – vtičnik za razširitev gradnika. Gradnik lahko posreduje podgraf in pripadajoče podatke ter matriko razdalj.

  Omrežje lahko ponazorimo z več algoritmi: naključna razporeditev točk, algoritem Fruchterman-Reingold, uteženi algoritem Fruchterman-Reingold, algoritem za krožno razporeditev točk, algoritem za krožno razporeditev točk s



Slika D.4: Primer gradnika Net Explorer.

čim manj križanji povezav, FragViz [107], MDS [84] in Pivot MDS [15]. Zadnji trije potrebujejo na vhodu matriko razdalj.

Omrežju lahko nastavljamo razne vizualne lastnosti kot so velikost in barva točk ter povezav (glede na vrednosti priloženih podatkov), izpis oznak točk. Kadar je točk veliko, lahko omejimo izpis oznak le na točke pod ali v okolici miškinega kazalca.

Točke lahko izbiramo in nad njimi izvajamo številne operacije. Izbiramo jih lahko po različnih merilih: ročno z miško (posamezno točko, oziroma povlečemo pravokotnik in izberemo skupino točk znotraj njega) ali glede na značilnosti omrežja. Glede na število povezav in vrednosti podatkov v izbor dodamo točke znotraj določene razdalje.

- `Net View plugins`: gradnik, ki pošlje gradniku `Net Explorer` novo funkcionalnost, na primer, lokalni pogled na omrežje (vtičnik `Net Inside View`), ki uporabniku omogoči "sprehod" po točkah. V sredini je prikazana izbrana točka, okoli nje pa množica sosednjih točk do določene razdalje. Ko uporabnik klikne na novo točke, se njihov razpored prilagodi okolici njenih sosedov.

- `Net Analysis`: gradnik, ki izračuna statistike omrežja ali točk v njem. Primeri statistik omrežja so: število točk in povezav, povprečna stopnja, premer grafa, povprečna dolžina najkrajše poti, gostota, število klik, povprečni količnik razvrščanja ali število komponent. Statistike točk gradnik posreduje naprej v tabeli s točkami v vrsticah in statistikami v stolpcih. Nekatere med njimi so: stopnja točke, povprečna stopnja njenih sosedov, količnik gručenja, število trikotnikov, klik, jeder in druge.

- `Net Clustering`: gradnik, ki poišče skupine točk v omrežju. Implementirana sta dva algoritma [99, 83]. Gradnik pripne podatkom o točkah še informacijo o skupini, ki ji pripada posamezna točka. Te lahko nato pobarvamo v gradniku `Net Explorer`.

D.4 Prostor klasifikacijskih modelov

Razvili smo metodo za izgradnjo omrežja napovednih modelov, ki jih lahko uporabnik interaktivno raziskuje z orodjem za njihovo ponazoritev. Definirali smo mero podobnosti med napovednimi modeli. Naš ključni prispevek je zamisel, da sta dva modela podobna, kadar dajeta podobne napovedi in ne kadar sta videti podobna (sta istega tipa ali zgrajena nad isto množico značilk).

D.4.1 Razdalja med napovednimi modeli

Razdaljo med modeloma m_i in m_j definiramo kot podobnost med njunimi napovedmi. Primerjamo lahko napovedani razred ali verjetnost napovedi. Najprej smo

predpisali razdaljo na osnovi relativne frekvence enake napovedi:

$$\delta(m_i, m_j) = 1 - \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} |\{c_{m_i}(e)\} \cap \{c_{m_j}(e)\}|, \tag{D.13}$$

kjer je $\mathcal{E}$ množica primerov, $c_{m_i}(e)$ pa napoved razreda za primer e, ki jo da model m_i. Modele lahko primerjamo tudi z medsebojno informacijo:

$$\delta(m_i, m_j) = 1 - 2\frac{I(c_{m_i}; c_{m_j})}{H(c_{m_i}) + H(c_{m_j})}, \tag{D.14}$$

kjer je $c_{m_i} : \mathcal{E} \to \mathcal{C}$ diskretna naključna spremenljivka, ki ustreza napovedim modela m_i, $\mathcal{C}$ je zaloga vrednosti razredne značilke, $I(c_{m_i}; c_{m_j})$ je medsebojna informacija med c_{m_i} in c_{m_j}, $H(c_{m_i})$ in $H(c_{m_j})$ pa sta mejni entropiji.

Takšni razdalji sta pogosto preveč "grobi", kar je nezaželeno, ko želimo sestaviti omrežje modelov. Drugi razlog za uvedbo mer na osnovi verjetnosti je želja, da bi razdalja kar najbolje odražala napoved modelov. Če imamo modele, ki znajo oceniti verjetnost napovedi, lahko definiramo povprečno evklidsko razdaljo med njimi:

$$\delta(m_i, m_j) = \frac{1}{\sqrt{2}|\mathcal{E}|} \sum_{e \in \mathcal{E}} \|p_{m_i}(e) - p_{m_j}(e)\|, \tag{D.15}$$

kjer sta $p_{m_i}(e)$ in $p_{m_j}(e)$ vektorja verjetnosti napovedi za vse razrede za primer e, ki ju vrneta modela m_i in m_j. Razdalja je normalizirana na interval $[0, 1]$.

Preizkusili smo tudi manhattansko razdaljo:

$$\delta(m_i, m_j) = \frac{1}{\sqrt{2}|\mathcal{E}|} \sum_{e \in \mathcal{E}} \|p_{m_i}(e) - p_{m_j}(e)\|_1, \tag{D.16}$$

ker smo pričakovali hitrejše računanje, saj ne potrebuje računsko zahtevnih operacij korenjenja in kvadriranja. Izkaže se, da je približno $1,5$-krat hitrejša od evklidske.

Nazadnje smo preizkusili še mero statistične odvisnosti med modeli. Ker nas zanima predvsem klasifikacija, smo uporabili Spearmanov koeficient korelacije [106], ki primerja range:

$$\delta(m_i, m_j) = 1 - \frac{\sum_{e \in \mathcal{E}; c \in \mathcal{C}} (r^c_{m_i}(e) - \overline{r_{m_i}})(r^c_{m_j}(e) - \overline{r_{m_j}})}{\sqrt{\sum_{e \in \mathcal{E}; c \in \mathcal{C}} (r^c_{m_i}(e) - \overline{r_{m_i}})^2 (r^c_{m_j}(e) - \overline{r_{m_j}})^2}}, \tag{D.17}$$

kjer sta $r^c_{m_i}(e)$ in $r^c_{m_j}(e)$ ranga napovedanih verjetnosti razreda c za primer e, ki ju vrneta modela m_i in m_j, $\overline{r_{m_i}}$ in $\overline{r_{m_j}}$ pa sta povprečji rangov.

D.4.2 Primerjava razdalj

Metrike smo primerjali na petih množicah podatkov iz zbirke UCI: Breast Cancer Wisconsin, Dermatology, Iris, Voting in Zoo (njihove lastnosti so v tabeli D.2). Za vsak tip podatkov smo zgradili več kot 3.500 projekcij in pripadajočih modelov k-najbližjih sosedov (k-NN) v preslikanem prostoru. Njihovo število je odvisno od števila značilk. Vsak model smo naučili desetkrat znotraj 10-kratnega prečnega preverjanja. Matrike razdalj smo izračunali za mere, ki smo jih opisali v prejšnjem poglavju.

Tabela D.2: Lastnosti petih množic podatkov iz zbirke UCI.

	Primeri	Diskretne značilke	Zvezne značilke	Vrednosti razreda
Breast Cancer Wisconsin	683	9	0	2
Dermatology	366	0	34	6
Iris	150	0	4	3
Voting	435	16	0	2
Zoo	101	16	0	7

Matrike smo primerjali s Spearmanovim testom. Korelacija med vsemi pari matrik je visoka (nad $0,7$), vse pripadajoče p-vrednosti so pod pragom $0,01$. Rezultati so v prilogi B.1. Če interpretiramo verjetnosti kot $|\mathcal{C}|$-dimenzionalne vektorje, predstavlja razlika, definirana v enačbi D.15, evklidsko razdaljo med modeli. Ker želimo prostor modelov risati v evklidski ravnini, imamo evklidske ali razdalje podobne evklidskim raje kot tiste, ki temeljijo, na primer, na Kullback-Leiblerjevi razliki. Ker sta si evklidska in manhattanska razdalja zelo podobni (korelacija med njima je vedno na $0,96$) in ker izračunamo manhattansko razdaljo $1,77$-krat hitreje, jo uporabljamo v primerih v disertaciji.

D.4.3 Omrežje modelov

V tem poglavju predlagamo metodologijo za izgradnjo omrežja klasifikacijskih modelov. Naj bo $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$ množica klasifikacijskih modelov in $\delta(m_i, m_j)$ mera razdalje med modeloma m_i in m_j. Predpostavimo, da množica vsebuje klasifikacijske modele in je lahko lahko homo- ali heterogena. Vsebuje lahko, na primer, klasifikacijska drevesa, metode SVM, razsevne diagrame in ostale projekcije, ki so preoblikovane v napovedni model k-NN, naučen na točkah projekcije. Modele v množici želimo urediti na način, ki bo omogočal raziskovanje prostora modelov. Radi bi sestavili, narisali in raziskovali omrežje modelov, ki mu pravimo tudi *zemljevid modelov*.

Vhodni podatki so množica modelov in verjetnosti njihovih napovedi za vsak primer. Pri tem moramo paziti, da napovedani primer ni v učni množici. V doktorskem delu smo uporabili 10-kratno prečno preverjanje. Tako smo se vsak model naučili desetkrat, in na njem napovedali preostalo desetino podatkov. Napovedi smo nato združili.

Predlagana metoda iz množice modelov $\mathcal{M}$ zgradi množico predstavnikov $\mathcal{M}'$. Ker želimo obdržati raznolikost, najprej razdelimo množico $\mathcal{M}$ v homogene podmnožice $\mathcal{M}_1$, $\mathcal{M}_2$, ..., $\mathcal{M}_l$, kjer vsaka vsebuje le modele ene vrste. Nad podmnožico $\mathcal{M}_i$ uporabimo poljubno metodo za razvrščanje in iz vsake izmed gruč izberemo po enega predstavnika glede na naš cilj: bodisi izberemo povprečen model, ki dobro ponazarja skupino, bodisi najboljšega glede na poljubno mero kakovosti. Predstavniki modelov iz množice $\mathcal{M}_i$ tvorijo novo množico $\mathcal{M}'_i$. Končna skupina modelov je unija vseh predstavnikov $\mathcal{M}' = \bigcup_i \mathcal{M}'_i$.

Napovedne modele predstavimo v obliki omrežja, v katerem vsakega povežemo s k najbližjimi modeli v $\mathcal{M}'$ (običajno je k ena ali dva). Omrežje modelov pogosto sestoji iz manjših nepovezanih komponent, zato ga ponazorimo z algoritmom FragViz.

D.5 Prostor vizualizacij

Iskanje dobrih vizualizacij je izziv, ki je s pojavom velikih količin podatkov vedno bolj zanimiv. Dobre vizualizacije, oziroma metode za iskanje le teh, so pomembne tudi na manjših zbirkah podatkov. Recimo, da želimo najti najboljši prikaz podatkov o živalih (Zoo data set) z metodo Radviz. Pregledati bi morali več kot 600 milijard različnih projekcij. Pri tem nam lahko pomagajo nekatere metode za avtomatsko iskanje projekcij. Metoda VizRank [82] je ena najbolj uspešnih in splošnih med njimi. Vendar pa nam pomaga le pri iskanju najboljše projekcije, iskanje raznovrstnih pa je še vedno odprt problem. Seznam desetih najboljših projekcij ekspertu namreč ni v veliko pomoč, če so vse skoraj identične.

Ideje iz poglavja D.4 bomo uporabili za razširitev metode VizRank. Leban s soavtorji [81] je raziskoval ocenjevanje vizualizacij glede na njihovo sposobnost ločevanja primerov na razrede. Njegova metoda (VizRank) obravnava projekcije, ki so spremenjene v napovedne modele. Te uredi po kakovosti in jih ponudi uporabniku v obliki seznama. Ocena projekcije je napovedna točnost modela najbližjih sosedov nad projeciranimi koordinatami. Naša metoda upošteva tudi raznolikost projekcij. Pokazali bomo, da nam vrne projekcije, ki imajo več informacij o podatkih pri enakem številu projekcij. V splošnem bi našo metodo lahko uporabili s poljubno metodo za rangiranje vizualizacij in je zato splošnejša od metode VizRank.

D.5.1 Metoda VizRank

Metoda VizRank ovrednoti poljubno projekcijo točk v ravnino na danih podatkih. Kot rezultat vrne seznam v katerem so projekcije urejene po "koristnosti". Opisali jo bomo v dveh korakih:

1. Izdela predstavitev podatkov (projekcijo) nad izbrano množico značilk.
2. Oceni koristnost predstavitve.

Prvi korak je samoumeven (tako v originalnem članku kot v doktorski disertaciji primerjamo projekcije tipa razsevni diagram in Radviz). Drugi korak bomo razložili bolj podrobno. VizRank poskuša oceniti, kako verjetno je, da bo uporabnik v projekciji videl vzorce. Avtorji trdijo, da je to povezano s tem, kako dobro projekcija loči podatke, kar ocenijo z napovednim algoritmom k-najbližjih sosedov (k-NN), ki ga naučijo na preslikanih podatkih (x in y koordinatah primerov). Algoritem k-najbližjih sosedov je primeren ravno zaradi svoje pristranosti, ki tvori "vizualno očitna" klasifikacijska pravila. V primerih smo v algoritmu k-NN uporabili evklidsko razdaljo, parameter k pa je enak kvadratnemu korenu števila instanc, kot predlaga Dasarathy [28].

Članek VizRank predlaga štiri mere kakovosti, od katerih se je za najboljšo izkazala povprečna verjetnost $\overline{P}$, da napovemo pravilni razred:

$$\overline{P} = E(P_f(y|x)) = \frac{1}{N}\sum_{i=1}^{N} P_f(y_i|x_i), \tag{D.18}$$

kjer je N število primerov, $P_f(y_i|x_i)$ verjetnost pravilno napovedane vrednosti y_i za primer x_i z modelom f, ki je v našem primeru model k-NN. Projekcije smo ocenili z 10-kratnim prečnim preverjanjem.

D.5.2 Vrednotenje projekcij z omrežjem modelov

Zamisel *zemljevida modelov* smo razširili na prostor projekcij. V tem primeru vsebuje množica $\mathcal{M}$ modele ene same vrste in je ni treba razdeliti na podmnožice. Naša metoda projekcije najprej razvrsti v gruče. Preizkusili smo pet metod za razvrščanje:

- Metodo k-means, kjer je k enak številu projekcij, ki jih želimo prikazati. V našem primeru je $k = 15$. Slabost te metode je, da moramo k podati vnaprej, kar je še posebej problematično pri raziskovalni analizi podatkov.
- Problem smo rešili z iterativno metodo k-means. Ta začne s parametrom $k = 2$. V vsaki ponovitvi najprej poženemo metodo k-means, izberemo predstavnike gruč, jih uredimo glede na $\overline{P}$ in dodamo v seznam (projekcijo preskočimo, če je že v seznamu). Povečamo k in ponavljamo dokler ni $k > |\mathcal{M}|$.
- Hierarhično razvrščanje z različnimi povezovalnimi funkcijami: s povezovanjem srednjih razdalj (angl. *aveage linkage*), najkrajše razdalje (angl. *single linkage*) in najdaljše razdalje (angl. *complete linkage*). Projekcije uredimo podobno kot

pri iterativni k-means metodi. Namesto da ponovimo algoritem za razvrščanje, vzamemo k najbolj zgornjih gruč.

Metoda rangiranja vizualizacij z *zemljevidom modelov* vrne urejen seznam projekcij – predstavnikov gruč $\mathcal{M}'_i$.

Uspešnost metode smo ocenili na 11 zbirkah podatkov. Poleg štirih v tabeli D.2 (podatkov Iris nismo uporabili, ker vsebujejo le štiri značilke in je zato vseh možnih projekcij malo) smo metodo testirali še na sedmih zbirkah podatkov iz tabele D.3.

Tabela D.3: Lastnosti sedmih dodatnih naborov podatkov. Šest jih je iz zbirke UCI Machine Learning Repository, podatki Marketing pa so iz enega od primerov v knjigi Hastieja in avtorjev [57].

	Primeri	Diskretne značilke	Zvezne značilke	Vrednosti razreda
Adult	32,561	8	6	2
Glass	214	0	9	10
Marketing	8,993	13	0	9
Mushroom	8,124	22	0	7
Primary Tumor	339	17	0	21
Vehicle	846	0	18	4
WDBC[1]	569	0	20	2

Najprej primerjamo koliko nove informacije dobimo s prvimi i ($i \in \{1, 2, \ldots, 15\}$) projekcijami v seznamu obeh metod (slike 5.1, 5.5 in 5.7). Predpostavimo, da podobne projekcije podobno ločujejo razrede, zato bo njihova skupna informacija manjša. Ker smo projekcije "zavili" v napovedni model, lahko izračunamo entropijo napovedanega razreda in jo primerjamo s skupno entropijo več projekcij:

$$H(P_1, \ldots, P_i) = -\sum_{c_1} \ldots \sum_{c_i} P(c_1, \ldots, c_i) \log_2 \big(P(c_1, \ldots, c_i)\big), \tag{D.19}$$

kjer je c_i napovedani razred pri projekciji P_k in $P(c_1, \ldots, c_i)$ verjetnost, da se vrednosti teh razredov napovejo hkrati. Na primer, če sta projekciji P_1 in P_2 podobni in imata enake napovedi, je njuna entropija enaka njuni skupni entropiji:

$$H(P_1) = H(P_2) = H(P_1, P_2). \tag{D.20}$$

Nasprotno, za popolnoma različni projekciji P_1 in P_2 je njuna skupna entropija enaka vsoti njunih entropij:

$$H(P_1) + H(P_2) = H(P_1, P_2). \tag{D.21}$$

Pokazali smo, da entropija projekcij, ki jih vrne metoda na osnovi *zemljevida modelov*, narašča hitreje. Iz tega lahko sklepamo, da daje naša metoda bolj raznolike projekcije kot metoda VizRank.

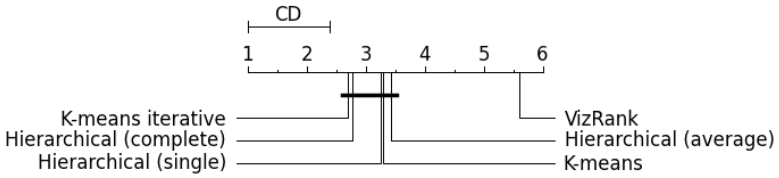
[1] Wisconsin Data on Breast Cancer

Zanimalo nas je ali nam ta različnost prinese več znanja o razredni spremenljivki. Opazovali smo, kako projekcije P_i ($i \in \{1, 2, \ldots, 15\}$) zmanjšajo entropijo razreda:

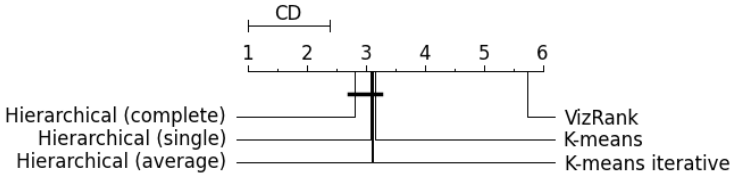
$$H(Y|P_1, \ldots, P_i) = H(Y, P_1, \ldots, P_i) - H(P_1, \ldots, P_i), \tag{D.22}$$

kjer je $H(Y, P_1, \ldots, P_i)$ skupna entropija originalnega razreda in napovedanih razredov, ki jo izračunamo po enačbi D.19. Na slikah 5.2, 5.6 in 5.8 vidimo, da projekcije, ki jih vrne metoda osnovi razdalje med modeli, povejo več tudi o razredu kot enako število projekcij pri metodi VizRank.

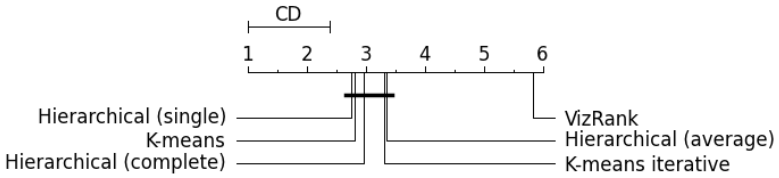
Nazadnje smo rezultate potrdili še s Friedmanovim statističnim testom χ_F^2 [43, 44], ki je pokazal, da je naša metoda značilno boljša od metode VizRank. Rezultati so prikazani v tabeli D.4 in z diagrami kritičnih razdalj [31] na sliki D.5.



(a) Upoštevaje prvih 5 projekcij.



(b) Upoštevaje prvih 10 projekcij.



(c) Upoštevaje prvih 15 projekcij.

Slika D.5: Primerjava 6 metod za rangiranje projekcij z Nemenyijevim [88] testom. Metode, med katerimi ni značilnih razlik, so povezane z daljico.

Tabela D.4: Rezultati Friedmanovega testa. V 6 vzorcev smo združili pogojne entropije $H(Y|P_1, \ldots, P_i)$ iz tabel 5.3, 5.5, in 5.6. Vsak vzorec ima 33 (3×11) vrednosti. Primerjali smo za $i \in \{5, 10, 15\}$.

	χ_F^2	p-vrednost
$P_1 ... P_5$	$50,668$	1.0×10^{-9}
$P_1 ... P_{10}$	$54,141$	2.0×10^{-10}
$P_1 ... P_{15}$	$67,024$	4.3×10^{-13}

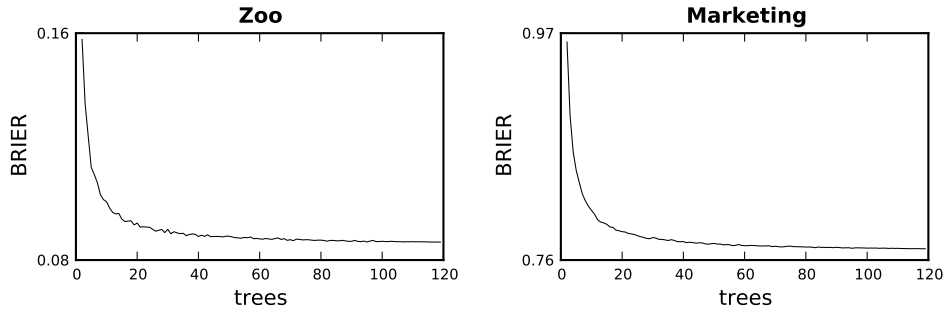
D.6 Raziskovanje kombiniranih učnih algoritmov

Učenje kombiniranih napovednih modelov je področje, ki raziskuje združevanje preprostih napovednih modelov. Večjo skupino šibkih napovednih modelov želimo združiti v enovit napovedni algoritem. Kombinirani napovedni modeli so priljubljeni, ker izredno dobro napovedujejo neznane primere. Njihova velika pomanjkljivost je pomanjkanje metod za vizualizacijo in zahtevna ali celo nemogoča razložljivost.

Zemljevid modelov smo uporabili za vizualizacijo in raziskovanje naključnih gozdov[18]. Ti so sestavljeni iz večjega števila napovednih dreves, ki so običajno majhna (vsebujejo do 10 vozlišč). Raziskali smo možnost uporabe omrežja modelov za učenje gozdov napovednih dreves. Različni deli omrežja napovednih modelov predstavljajo različne poglede na podatke, zato domnevamo, da napovedni modeli, izbrani iz različnih področij omrežja, tvorijo dober sestavljen model.

D.6.1 Ponazoritev naključnih gozdov

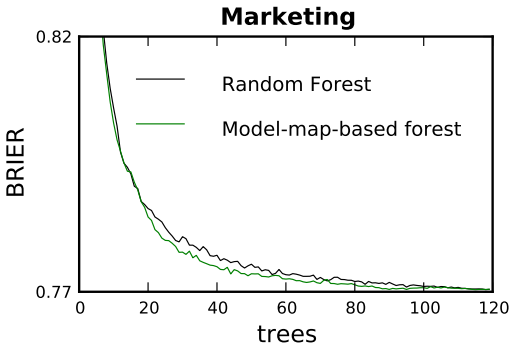
Zemljevid modelov ponazarja posamezna drevesa v naključnem gozdu. Ogledamo si lahko posamezna drevesa ali različna področja naključnega gozda. V disertaciji smo raziskovali dva primera naključnega gozda na podatkih Zoo in Marketing. Polovico podatkov smo uporabili za učenje modela, na drugi polovici pa smo izračunali razdalje med drevesi. V obeh primerih smo zgradili gozd s stotimi drevesi, ki zadoščajo za optimalno napoved (slika D.6). Kvaliteto napovedi smo ocenili z Brierjevo oceno. V omrežju je vsako drevo povezano z najbolj podobnim.



(a) podatki Zoo (b) podatki Marketing

Slika D.6: Brierjeva ocena (izboljšuje se s številom dreves) naključnega gozda.

Izdelali smo paket gradnikov Orange Model Maps za raziskovanje *zemljevida modelov*. Premikanje miške med drevesi v gradniku Model Map označi drevesa v bližini. Gradnik pokaže povzetke kvalitete njihovih napovedi za posamezen razred in vpliv atributov v označenih drevesih. Model lahko raziskujemo in tudi ročno spreminjamo. Izberemo lahko množico dreves, ki jih analiziramo posebej v drugih gradnikih v okolju Orange. Posamezna drevesa lahko tudi ročno dodajamo ali odvzemamo iz gozda in še bolj izboljšamo napovedno točnost našega modela.



Slika D.7: Brierjeva ocena naključnih gozdov in gozdov na osnovi razdalje med drevesi glede na število dreves v gozdu. Za vsako število dreves smo naučili 500 gozdov in prikazali povprečno oceno.

D.6.2 Gozd iz omrežja modelov

Na primeru podatkov Marketing smo opazili zanimivo značilnost. Če sestavimo gozd z dvakrat več drevesi kot jih potrebujemo, se nova drevesa pridružijo že obstoječim gručam. Sklepali smo, da bi lahko gozd gradili hitreje, če bi izbirali "pravilna" drevesa, torej predstavnike posameznih gruč. Raziskali smo možnost takšne hevristike.

Gozd smo gradili na dva načina: z algoritmom naključnih gozdov in z algoritmom gozdov na osnovi razdalj med drevesi. Izkaže se (slika D.7), da bi na teh in podobnih podatkih lahko uporabljali hevristiko za gradnjo gozda z *zemljevidom modelov*. Našo domnevo smo potrdili s Wilcoxonovim testom [116] (tabeli D.5 in D.6).

Metodo smo preizkusili tudi na več drugih zbirkah podatkov, ki smo jih že opisali v tabelah D.2 in D.3, kjer pa smo dobili slabše rezultate. Metoda se je dobro obnesla še na enem tipu podatkov (WDBC), na štirih je bila konkurenčna (Breast Cancer Wisconsin, Glass, Primary Tumor in Vehicle), pri ostalih pa slabša. V prihodnjem delu bi bilo zanimivo raziskati, katere karakteristike podatkov vplivajo na uspešnost naše metode.

Tabela D.5: Povprečna ranga naključnih gozdov in gozdov na osnovi razdalje med drevesi (na podatkih Marketing).

	rang
naključni gozd	1.746
gozd z *zemljevidom modelov*	1.254

Tabela D.6: Rezultat Wilcoxonovega testa. Primerjali smo povprečno Brierjevo oceno naključnih gozdov in gozdov na osnovi razdalje med drevesi, ki smo jih naučili pri različnem številu dreves (na podatkih Marketing).

	z	p-value
Marketing	1417.0	1.89×10^{-8}

D.7 Zaključek

Kopičenje podatkov je povzročilo revolucijo na področju obdelave in učenja iz podatkov. Strokovnjaki potrebujejo učinkovita orodja za preiskovanje, vizualizacijo in odkrivanje skritih vzorcev. Vizualizacija informacij in vizualna analiza podatkov sta postali živahni raziskovalni področji, saj omogočata interaktivno raziskovanje podatkov in odkrivanje skritega znanja.

Namenski algoritmi za vizualizacijo, ki so dovolj učinkoviti za gradnjo interaktivnih in odzivnih vmesnikov, so pomembni za učinkovito raziskovalno analizo podatkov. Predstavili smo algoritem za optimizacijo ponazoritve omrežij FragViz, ki izpolnjuje omenjene zahteve. Primeren je predvsem za vizualizacijo razdrobljenih mrež, kjer običajni algoritmi ne upoštevajo podobnosti med nepovezanimi deli omrežja.

V primerjavi z obstoječimi metodami FragViz ni niti najhitrejši niti najnatančnejši v smislu razlike med podanimi razdaljami in pozicijo točk. Je počasnejši od algoritma Fructherman-Reingold, ker pri izrisu upošteva več informacij. Končni položaj točk tudi slabše odraža matriko podobnosti med njimi, kot pri metodi večdimenzionalnega lestvičenja, kar je posledica vnaprej določene postavitve posameznih komponent. Oboje je posledica predpostavke, da naj metoda FragViz nudi jasno lokalno sliko in globalni pregled nad omrežjem.

Na potrebo po novih interaktivnih orodjih s preprostim uporabniškim vmesnikom smo odgovorili z orodjem Orange Network. Naše orodje ima poleg funkcionalnosti, ki jih ponujajo obstoječa orodja tudi metode za lokalno raziskovanje omrežij: povečava dela grafa, uporaba lastnosti omrežja za izbor podatkov, opazovanje razvoja omrežja v času, analizo omrežja z algoritmi za podatkovno rudarjenje in druge. Glavna prednost pa je povezava z drugimi gradniki v orodju Orange, kar omogoča povezovanje standardnih analitičnih metod z metodami za analizo omrežja.

Razvili smo metodologijo za izgradnjo omrežja napovednih modelov, t.i. *zemljevida modelov*. Ključna je ideja, da sta dva napovedna modela podobna takrat, ko dajeta podobne napovedi in ne, če sta zgolj videti podobno (na primer, oba sta odločitveni drevesi ali pa sta zgrajena nad istimi značilkami). Na ta način lahko primerjamo raznovrstne napovedne modele in jih predstavimo v enem omrežju.

Napovedi modelov so naključne spremenljivke (diskretne, ko napovedujemo razred ali zvezne, ko napovedujemo verjetnost razreda). Če želimo primerjati dva modela, moramo torej primerjati dve naključni spremenljivki. Predlagali smo pet metrik

razdalje med napovednimi modeli, med katerimi nismo našli značilnih razlik. Že sama ideja, da za primerjavo uporabimo napovedi modelov, je dovolj robustna. V primerih uporabe smo uporabili manhattansko razdaljo zaradi najboljše časovne zahtevnosti in ker je primernejša za ponazoritev točk v evklidski ravnini kot tiste, ki temeljijo na Kullback-Leiblerjevi razdalji.

Predstavili smo dva primera uporabe *zemljevida modelov*. Prvi je metoda za iskanje zanimivih projekcij podatkov, ki poleg kvalitete projekcij upošteva tudi njihovo raznolikost. Našo metoda nam pomaga, da o problemski domeni hitreje izvemo več informacij kot z metodo VizRank. Nudi nam celo več informacij tako o podatkih kot tudi o razredu.

Predstavili smo zanimiv primer uporabe *zemljevida modelov* za vizualizacijo projekcij z omrežjem. Domenski strokovnjak lahko raziskuje zanimivih vizualizacije, ki so povezane v omrežje, namesto da se sprehaja po seznamu. Uporabnost takšnega načina raziskovalne analitike bomo raziskali v nadaljnjem delu.

Drugi primer uporabe je metodologija za izdelavo in raziskovanje omrežja napovednih dreves v naključnih gozdovih. Običajno metode algoritme, ki kombinirajo napovedne modele, vizualizirajo ali razlagajo posredno. Obravnavajo jih kot "črne škatle". Naša metoda omogoča, da si zgradbo naključnega gozda (dreves in podobnosti med njimi) ogledamo neposredno. Prikaže lahko razmerja med posameznimi drevesi. Različna področja prostora modelov (skupine modelov) lahko interaktivno raziskujemo z orodjem Orange Model Maps.

Poleg pregledovanja takšnega kombiniranega modela lahko igramo aktivno vlogo pri njegovi konstrukciji. Izbiramo lahko področja dreves in jih pošljemo v nadaljnjo analizo v druge gradnike. Drevesa lahko izločimo iz gozda ali jih vanj dodamo ter algoritem učenja še bolj prilagodimo naši domeni. Raziskave na tem področju predlagamo kot eno izmed možnih smeri za nadaljnje delo.

Predlagali smo tudi potencialno hevristiko za gradnjo naključnih gozdov. Primerjali smo hitrost učenja dveh algoritmov (naključnega gozda in gozda na osnovi *zemljevida modelov*). Na podatkih Marketing smo primerjali hitrost učenja in dobili značilno boljše rezultate. Sklepamo, da bi lahko pohitrili učenje naključnih gozdov, če bi poznali zakone, ki veljajo v prostoru modelov. Raziskave podatkovnih domen, na katerih bi bila predlagana hevristika uporabna, bodo predmet nadaljnjega dela.

Orange (`http://orange.biolab.si`), Orange Network, Orange Model Maps in druge potrebne knjižnice so na voljo pod licenco GNU GPL za MS Windows, Linux in Mac OS X operacijske sisteme. Izvorna koda, omrežja in podatki, ki smo jih uporabili v primerih uporabe, so vključeni v orodje Orange.

D.7.1 Prispevki k znanosti

Povzetek prispevkov k znanosti:

- Metoda za vizualizacijo omrežij – FragViz; primerna je predvsem za razdrobljena omrežja, ki vsebujejo veliko majhnih komponent.
- Definicija novih mer podobnosti med napovednimi modeli, ki so primerne za njihovo analizo v obliki omrežij.
- Metoda za analizo in interaktivno preiskovanje omrežij napovednih modelov.
- Razširitev metode VizRank, ki temelji na mreži namesto na seznamu vizualizacij, in poleg kvalitete vsebuje tudi informacijo o različnosti projekcij.
- Metodologija za vizualizacijo zbirk napovednih modelov (angl. *ensemble of classifiers*). Poleg vizualizacije splošnih zbirk napovednih modelov bomo posebno pozornost posvetili vizualizaciji naključnih gozdov.

Izjava

Izjavljam, da sem doktorsko disertacijo izdelal samostojno pod mentorstvom izr. prof. dr. Janeza Demšarja. Sodelavce, ki so mi pomagali pri nalogi, sem navedel v razdelku "Acknowledgements" na strani vii.

Ljubljana, junij 2012 Miha Štajdohar

Bibliography

[1] G. J. J. Adeva, U. B. Cervino, and R. A. Calvo. Accuracy and diversity in ensembles of text categorisers. *CLEI Electronic Journal*, 9(1), 2005.

[2] D. Aha. Generalizing from case studies: A case study. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 1–10. Citeseer, 1992.

[3] B. Andreas and D. Asimov. Grand tour methods: an outline. *Computing Science and Statistics*, 17:63–67, 1986.

[4] D. Ascher, P. F. Dubois, K. Hinsen, J. Hugunin, and T. Oliphant. Numerical python. *Available via the World Wide Web*, 2001.

[5] D. Asimov. The Grand Tour: A Tool for Viewing Multidimensional Data. *SIAM Journal on Scientific and Statistical Computing*, 6(1):128, 1985.

[6] E. Bair, T. Hastie, D. Paul, and R. Tibshirani. Prediction by Supervised Principal Components. *Journal of the American Statistical Association*, 101(473):119–137, 2006.

[7] A. Barabási. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, 1999.

[8] M. Bastian, S. Heymann, and M. Jacomy. Gephi: An open source software for exploring and manipulating networks. In *ICWSM*, 2009.

[9] N. N. Batada, T. Reguly, A. Breitkreutz, L. Boucher, B.-J. Breitkreutz, L. D. Hurst, and M. Tyers. Stratus not altocumulus: a new view of the yeast protein interaction network. *PLoS biology*, 4(10):e317, 2006.

[10] V. Batagelj and A. Mrvar. Pajek - program for large network analysis. *Connections*, 21:47–57, 1998.

[11] M. Baur and U. Brandes. Crossing reduction in circular layouts. In *WG*, pages 332–343, 2004.

[12] S. K. Bhavnani, F. Eichinger, S. Martini, P. Saxman, H. V. Jagadish, and M. Kretzler. Network analysis of genes regulated in renal diseases: implications for a molecular-based classification. *BMC bioinformatics*, 10 Suppl 9:S3, 2009.

[13] N. L. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory 1736-1936*. Oxford University Press, USA, 1999.

[14] J. A. Blake, J. C. Matese, S. S. Dwight, H. Butler, A. P. Davis, J. M. Cherry, K. Dolinski, L. Issel-Tarver, J. T. Eppig, M. Ashburner, G. M. Rubin, S. Lewis, M. Ringwald, M. A. Harris, D. P. Hill, A. Kasarskis, C. A. Ball, J. E. Richardson, D. Botstein, and G. Sherlock. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat Genet*, 25(1):25–29, 2000.

[15] U. Brandes. Eigensolver methods for progressive multidimensional scaling of large data. *Graph Drawing*, 2007.

[16] P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta. *Metalearning: Applications to data mining*. Springer-Verlag New York Inc, 2009.

[17] L. Breiman. Bagging predictors. In *Machine Learning*, volume 24, pages 123–140, 1996.

[18] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[19] G. W. Brier. Verification of Forecasts Expressed in Terms of Probability. *Monthly Weather Review*, 78(1):1–3, 1950.

[20] G. Brown, J. Wyatt, R. Harris, and X. Yao. Diversity creation methods: A survey and categorisation. *Journal of Information Fusion*, 6:5–20, 2005.

[21] R. Caruana. Multitask Learning. *Machine Learning - Special issue on inductive transfer*, 28(1), 1997.

[22] O. Celma. *Music Recommendation and Discovery*. Springer, 2010.

[23] J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. Tukey. *Graphical Methods for Data Analysis*. Wadsworth; figures 2.8, 4.22, 5.4., 1983.

[24] W. S. Cleveland. *Visualizing Data*. Hobart Press, 1993.

[25] G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems, 2006.

[26] D. R. Cutler, T. C. Edwards, K. H. Beard, A. Cutler, K. T. Hess, J. Gibson, and J. J. Lawler. Random forests for classification in ecology. *Ecology*, 88(11):2783–2792, 2007.

[27] Cyram. *NetMiner User Manual*. Seoul: Cyram, 2003.

[28] B. V. Dasarathy. *Nearest neighbor (NN) norms : NN pattern classification techniques*. IEEE Computer Society Press, 1991.

[29] W. de Nooy, A. Mrvar, and V. Batagelj. *Exploratory Social Network Analysis with Pajek (Structural Analysis in the Social Sciences)*. Cambridge University Press, 2005.

[30] J. Demsar, B. Zupan, and G. Leban. Orange: From experimental machine learning to interactive data mining. Faculty of Computer and Information Science, University of Ljubljana, 2004.

[31] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, 2006.

[32] J. Demšar, B. Zupan, G. Leban, and T. Curk. Orange: From experimental machine learning to interactive data mining. In *Knowledge Discovery in Databases: PKDD 2004*, pages 537–539. Springer, 2004.

[33] I. S. Dhillon, D. S. Modha, and W. S. Spangler. Visualizing class structure of multidimensional data. In *Proceedings of the 30th Symposium on the Interface: Computing Science and Statistics*, pages 488—-493, 1998.

[34] J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002.

[35] P. Eades. A Heuristic for Graph Drawing. *Congressus Numerantium*, 42:149 – 160, 1984.

[36] P. Eades and M. Huang. Navigating clustered graphs using force-directed methods. *Journal of Graph Algorithms and Applications*, 4(3):157–181, 2000.

[37] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull. Graphviz - open source graph drawing tools. *Graph Drawing*, pages 483–484, 2001.

[38] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Human Genetics*, 7(2):179–188, 1936.

[39] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2010.

[40] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, EuroCOLT '95, pages 23–37. Springer-Verlag, 1995.

[41] A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In *Graph Drawing*, pages 388–403. Springer-Verlag, 1995.

[42] J. Friedman and J. Tukey. A Projection Pursuit Algorithm for Exploratory Data Analysis. *IEEE Transactions on Computers*, C-23(9):881–890, 1974.

[43] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

[44] M. Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92, 1940.

[45] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software- Practice and Experience*, 21(11):1129–1164, 1991.

[46] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.

[47] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(12):7821–6, 2002.

[48] K. Goh, M. Cusick, D. Valle, B. Childs, M. Vidal, and A.-L. Barabási. The human disease network. *Proceedings of the National Academy of Sciences*, 104(21):8685, 2007.

[49] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science (New York, N.Y.)*, 286(5439):531–7, 1999.

[50] D. F. Gordon and M. Desjardins. Evaluation and selection of biases in machine learning. *Machine Learning*, 20(1-2):5–22, 1995.

[51] A. Hagberg, D. Schult, and P. Swart. Networkx. high productivity software for complex networks, 2006.

[52] D. Hanahan and R. A. Weinberg. The hallmarks of cancer. *Cell*, 100(1):57–70, 2000.

[53] M. S. Handcock, D. R. Hunter, C. T. Butts, S. M. Goodreau, and M. Morris. *statnet: Software tools for the Statistical Modeling of Network Data*. Seattle, WA, 2003. Version 2.0.

[54] T. L. Hankins. Blood, dirt, and nomograms: A particular history of graphs. *Isis*, 90(1):50–80, 1999.

[55] R. Harris. *Information graphics: A comprehensive illustrated reference*. Oxford University Press, USA, 1999.

[56] R. L. Harris. *Information Graphics: A Comprehensive Illustrated Reference*. Oxford University Press, USA, 1 edition, 2000.

[57] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, corrected edition, 2003.

[58] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2003.

[59] M. Himsolt. GML: A portable Graph File Format. *Syntax*, 1996.

[60] J. A. Hoeting, D. Madigan, A. E. Raftery, and C. T. Volinsky. Bayesian model averaging: A tutorial. *Statistical Science*, 14(4):382–401, 1999.

[61] P. Hoffman, G. Grinstein, K. Marx, I. Grosse, and E. Stanley. *DNA visual and analytic data mining*. IEEE, 1997.

[62] J. Holland. Adaptation in natural and artificial systems. *MIT Press*, 1992.

[63] Y. Hu. Efficient and high quality force-directed graph drawing. *Mathematica Journal*, 10:37–71, 2005.

[64] C. Huttenhower, E. Haley, M. Hibbs, V. Dumeaux, D. Barrett, H. Coller, and O. Troyanskaya. Exploring the human genome with functional maps. *Genome research*, 19(6):1093, 2009.

[65] F. Iorio, R. Tagliaferri, and D. di Bernardo. Identifying network of drug mode of action by gene expression profiling. *Journal of computational biology : a journal of computational molecular cell biology*, 16(2):241–51, 2009.

[66] A. Jakulin, M. Možina, J. Demšar, I. Bratko, and B. Zupan. Nomograms for visualizing support vector machines. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 108–117. ACM, 2005.

[67] E. Jones, T. Oliphant, and P. Peterson. Scipy: Open source scientific tools for python, 2001.

[68] D. Jowett. Analysis of variance in complex experimental designs (harold r. lindman). *SIAM Review*, 18(1):134–137, 1976.

[69] P. Kaaijk, G. J. L. Kaspers, E. R. Van Wering, G. J. Broekema, A. H. Loonen, K. Hählen, K. Schmiegelow, G. E. Janka-Schaub, G. Henze, U. Creutzig, and A. J. P. Veerman. Cell proliferation is related to in vitro drug resistance in childhood acute leukaemia. *British journal of cancer*, 88(5):775–81, 2003.

[70] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[71] M. Kearns. Thoughts on hypothesis boosting. *Unpublished manuscript*, 1988.

[72] D. Keim. Visualization techniques for mining large databases: A comparison. *Knowledge and Data Engineering,*, 8(6):923–938, 2002.

[73] T. S. Keshava Prasad, R. Goel, K. Kandasamy, S. Keerthikumar, S. Kumar, S. Mathivanan, D. Telikicherla, R. Raju, B. Shafreen, A. Venugopal, L. Balakrishnan, A. Marimuthu, S. Banerjee, D. S. Somanathan, A. Sebastian, S. Rani, S. Ray, C. J. Harrys Kishore, S. Kanth, M. Ahmed, M. K. Kashyap, R. Mohmood, Y. L. Ramachandra, V. Krishna, B. A. Rahiman, S. Mohan, P. Ranganathan, S. Ramabadran, R. Chaerkady, and A. Pandey. Human Protein Reference Database–2009 update. *Nucleic acids research*, 37(Database issue):D767–72, 2009.

[74] T. Kohonen, P. Lehtio, J. Rovamo, J. Hyvarinen, K. Bry, and L. Vainio. A principle of neural associative memory. *Neurosci*, 2(6):1065–1076, 1977.

[75] I. Kononenko. Estimating attributes: Analysis and extensions of Relief. In F. Bergadano and L. D. Raedt, editors, *Proceedings of the European Conference on Machine Learning (ECML-94)*, pages 171–182. Springer-Verlag, 1994.

[76] I. Kononenko and M. Kukar. *Machine Learning and Data Mining*. Woodhead Publishing, 2007.

[77] J. Kruskal. Toward a practical method which helps uncover the structure of a set of observations by finding the line tranformation which optimizes a new "index of condensation". In R. C. MILTON and J. A. Nelder, editors, *Statistical Computation*, pages 427–440, New York, 1969. Academic Press.

[78] J. Kruskal. Linear transformation of multivariate data to reveal clustering. In R. N. Shepard, A. K. Romney, and S. B. Nerlove, editors, *Multidimensional scaling: Theory and Applications in the Behavioural Sciences*, volume 1, pages 179–191. Seminar Press, London, 1972.

[79] L. I. Kuncheva and C. J. Whitaker. Measures of diversity in classifier ensembles. *Machine Learning*, 51(2), 2000.

[80] P. Lanzi and W. Stolzmann. Learning Classifier Systems. From Foundations to Applications. *Lecture Notes in Computer Science*, 1813, 2000.

[81] G. Leban, I. Bratko, U. Petrovic, T. Curk, and B. Zupan. VizRank: finding informative data projections in functional genomics by machine learning. *Bioinformatics (Oxford, England)*, 21(3):413–4, 2005.

[82] G. Leban, B. Zupan, G. Vidmar, and I. Bratko. VizRank: Data Visualization Guided by Machine Learning. *Data Mining and Knowledge Discovery*, 13(2):119–136, 2006.

[83] I. Leung, P. Hui, P. Liò, and J. Crowcroft. Towards real-time community detection in large networks. *Physical Review E*, 79(6):1–10, June 2009.

[84] P. Mair and J. De Leeuw. Multidimensional Scaling Using Majorization: SMACOF in R. *Journal of Statistical Software*, 31(03), 2009.

[85] T. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[86] T. M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997.

[87] M. Možina, J. Demšar, M. W. Kattan, and B. Zupan. Nomograms for visualization of naive bayesian classifier. In *PKDD*, pages 337–348, 2004.

[88] P. Nemenyi. *Distribution-free multiple comparisons*. Docotoral Thesis. Princeton University, 1963.

[89] T. Nepssz. Reconstructing the structure of the world-wide music scene with Last.fm, 2009.

[90] M. Newman. The physics of networks. *Physics Today*, pages 33–38, 2008.

[91] M. Newman. *Networks: An Introduction*. Oxford University Press, USA, 2010.

[92] NWB Team. *Network Workbench Tool*. Indiana University, Northeastern University and University of Michigan, 2006.

[93] V. U. Onay, L. Briollais, J. A. Knight, E. Shi, Y. Wang, S. Wells, H. Li, I. Rajendram, I. L. Andrulis, and H. Ozcelik. SNP-SNP interactions in breast cancer susceptibility. *BMC cancer*, 6:114, 2006.

[94] P. Pagel, S. Kovac, M. Oesterheld, B. Brauner, I. Dunger-Kaltenbach, G. Frishman, C. Montrone, P. Mark, V. Stumpflen, H.-W. Mewes, A. Ruepp, and D. Frishman. The MIPS mammalian protein-protein interaction database. *Bioinformatics (Oxford, England)*, 21(6):832–4, 2005.

[95] A. Paulovich, S. Mukherjee, B. L. Ebert, J. P. Mesirov, A. Subramanian, S. L. Pomeroy, M. A. Gillette, E. S. Lander, P. Tamayo, T. R. Golub, and V. K. Mootha. From the Cover: Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. *PNAS*, 102(43):15545–15550, 2005.

[96] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[97] C.-H. Pui and W. Evans. Treatment of acute lymphoblastic leukemia. *The New England journal of medicine*, 354(2):166, 2006.

[98] A. E. Raftery. Choosing models for cross-classifications. *American Sociological Review*, 51(1):145–146, 1986.

[99] U. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3), 2007.

[100] L. Rendell, R. Seshu, and D. Tcheng. Layered concept learning and dynamically-variable bias management. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI'87)*, pages 308–314. Citeseer, 1987.

[101] C. G. J. Saris, S. Horvath, P. W. J. van Vught, M. A. van Es, H. M. Blauw, T. F. Fuller, P. Langfelder, J. DeYoung, J. H. J. Wokke, J. H. Veldink, L. H. van den Berg, and R. A. Ophoff. Weighted gene co-expression network analysis of the peripheral blood from Amyotrophic Lateral Sclerosis patients. *BMC genomics*, 10:405, 2009.

[102] E. M. Scholar and P. Calabresi. Identification of the enzymatic pathways of nucleotide metabolism in human lymphocytes and leukemia cells. *Cancer research*, 33(1):94–103, 1973.

[103] R. Sokal and C. Michener. A statistical method for evaluating systematic relationships. *Multivariate statistical methods, among-groups covariation*, 28:269, 1975.

[104] P. Sollich and A. Krogh. Learning with ensembles: How overfitting can be useful. *Advances in Neural Information Processing Systems*, 8:190–196, 1996.

[105] C. Spearman. "general intelligence," objectively determined and measured. *American Journal of Psychology*, 15:201–293, 1904.

[106] C. Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 15:72–101, 1904.

[107] M. Stajdohar, M. Mramor, B. Zupan, and J. Demšar. FragViz: visualization of fragmented networks. *BMC bioinformatics*, 11:475, 2010.

[108] E. Strumbelj and I. Kononenko. An efficient explanation of individual classifications using game theory. *J. Mach. Learn. Res.*, 11:1–18, 2010.

[109] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, 2005.

[110] S. Thrun and P. L. Y. *Learning To Learn*. Kluwer Academic Publishers, Boston, MA, 1998.

[111] J. W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[112] R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002.

[113] E. Štrumbelj, I. Kononenko, and M. Robnik Šikonja. Explaining instance classifications with interactions of subsets of feature values. *Data Knowl. Eng.*, 68(10):886–904, 2009.

[114] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–2, 1998.

[115] D. M. White, A. G. Smith, and J. L. Smith. Assessment of proliferative activity in leukaemic bone marrow using the monoclonal antibody Ki-67. *Journal of clinical pathology*, 47(3):209–13, 1994.

[116] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[117] I. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[118] D. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.