Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Strojno učenje s simplicialnimi kompleksi

Aleks Jakulin

Delo je pripravljeno pod mentorstvom
doc. dr. Blaž Zupan
doc. dr. Neža Mramor-Kosta

Sežana, 1999

Povzetek

Cilj tega besedila je prikazati in upravičiti uporabo matematične strukture simplicialnih kompleksov v strojnem učenju. V tem okviru predstavimo metodo strojnega učenja, ki hkrati kombinira lokalnost in nehierarhičnost in ki lahko učinkovito deluje z zveznimi atributi. Strojno učenje hipotez, predstavljenih s simplicialnimi kompleksi, izpolnjuje te pogoje in zato odpira nekatera nova področja uporabe, na primer pomoč pri vizualizaciji podatkov. Ker so simplicialni kompleksi matematični objekti, v besedilu predstavimo matematično ozadje, ki sega v topologijo, linearno algebro in do neke mere v računsko geometrijo. V nadaljevanju opišemo osnovne postopke za izgradnjo klasifikacijskih hipotez in klasifikacijo ter značilnosti same implementacije. Metodo je mogoče prilagoditi za obravnavanje šuma ter jo integrirati z drugimi metodami, na primer z dekompozicijo. Prikažemo tudi načine pohitritve uporabljenih algoritmov. Končamo s kratkim pregledom prednosti in slabosti postopka. V dodatku predstavimo še aplikacijo CCW, ki uporabniku omogoča enostavno vizualno delo s klasifikatorji na podlagi simplicialnih kompleksov v dveh dimenzijah, kar ilustriramo z več primeri in s preizkusom uspešnosti.

Zahvala

Zahvaljujem se mentorju doc. dr. Blažu Zupanu za več kot dve leti spodbude in vodenja pri konkretizaciji in realizaciji mojih meglenih idej ter somentorici doc. dr. Neži Mramor-Kosta za veliko pomoči pri soočenju z matematičnimi orodji ter terminologijo. Tudi mag. Janez Demšar mi je dostikrat prijazno dal marsikateri izredno koristen nasvet in komentar. Hvaležen sem prof. dr. Igorju Kononenku ter prof. dr. Ivanu Bratku za odlična predavanja in pogovore ter za izredno aktivnost, saj sta me že pred leti s svojim delom in publikacijami navdušila za področje umetne inteligence.

Sežana, oktober 1999 Aleks Jakulin

Kazalo

1. Uvod

V Uvodu so predstavljeni osnovni koncepti strojnega učenja, ki jih mora učinkovita metoda izpolnjevati. Osredotočili se bomo na probleme, ki se jih da smiselno predstaviti kot točke v prostoru atributov, ter orodja in tehnike, ki jih uporabljajo ljudje pri obvladovanju takih problemov. Poznavalec strojnega učenja bi mogoče hotel preskočiti kar k Izzivu, kjer si bomo zastavili željeni cilj. Terminologija in koncepti v tem razdelku izhajajo večinoma iz [50, 46, 67].

1.1. Strojno učenje

Strojno učenje je zelo širok pojem, a v tem besedilu se bomo omejili na učenje klasifikatorjev na podlagi učnih primerov podanih z atributi. **Klasifikator** ali model za razvrščanje na podlagi nekega vektorja vrednosti atributov določi kategorični **razred**. Na klasifikator lahko gledamo tudi kot na sinhron **sistem**, ki na vhodu sprejema vektor vrednosti **atributov** ter na izhod daje vrednost **razreda**. Atributi ustrezajo neodvisnim spremenljivkam, ali značilkam *(features)*, vrednost razreda pa je odvisna spremenljivka, če gledamo na problem klasifikacije kot na enačbo. Cilj učenja je, da zgradimo klasifikacijski sistem tako, da posnema sistem, ki je podlaga same **domene**. Ponavadi je bolj natančno in učinkovito oponašanje tudi bolj uspešno. Problem imenujemo **regresijski**, ko je množica vrednosti razreda kvantitativna ali zvezna. Učenju na klasifikacijskih domenah pravimo klasifikacijsko učenje *(classification learning)* ali tudi prepoznavanje vzorcev *(pattern recognition)*, učenju regresijskih domen pa regresijsko učenje *(regression* ali *mapping learning)*.

Da bi razjasnili terminologijo, si poglejmo primer diagnosticiranja bolnika. Domena je ugotavljanje, ali ima nek bolnik gripo. Sistem je telo bolnika, ki deluje v interakciji z okoljem. Idealen adaptiven sistem bi bil simulacija celega telesa s pospešenim ali upočasnjenim časom in popolnim vpogledom, vendar pa to seveda ni izvedljivo. S strojnim učenjem moramo najti približek te simulacije v obliki sistema, ki obravnava le tiste elemente, ki nas zanimajo. V tem primeru nas zanima prisotnost velikega števila virusov gripe, ki povzroči slabše počutje bolnika. Razreda sta torej dva: ali ima bolnik gripo ali ne. Izhod iz približka sistema je ravno vrednost razreda. Ker bolnika ne moremo razrezati na koščke, natančne preiskave z mikroskopom pa bi bile predrage, pripravimo spisek značilnosti bolnika, ki so enostavno razvidne in jih bomo uporabljali pri sklepanju, to so atributi. Seveda vnaprej še ne vemo, ali bodo vsi atributi tudi koristni. Nekateri atributi so diskretni, na primer spol bolnika, večina pa jih je zveznih, kot recimo telesna temperatura, teža ter število levkocitov v krvi. Učimo se induktivno, torej samo na podlagi izkušenj z več bolniki, kjer smo z natančno preiskavo potrdili, ali so imeli gripo ali ne, ter ugotovili vrednosti njihovih atributov, drugo znanje in podatki pa nam niso na voljo. Tem izkušnjam pravimo **učni primeri**.

Bistvo učenja je torej sposobnost **posploševanja** *(generalization)*, kjer gre za to, da tudi na območjih, kjer nismo videli nobenega učnega primera, čim bolj pravilno določimo razred na osnovi obdelanih učnih primerov. Implicitno je posploševanje tudi posledica izbire atributov, saj tisti atributi, za katere uporabnik algoritma misli, da niso pomembni, niso vključeni v učenje. V praksi posploševanje pomeni, da poskušamo pravilno diagnosticirati tudi bolnika, ki ga še nismo videli. Pri tem moramo paziti, kako močno bomo zaupali svojim hipotezam. V praksi na primer ne moremo

kupiti veliko število delnic podjetja, ki se sicer zdi najboljše, a v to nismo popolnoma prepričani. Po drugi strani pa lahko več vložimo v manj uspešno podjetje, ki pa ga boljše poznamo in je zato naše znanje bolj gotovo. Z drugimi besedami — v neraziskanem območju moramo biti bolj previdni kot pa tam, kjer smo videli nekaj tisoč identičnih učnih primerov in verjamemo, da se težnja tudi v prihodnje ne bo spremenila.

Algoritme za strojno učenje lahko metodološko razdelimo na simbolične in nesimbolične metode. Osnovna razlika je v načinu predstavitve naučenega modela: simbolične metode predstavijo naučeno hipotezo na način, ki je razumljiv tudi človeku, predvsem v smislu uporabe lingvističnih in logičnih izrazov s pravili in odločitvami za izgradnjo hipotez: »Če so na nebu temni oblaki, potem bo deževalo.« Cilj učenja namreč ni nujno le napovedovanje, temveč tudi predstavitev znanja, na primer za izdelavo povzetkov kompliciranih domen, ali pa predstavitev zanimivih zakonitosti v podatkih. Nesimbolične ali numerične metode te zahteve ne izpolnjujejo vedno, ob tem pa se zaradi večje izraznosti svojih modelov bolje obnesejo pri domenah, ki niso učinkovito opisljive z logičnimi pravili, še posebej pri domenah z zveznimi atributi. Čudno bi bilo napovedovati vreme le z logičnimi pravili, kot na primer: »Če je vlažnost 88% in je temperatura 30 stopinj Celzija, ali če je vlažnost 89% in je temperatura 28 stopinj Celzija, potem...« Po drugi strani pa bi težko razumeli, uporabljali ali preverili hipotezo v kakem nenavadnem opisu, kot na primer: »Deževalo bo, če $1.826566 * temp - 9.123476 * \sin(vlaga) + 5.34576 > 0.0$.«

Večina uporab induktivnega strojnega učenja, z izjemo učenja z ojačanjem *(reinforcement learning)* in aktivnega učenja, deluje na podlagi paketov — vsi učni primeri so že na voljo, hipotezo ustvarimo le enkrat, potem pa je ne spreminjamo več. Učenje z ojačanjem pa obstoječo hipotezo sproti dopoljnjuje, z vsakim novim primerom. Da bi bilo to enostavno izvedljivo, uporablja enostavne predstavitve modelov, ki temeljijo na prostoru stanj in prehodov med njimi. Ob tem tudi rešuje problem porazdelitve uspeha na vse korake, ki so prispevali h končnemu uspehu.

Aktivno učenje [21] temelji na sposobnosti postopka za strojno učenje, da sam izbira nove učne primere, iz katerih se bo najboljše učil. Ob tem je važno, da je hipoteza predstavljena na način, ki omogoča učinkovito ocenjevanje uspešnosti klasifikatorja na nekem območju domene, kar je bistvenega pomena za usmerjanje učenja. Pri aktivnem učenju in učenju z ojačanjem je torej bistveno, da model dovoljuje shranjevanje hipotez na način, ki omogoča njihovo nadaljnjo obdelavo z algoritmi. Po drugi strani pa zahteva po človeku razumljivi predstavitvi hipotez omogoča tudi ljudem na podlagi razumevanja nadaljnjo analizo in uporabo hipotez.

Omeniti moramo tudi odpornost proti šumu: včasih se pri merjenju temperature zmotimo, sam termometer pa tudi ni popolnoma natančen, vendar mora zaradi tega klasifikacija na podlagi temperature še vedno čimbolj pravilno delovati. Šum se ne pojavlja le pri odstopanjih v vrednostih atributov, ampak tudi v napačnih vrednostih razreda za nek učni primer. Dober postopek strojnega učenja bo šum upošteval in dosegel boljše rezultate od postopka, ki ga ignorira. Zanemarjanje šuma namreč povzroči preveliko prileganje učnim primerom *(overfitting)*, kar pomeni, da smo šum vključili v hipotezo, namesto da bi ga izločili.

1.2. Modeli in znanje

Model je način razumevanja domene, je jezik s katerim lahko opisujemo znanje. Ko izberemo model, v njegovem okviru opišemo konkretno znanje z neko **hipotezo**. Dobra hipoteza je **skladna** *(consistent)* z učnimi primeri, kar pomeni, da bo za večino učnih primerov pravilno opisala razred. Ob tem moramo paziti, saj se lahko zgodi, da za neko domeno najbolj uspešna hipoteza zaradi šumnih učnih primerov sploh ni skladna.

Večina algoritmov za strojno učenje uporablja le en model, ki določenim domenam ustreza

bolj, drugim pa manj. Simbolične metode temeljijo na logičnem in lingvističnem, numerične pa na matematičnem jeziku. Logična pravila bodo ustrezna pri predvidevanju izhoda iz integriranega vezja, model na osnovi diferencialnih enačb bo primeren za opisovanja gibanja planetov, verjetnostni model pa je najbolj primeren za predvidevanje uspeha pri vlečenju kart iz kupčka in metanju bolj ali manj goljufivih kock.

Seveda pa istim podatkom lahko ustreza poljubno število hipotez v več različnih modelih. Ob tem je važen izraz **pristranskost** *(bias)*, ki označuje, s katerim modelom in katero hipotezo bo nek algoritem za strojno učenje naučeno znanje dejansko shranil in uporabljal. Pristranskost lahko uporabimo tudi pri iskanju ustrezne hipoteze, tako da najprej preizkusimo ustreznost najljubših hipotez. Čeprav ima pristranskost negativen prizvok, brez nje posploševanje sploh ne bi bilo mogoče. Pristranski moramo biti zato, ker bi bilo zaradi časovne in prostorske kompleksnosti učenja popolnoma nepraktično preveriti vse možne hipoteze, da bi našli primerne (ponavadi jih je tudi neskončno mnogo), potem pa se med več primernimi tudi odločiti za najboljšo. Primer pogoste in učinkovite pristranskosti je podobnost — če spečemo dve potici iz podobnih sestavin na podoben način, bosta obe imeli podoben okus. Ta tehnika pa vedno ne bo delovala — v podobnih okoliščinah v vojni ne bi bilo pametno z gotovostjo pričakovati, da bo nasprotnik uporabil isto diverzantsko taktiko, saj se zaveda, da smo se po njegovi prejšnji na podobne že pripravili.

Napačno bi bilo iskati model, ki bi bil ustrezen za vse domene in hkrati enostaven. Večina današnjih postopkov strojnega učenja uporablja neko precej omejeno množico enostavnih modelov. Pri vsakem modelu je dobro vedeti, za kakšne domene bo najbolj primeren. Če za neko domeno obstaja uspešen in enostaven deterministične model, bi bilo slabše vkalupljati domeno v nedeterministični model. Če lahko gibanje Zemlje okoli Sonca opišemo z enostavnimi diferencialnimi enačbami, bi tak model boljše deloval kot pa diskretizacija prostora osončja in napovedovanje položaja na nekem območju po metodi naivnega Bayesa. Ravno zato je treba na strojno učenje gledati tudi iz vidika integracije več modelov v en sam sistem. Novo in precej aktivno področje raziskovanja je učenje zbirk *(ensemble learning)* [25], kjer za neko domeno pripravimo več hipotez z manipulacijo učnih primerov ali odločitev pri gradnji hipotez. Vendar pa s popolno ločitvijo metod na tak način ne moremo uspešno obravnavati domen, ki vsebujejo več različnih sistemov, na primer učenje igranja na ruleti, kjer so pravila same igre lahko predstavljena s končnim avtomatom, rezultat igre v obliki zaslužka pa je od stanja odvisna stohastična spremenljivka. To poskušajo rešiti z učenjem hibridnih modelov.

Koristen napotek, sestavni del pristranskosti, ali **hevristika** za večjo odpornost proti šumu svetuje, naj ne uporabimo komplicirane hipoteze, če nam popolnoma zadošča že enostavna, saj se bomo s tem le v preveliki meri navezali le na šum in slučajne podobnosti, v skupnem pa nič pridobili. Ta napotek je znan kot **Occamova britev** *(Occam razor)*, ker ga je prvi omenil v 14. stoletju angleški teolog William of Occam kot: ≫*Frustra fit per plura, quod fieri potest per pauciora.*≪

Učimo se lahko **z vrha** *(top-down)* ali **z dna** *(bottom-up)*. V splošnem, pri učenju z vrha neko hipotezo preverimo na vseh primerih in ob tem tvegamo, da zaradi gozda ne vidimo dreves, pri učenju z dna pa postopoma dopolnjujemo hipotezo z upoštevanjem posameznih primerov, a se lahko med drevesi izgubimo. Za enkratno učenje nam bo verjetno zadoščalo učenje z vrha, pri učenju z ojačanjem pa bomo prisiljeni uporabiti tudi učenje z dna, saj vseh situacij ne moremo v nedogled shranjevati[1].

1.3. Geometrijska predstavitev znanja

Že prej smo z nekaj primeri pokazali, da sam jezik logike ni vedno najbolj primeren za opisovanje

[1] kjer se pomni le omejeno število primerov.

naravnih pojavov, predvsem zaradi pomanjkanja opisno zelo učinkovitih analitičnih matematičnih konceptov, recimo funkcij, odvodov, ipd. Po drugi strani pa je jezik matematike strukturno zelo podoben naravnim jezikom zaradi svoje zaporedne in diskretne zgradbe, ki izhaja iz deduktivne in aksiomatske usmeritve matematike. Današnji računalniki so nasledniki Turingovega stroja, ta pa je bil zamišljen za obravnavo matematičnih jezikov in konceptov s tem, da podatke postopoma z uporabo množice aksiomov pretvarja v nekaj drugega. Taka arhitektura je sicer zelo primerna za čisto dedukcijo, ni pa za indukcijo in odkrivanje vzorcev *(pattern matching)*.

Induktivno učenje temelji na dojemanju večjega števila učnih primerov hkrati, kar omogoča, da med njimi najdemo podobnosti. Ob odkrivanju vzorcev pa je zaželjeno, da na podatke kot celoto gledamo pri različnih stopnjah podrobnosti. Ko morajo take probleme reševati ljudje, ponavadi najprej poskušajo podatke združiti in prikazati na pregleden način. Zato kot pomožni orodji daleč prevladujeta statistika in vizualizacija, ki poskušata podatke čimbolj pregledno predstaviti človeku, pri tem se statistika zanaša na simbolične, vizualizacija pa na vizualne predstavitve podatkov. Vizualizacija se zanaša na intuicijo človeka, da bo prepoznal zakonitosti v podatkih ter jih znal opisati. Namen vizualizacije pa je predstaviti podatke tako, da jih bo lahko človek s svojimi čuti dojel v celoti in o njih lažje razmišljal. Ob tem vizualizacija uporablja barve, oblike, vzorce, animacijo in s tem omogoča sočasno zaznavanje velikega števila atributov.

Prej smo predstavili problema klasifikacije in regresije na podlagi atributov. Vsak atribut lahko predstavlja dimenzijo v nekem afinem prostoru, nek učni primer pa je točka *(point)* v tem prostoru s prirejeno vrednostjo — razredom. Taki točki pravimo tudi vektor značilk *(feature vector)*. Problem klasifikacije s tem postane vprašanje, kakšen razred ima neka točka v prostoru. V geometrijskem modelu znanja bodo hipoteze geometrijske strukture.

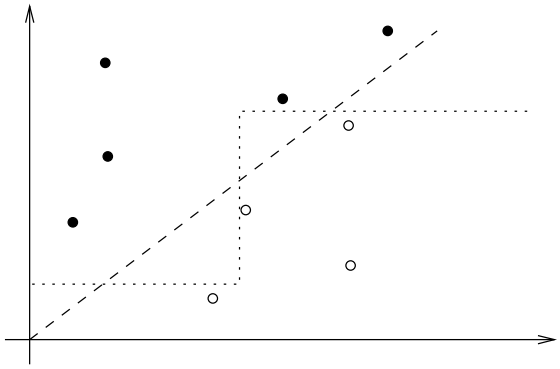
1.4. Izziv

Kljub opisani koristi obdelave več dimenzionalnih domen čimbolj celostno, z upoštevanjem čim večjega števila atributov hkrati, nekateri najbolj uveljavljeni modeli strojnega učenja, na primer odločitvena drevesa, upoštevajo pri odločitvah le en atribut naenkrat. Algoritmi pri izgradnji poskušajo izdelati čim boljše odločitveno drevo v smislu kompleksnosti ali klasifikacijske točnosti tako, da se pri vsaki odločitvi trudijo izbrati najbolj primeren atribut. Ob tem ocenjevanje atributov večinoma poteka z vrha navzdol in je kratkovidno, kar povzroči težave, če so atributi med seboj odvisni. Raziskovalci so se tega zavedali in predstavili izboljšane postopke za ocenjevanje atributov, ki ne trpijo zaradi preveč globalnega pogleda, tak je recimo RELIEF [46]. Vendar je težava v tem, da se pri posamezni odločitvi v vozlišču drevesa še vedno upošteva le en atribut naenkrat.

Z upoštevanjem večjega števila atributov hkrati ne bi dosegli bistvenih izboljšav, če bi bili atributi med seboj neodvisni ali pa če se jih ne bi dalo smiselno obravnavati po podobnosti. Upoštevanje podobnosti je namreč smiselno le, če podobnost vrednosti razredov za dva primera vsaj deloma izhaja tudi iz medsebojne podobnosti njunih atributov.

Če so atributi zvezni, lahko pridemo do zelo nenaravnih hipotez, kar prikazuje slika 1.1. Vzrok takim in podobnim hipotezam leži v kvantizaciji prostora atributov, z namenom, da bi lahko zvezne atribute obravnavali z enostavnimi logičnimi pravili, ki upoštevajo le en atribut naenkrat. Kvantizacija je razbijanje prostora atributov z izgubo natančnosti na končno mnogo intervalov.

Druga težava, s katero se soočimo pri odločitvenih drevesih in pravilih je, da so hipoteze opisane na način, ki ne omogoča enostavnega in učinkovitega dodajanja podatkov v hipotezo z upoštevanjem dodatnih učnih primerov, kar je potrebno na primer za učenje z ojačanjem. Dodajanje je sicer možno, a odločitveno drevo se po obsegu začne hitro povečevati, po strukturi pa postane fragmentirano. Algoritmi, ki bi to drevo uredili brez shranjevanja vseh primerov in ponovne obdelave,



Slika 1.1.: Mejna ploskev pri trivialni domeni, kjer se ugotavlja, ali vozniki vozijo prehitro in se meri premik po času, je lahko zelo zavajajoče predstavljena z odločitvenimi drevesi, ki predpostavljajo medsebojno neodvisnost atributov (drobno črtkana črta), v primerjavi z enostavnejšo in (v tej domeni) pravilno ločitveno hiperravnino.

avtorju niso znani.

Po drugi strani pa nesimbolični postopki, na primer metoda k-NN (k najbližjih sosedov) [22], ki upoštevajo več atributov naenkrat, hipoteze ponavadi ne predstavijo na enostaven način, ki bi bil primeren za predstavitev človeku ali za enostavno sklepanje o domeni in analizo njenih zakonitosti. V zvezi s tem delimo postopke strojnega učenja na lene *(lazy)*, za katere je hipoteza kar množica učnih primerov, in zagnane *(eager)*, ki poskušajo hipoteze predstaviti z nekim bolj koncentriranim opisom. Čeprav bi lahko lene metode popolnoma enako uspešno klasificirale kot zagnane, ne poskušajo znanja koncentrirati v hipoteze.

Ko morajo ljudje opisati ali preučiti klasifikacijsko domeno z dvema zveznima atributoma, se tega velikokrat lotijo tako, da jo predstavijo z diskretno množico oznak na ravnini *(scatter plot, scatter diagram)*, kjer vsaki oznaki ustreza nek učni primer, tip oznake pa razredu. Območje, ki pripada posameznemu razredu je potem omejeno z neko krivuljo. Podobno velja tudi za regresijske probleme, le da pri njih predstavlja vrednost razreda ena od koordinatnih osi. Ta način je ustrezen za vse domene — z odvisnimi ali neodvisnimi atributi. Krivulja je tudi učinkovita, prilagodljiva in razumljiva predstavitev znanja. Krivulje so uporabljene za predstavitev veliko vrst znanja, spomnimo se le izohips, izobat, izodinam, izorahij, izoseist, izogeoterm, izoklin, izogon iz geografije, izonef, izobar, izohiger, izohimen, izohiet, izohelij iz meteorologije, pa izofot, izoanomal, izogam, izohor in izohron iz fizike. Nešteto podobnih, a brezimnih prikazov informacij se pojavlja tudi v kemiji, strojništvu, metalurgiji, ekonomiji, itd.

Vizualno učenje na podlagi krivulj ima precej lepih lastnosti. Je lokalno, saj določen segment krivulje obravnava le delček cele domene. Lokalnost je zelo pomembna za hitro učenje, kar dokazuje tudi velik uspeh metod, ki temeljijo na RBF *(radial basis functions)* in SVM *(support vector machines)* [17]. Ob tem je zanimivo opaziti, da je lokalnost ponavadi povezana z lenobo. Metode z RBF v svojem bistvu spominjajo na izbiranje ali ustvarjanje relevantne in obtežene podmnožice učnih primerov, ob tem pa lokalno predpostavljajo neodvisnost atributov. Učenje s krivuljami pa ni tako, saj ne predpostavlja neodvisnosti in krivulje predstavi ločeno od učnih primerov. To se pokaže kot prednost, če hočemo hipotezo inkrementalno prilagoditi, saj lahko to naredimo s popravki na omejenem delu krivulje, brez potrebe po natančnem upoštevanju prejšnjih učnih primerov.

Na podlagi oblike krivulje se odločamo, kje bi se splačalo opraviti nove eksperimente in meritve, kar je potrebno za aktivno učenje. Kompleksnost hipoteze je enostavno razvidna na podlagi grobosti in kompliciranosti krivulje, oziroma števila parametrov, potrebnih za njen opis. To bi nam mogočilo

obravnavo šuma in iskanje najbolj ustrezne metrike. Krivuljo lahko obravnavamo analitično in poskušamo najti zakonitosti, gradiente, ipd. Več nivojskih krivulj nam tudi da dobro osnovo za interpolacijo med njimi in s tem obravnavo regresijskih problemov. V treh dimenzijah se uporabljajo namesto krivulj ploskve *(surfaces)*, uporabne še na veliko področjih, na primer pri medicinski vizualizaciji.

Krivulje lahko formalno predstavimo na različne načine, na primer kot zlepke, polinomske enačbe, itd. Način predstavitve ustreza konceptu modela. En model krivulj so tiste, predstavljene s polinomskimi enačbami. Konkretna krivulja ustreza hipotezi znotraj modela. Nekateri modeli dovoljujejo poljubno število parametrov. S polinomskimi enačbami predstavljene krivulje lahko temeljijo na polinomih različnih redov. Iz tega izhaja koncept stopenj podrobnosti. Neki množici učnih primerov lahko poskušamo najti ustrezno mejno krivuljo, ki je predstavljena s polinomom visokega reda, kar ustreza visoki stopnji podrobnosti. Po drugi strani pa lahko poskušamo isto domeno opisati s polinomom reda 2, kar ustreza nižji stopnji podrobnosti.

Zelo podrobne krivulje uporabljamo pri pomembnih ali negotovih odločitvah, manj podrobne pa, ko ni dovolj časa ali prostora za obdelavo, ko gre za tipične situacije ali ko podrobnosti ne igrajo velike vloge. Najprej poskusimo sprejeti odločitev z uporabo enostavne krivulje. Če je dani testni primer daleč od nje, torej dlje, kot je od nje največ oddaljena najbolj podrobna krivulja (to se da izračunati vnaprej), lahko sprejmemo odločitev kar na podlagi enostavne krivulje. Le če smo preblizu, da bi lahko nedvoumno sprejeli tako odločitev, uporabimo bolj podrobno krivuljo. Tako kombiniramo natančnost podrobnih opisov s hitrostjo enostavnih opisov. Na podlagi oddaljenosti testnega primera od krivulje lahko tudi sklepamo, da se nahajamo v neraziskanem področju.

V povezavi z Occamovo britvijo pričakujemo večjo klasifikacijsko natančnost pri uporabi nižjih stopenj podrobnosti, ker te v manjši meri trpijo zaradi problema prevelikega prileganja šumu v učnih podatkih. Čudno bi bilo, če bi opis modela vseboval več podatkov kot pa opis primerov samih. Ta ugotovitev je poznana kot princip MDL *(minimum description length)* [60], obstajajo pa tudi druge podobne formalizacije, na primer MML. V primeru, da parametrov šuma ne poznamo, moramo s tehnikami, kot na primer križnim preverjanjem, ugotoviti, katera stopnja podrobnosti bo dala najboljše rezultate.

Zanimivo je, da zaenkrat skoraj noben postopek strojnega učenja ne poskuša posnemati tega lokaliziranega geometrijskega načina učenja s krivuljami in ploskvami, ob tem pa avtorji spisov o strojem učenju in prepoznavanju vzorcev uporabljajo krivulje z izrazom **mejna ploskev** *(decision surface, decision boundary)* kot glavni način ilustracije hipotez in posploševanja. Seveda obstajajo postopki, ki tudi mejne ploskve eksplicitno obravnavajo, na primer učenje logističnih in linearnih diskriminante, vendar te metode gradijo globalne hipoteze z vrha navzdol.

Ljudje lahko rišejo in si predstavljajo nivojske krivulje ali ploskve le v dveh ali treh dimenzijah, kar je mogoče tudi meja človeški sposobnosti sklepanja, če ji ne pomagamo s statistiko ali vizualizacijo. Kako ta način geometrijskega razmišljanja z nivojskimi ploskvami in krivuljami posplošiti na prostor s poljubnim številom dimenzij?

2. Osnove

Ker se postopek klasifikacije s simplicialnimi kompleksi nanaša na teorijo iz več področij, ta razdelek predstavi matematične koncepte iz področij topologije, linearne algebre ter računske geometrije, na katere se bodo nanašali in jih uporabljali postopki učenja. Matematikom je priporočeno, da ta razdelek le preletijo, pogledajo pa naj si vsaj terminologijo ter implementacije nekaterih algoritmov računske geometrije na koncu razdelka. Kot osnovne reference smo uporabljali pri linearni algebri [57, 36, 59, 47], pri topologiji [45], pri računski geometriji pa [56, 53, 35, 3].

2.1. Mejne ploskve in topologija

Najprej moramo odgovoriti na zelo pomembno vprašanje: kako bomo opisali mejne ploskve v poljubno dimenzionalnem Evklidskem prostoru. Za uporabnost mejne ploskve je nujno, da ima nedvoumno dve strani. V topologiji temu ustreza koncept orientabilne mnogoterosti z mejo ali brez nje *(orientable manifold)*, ki mora prostor nedvoumno ločiti ali separirati na natanko dva dela. Na primer, daljica v 2-dimenzionalnem prostoru je orientabilna mnogoterost z mejo, vendar pa prostora ne loči na dva dela. Podmnožici evklidskega prostora $\mathbb{R}^k$ pravimo k-dimenzionalna mnogoterost z mejo, če ima vsaka točka okolico *(neighborhood)*, homeomorfno k-dimenzionalnemu odprtemu k-disku s središčem $\vec{0}$ in radijem r,

$$D^k(r) = \left\{ \vec{x} \in \mathbb{R}^k : \|\vec{x}\| < r \right\},$$

ali pol-disku s središčem $\vec{0}$ in radijem r,

$$D^k_+(r) = \left\{ \vec{x} = (x_1, x_2, \dots, x_k) \in \mathbb{R}^k : \|\vec{x}\| < r \wedge x_k \geq 0 \right\}.$$

Točke, ki imajo okolico ekvivalentno pol-disku sestavljajo rob. Naj omenimo še, da dimenzija mnogoterosti ponavadi ni enaka dimenziji prostora, v katerega jo postavljamo.

Za klasifikacijo je nujno za vsako konkretno točko v prostoru izračunati, na kateri strani mejne ploskve leži, pri čemer ni važna le topološka oblika mejne ploskve, temveč tudi njen položaj. Tako računanje je zelo učinkovito s hiperravninami. Hiperravnina je namreč primer $(n-1)$-mnogoterosti. Hiperravnino v n-dimenzionalnem vektorskem prostoru $\mathbb{R}^n$ opišemo z skalarjem d in vektorjem $\vec{n} \equiv [s_1, \dots, s_n]^T$. Neka točka $\vec{p} \in \mathbb{R}^n$ leži na hiperravnini če $\vec{n} \cdot \vec{p} = d$. Točke na hiperravnini predstavljajo $(n-1)$-dimenzionalni afini podprostor $\mathbb{R}^n$, a nas zaenkrat ne zanimajo, saj za klasifikacijo želimo za vsako točko (tudi za tisto na hiperravnini) povedati, natanko na kateri strani leži. Hiperravnina, določena s parom $\langle \vec{n}, d \rangle$, določa dva polprostora, odprtega negativnega, kjer velja $\vec{n} \cdot \vec{p} < d$, in pa zaprtega pozitivnega, kjer je $\vec{n} \cdot \vec{p} \geq d$. Pri klasifikaciji vsakemu polprostoru pripišemo svoj razred. V strojnem učenju obstaja postopek učenja z linearnimi diskriminantami, kjer je model predstavljen z eno hiperravnino.

Glavna težava hiperravnin pri klasifikaciji je njihova globalnost, želeli pa bi lokalnost, prednosti katere smo obravnavali malo prej. S polinomskimi enačbami predstavljene krivulje in ploskve tudi

trpijo zaradi svoje globalnosti — spreminjanje enega samega parametra vpliva na celotno krivuljo ali ploskev. Potrebovali bi neko bolj lokalizirano strukturo in to je **celični kompleks**, **celica** pa je njegov sestavni del.

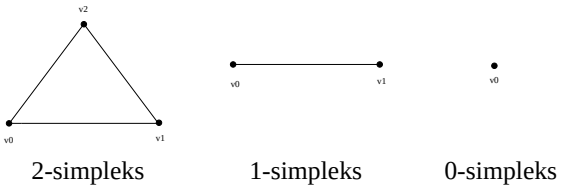
Notranjost *(interior)* n-celice mora biti homeomorfna odprtem n-dimenzionalnem disku, njen **rob** *(boundary)* pa mora biti sestavljen iz končnega števila celic nižje dimenzije, te celice imenujemo lica. 0-celica je na primer točka, 1-celica je daljica *(segment)*, 2-celica je poligon, 3-celica pa je omejen 3-dimenzionalen polieder *(polyhedron)*. Celice so še vedno relativno splošne, želeli bi si obravnavati le en tip n-celice. Najenostavnejše n-celice so **simpleksi:**

Neizrojen n-simpleks je določen z natanko $n+1$ v $\mathbb{R}^k, k \geq n$ afino neodvisnimi točkami, **oglišči** *(vertices)*, in je množica vseh konveksnih kombinacij teh točk. 2-simpleks je torej trikotnik, 3-simpleks pa tetraeder. n-simpleks, $n > 0$, je izrojen *(degenerate)*, če so njegova oglišča afino odvisna v $\mathbb{R}^n$. Vsak 0-simpleks je zaradi konsistentnosti definiran kot neizrojen. Če zanj izberemo nek vrstni red oglišč, ima simpleks tudi pozitivno ali negativno orientacijo, ki jo ohranjajo sode permutacije oglišč, lihe permutacije pa jo obrnejo.

Vsak n-simpleks ima natanko $n+1$ **glavnih lic** *(facets)*, ki so vse $(n-1)$-simpleksi. Glavna lica glavnih lic nekega simpleksa so **grebeni** *(ridges)*, povezave dveh oglišč pa kar **povezave** *(edges)*. Glavna lica so v trikotniku torej daljice, pri daljici pa oglišča. Ker pa si lica med seboj delijo tudi svoja lica, lahko izračunamo, da n-simpleks vsebuje $\binom{n+1}{m+1}$ m-simpleksov na svojem robu — torej vse izbire $m+1$ izmed $n+1$ oglišč. Te m-simplekse poimenujmo m**-lica** ali **podsimpleksi** *(subsimplices)*. Tetraeder na primer sestavljajo 4 trikotniki kot glavna lica, 4 oglišča in pa 6 daljic kot grebenov in povezav hkrati. 1-lica nekega n-simpleksa lahko predstavimo tudi kot povezan graf z $n+1$ oglišči, kar nam bo koristilo bi kombinatorični obravnavi simpleksov, še posebej tistih v prostorih višjih dimenzij, kjer si geometrijska telesa težje predstavljamo.

Neizrojen orientiran n-simpleks naj bo označen z $\Delta\langle\vec{v}_0, \vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n\rangle$ in omejuje določen del prostora. n-simpleks v $\mathbb{R}^n$ je izrojen natanko takrat, ko ima ničelen volumen. Neizrojen n-simpleks lahko opišemo tudi kot končen presek $n+1$ različnih zaprtih pozitivnih polprostorov.

Da bi poenostavili izraze, bomo za nek n-dimenzionalen prostor definirali še h-simpleks $S = \delta\langle\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n\rangle$ kot neizrojen orientiran $(n-1)$-simpleks v $\mathbb{R}^n$, katerega oglišča natanko določajo $h$iperravnino $\langle\vec{n}_S, d_S\rangle$ in ta razpenja dva polprostora. h-simpleks ima v $\mathbb{R}^n$ definirano končno in neničelno ploščino, ki ustreza volumnu v $\mathbb{R}^{n-1}$. V 3-dimenzionalnem prostoru bo torej h-simpleks trikotnik, n-simpleks pa tetraeder.



Slika 2.1.: Simpleksi.

Vendar pa s posameznimi simpleksi še ne moremo opisati mejnih ploskev, ki so mnogoterosti. K sreči se da vse gladke in vse 2-dimenzionalne mnogoterosti v $\mathbb{R}^n$ do poljubne natančnosti aproksimirati s simplicialnimi kompleksi tako, da jih trianguliramo — predstavimo s kompleksom, ki je unija neke množice h-simpleksov, tako da ne obstaja v njem nobeno lice, ki ne bi bilo sestavni del nekega h-simpleksa. Malo širši pojem celičnega kompleksa označuje končno množico zaprtih celic, katerih notranjosti se ne sekajo, dve celici pa imata lahko v skupnem le končno mnogo svojih celih lic ali pa nič. Torej ni dovoljeno, da bi se lahko dve celici v kompleksu sekali, ali pa da bi se na primer neke daljice dotikalo neko tretje oglišče. Simplicialni kompleks je kompleks sestavljen

iz samih simpleksov. Za kompleks samih n-simpleksov pa velja, da je vsako oglišče lice vsaj enega n-simpleksa.

2.2. Računanje s simpleksi

Da bi lahko simplekse koristno uporabili, bomo potrebovali nekaj orodij, ki nam jih nudi linearna algebra. Predpostavlja se, da je bralec seznanjen z osnovnimi koncepti, kot recimo normami, vektorskimi prostori, determinantami, rangi, defekti, matrikami, konveksnimi množicami, vektorskim produktom in s pojmom metrike in metričnega prostora. V tem besedilu se bomo osredotočili le na evklidske prostore in evklidsko normo.

Hiperravnino kot par normalnega vektorja in skalarja $\langle \vec{n}, d \rangle$ smo sicer že definirali, a poglejmo si še, kako do nje pridemo in kaj lahko z njo naredimo. Zanima nas predvsem določiti hiperravnino v n-dimenzionalnem prostoru, na kateri bo ležalo n točk $\langle \vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n \rangle$. V treh dimenzijah pridemo do normalnega vektorja $\vec{n}$ z uporabo vektorskega produkta, tako da je $\vec{n} = (\vec{v}_2 - \vec{v}_1) \times (\vec{v}_3 - \vec{v}_1)$, če le $\vec{n} \neq \vec{0}$. Kaj pa bomo naredili v dimenzijah, ki niso enake 3?

Odgovor je posplošitev vektorskega produkta na poljubno število dimenzij. Kot vemo, lahko v 3-dimenzionalnem vektorskem prostoru, opisanem z ortonormirano bazo $\left\{\vec{i}, \vec{j}, \vec{k}\right\}$, vektorski produkt dveh vektorjev $\vec{x}_1$ in $\vec{x}_2$ zapišemo kot determinanto:

$$\begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ \vec{x}_1 \cdot \vec{i} & \vec{x}_1 \cdot \vec{j} & \vec{x}_1 \cdot \vec{k} \\ \vec{x}_2 \cdot \vec{i} & \vec{x}_2 \cdot \vec{j} & \vec{x}_2 \cdot \vec{k} \end{vmatrix}$$

V n-dimenzionalnih prostorih, kjer n ni 3, pa vektorski produkt ni več binarna operacija, temveč funkcija $n-1$ vektorjev. Označimo ga kot $\times(\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_{n-1})$ in priredi neničelnim linearno neodvisnim vektorjem $\vec{v}_1, \ldots, \vec{v}_{n-1}$ vektor $\vec{n}$, ki je pravokoten na ravnino, na kateri ležijo $\vec{v}_1, \ldots, \vec{v}_{n-1}$, zaporedje $\langle \vec{v}_1, \ldots, \vec{v}_{n-1}, \vec{n} \rangle$ pa ima v $\mathbb{R}^n$ isto orientacijo kot zaporedje $\langle \vec{e}_1, \ldots, \vec{e}_n \rangle$ baznih vektorjev kanonične baze. V 1-dimenzionalnem prostoru je normalni vektor kar bazni vektor sam, v 2-dimenzionalnem pa ga lahko določimo z naslednjo determinanto:

$$\begin{vmatrix} \vec{i} & \vec{j} \\ \vec{x} \cdot \vec{i} & \vec{x} \cdot \vec{j} \end{vmatrix}$$

Če postavimo $\vec{x}_k = \vec{v}_k - \vec{v}_1$, kjer $\vec{x}_{k,1}$ označuje skalarni produkt $\vec{x}_k$ s prvim vektorjem kanonične baze, za vsak $\vec{n}$ velja:

$$\vec{n} = \left[\begin{vmatrix} \vec{x}_{2,2} & \vec{x}_{2,3} & \cdots & \vec{x}_{2,n} \\ \vec{x}_{3,2} & \vec{x}_{3,3} & \cdots & \vec{x}_{3,n} \\ \vdots & \vdots & \ddots & \vdots \\ \vec{x}_{n,2} & \vec{x}_{n,3} & \cdots & \vec{x}_{n,n} \end{vmatrix}, - \begin{vmatrix} \vec{x}_{2,1} & \vec{x}_{2,3} & \cdots & \vec{x}_{2,n} \\ \vec{x}_{3,1} & \vec{x}_{3,3} & \cdots & \vec{x}_{3,n} \\ \vdots & \vdots & \ddots & \vdots \\ \vec{x}_{n,1} & \vec{x}_{n,3} & \cdots & \vec{x}_{n,n} \end{vmatrix}, \ldots, (-1)^{n+1} \begin{vmatrix} \vec{x}_{2,1} & \vec{x}_{2,2} & \cdots & \vec{x}_{2,n-1} \\ \vec{x}_{3,1} & \vec{x}_{3,2} & \cdots & \vec{x}_{3,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \vec{x}_{n,1} & \vec{x}_{n,2} & \cdots & \vec{x}_{n,n-1} \end{vmatrix} \right]^T$$

kar ustreza $\vec{n} = \times(\vec{v}_2 - \vec{v}_1, \vec{v}_3 - \vec{v}_1, \ldots, \vec{v}_n - \vec{v}_1)$.

V primeru, da je $\vec{n}$ enak $\vec{0}$, potem z danimi točkami ne moremo določiti hiperravnine, saj vse ležijo na nekem m-dimenzionalnem podprostoru, kjer je $m < n-1$. Če je $|\alpha| > 0$, določa par $\langle \vec{n}, d \rangle$ isto hiperravnino kot $\langle \alpha\vec{n}, \alpha d \rangle$. Ko je $|\vec{n}| = 1$, torej ko je $\vec{n}$ normaliziran in vsako hiperravnino lahko zapišemo tako, da je $\vec{n}$ normaliziran, je za neko točko $\vec{p}$ njena oddaljenost od hiperravnine enaka $|\vec{n} \cdot \vec{p} - d|$. Projekcija točke $\vec{p}$ na hiperravnino je $\vec{p}' = \vec{p} - (\vec{n} \cdot \vec{p} - d)\vec{n}$, hkrati pa je $\vec{p}'$ od vseh točk na hiperravnini najbližja točki $\vec{p}$.

Z vektorskim produktom pa lahko tudi izračunamo volumen nekega n-simpleksa [18]:

$$\mathrm{Vol}\left(\Delta\left\langle \vec{\mathrm{v}}_0, \vec{\mathrm{v}}_1, \ldots, \vec{\mathrm{v}}_\mathrm{n}\right\rangle\right) = \frac{\left|\times(\vec{\mathrm{v}}_1 - \vec{\mathrm{v}}_0, \vec{\mathrm{v}}_2 - \vec{\mathrm{v}}_0, \ldots, \vec{\mathrm{v}}_\mathrm{n} - \vec{\mathrm{v}}_0)\right|}{\mathrm{n!}}$$

Ko imamo v n-dimenzionalnem prostoru $n+1$ točk $\langle \vec{v}_0, \vec{v}_1, \ldots, \vec{v}_n \rangle$, ki sestavljajo neizrojen n-simpleks, jim lahko priredimo natanko eno n-kroglo tako, da vse točke ležijo na njeni površini. V primeru, da je ustrezni n-simpleks izrojen, ima n-krogla neskončen radij in je enaka hiperravnini, na kateri so vse točke $\vec{v}_0, \ldots, \vec{v}_n$. Ta hiperravnina je lahko ena sama, ali pa jih je poljubno mnogo.

Središče n-krogle je tam, kjer se sekajo vse take hiperravnine, ki so ekvidistantne od vsakega možnega para točk. Za točki $\vec{a}$ in $\vec{b}$ je od njiju ekvidistantna ravnina $\left\langle \vec{a} - \vec{b}, \frac{(\vec{a}+\vec{b})\cdot(\vec{a}-\vec{b})}{2} \right\rangle$. Da pridemo do središča $\vec{c}$, si izberimo eno izmed n točk, recimo $\vec{v}_0$, za konico *(apex)*, potem pa rešimo naslednji sistem linearnih enačb:

$$\begin{bmatrix} \vec{v}_1 - \vec{v}_0 \\ \vec{v}_2 - \vec{v}_0 \\ \vdots \\ \vec{v}_n - \vec{v}_0 \end{bmatrix} \cdot \vec{c} = \frac{1}{2} \begin{bmatrix} (\vec{v}_1 + \vec{v}_0)\cdot(\vec{v}_1 - \vec{v}_0) \\ (\vec{v}_2 + \vec{v}_0)\cdot(\vec{v}_2 - \vec{v}_0) \\ \vdots \\ (\vec{v}_n + \vec{v}_0)\cdot(\vec{v}_n - \vec{v}_0) \end{bmatrix}$$

2.3. Gradnja simplicialnih kompleksov

Računska geometrija *(computational geometry)* se loteva geometričnih problemov na algoritmičen način. V zvezi s tem obravnava časovno in prostorsko kompleksnost, računsko natančnost postopkov, podatkovne strukture ter druge dejavnike, povezane z računalniki.

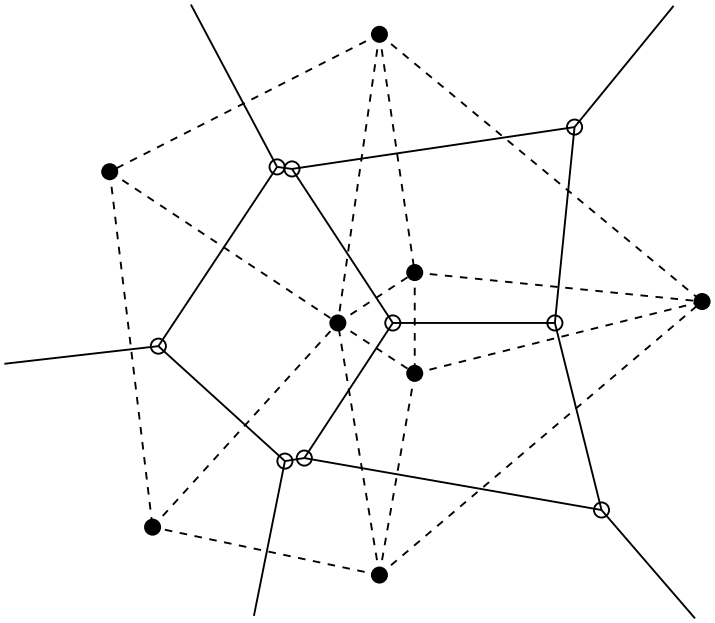
Eden izmed najbolj osnovnih pojmov v računski geometriji je **konveksna ovojnica** *(convex hull)*. Če je dana množica točk $S \subset \mathbb{R}^n$, je konveksna ovojnica conv(S) definirana kot najmanjša konveksna množica v $\mathbb{R}^n$, ki vsebuje S. Množico conv(S) natanko opisuje množica ekstremnih točk množice S, saj se vsako drugo točko v conv(S) lahko opiše z konveksno kombinacijo teh ekstemnih točk. Množica ekstremnih točk je tudi podmnožica S. V večini uporab je S končna in algoritmi to tudi predpostavljajo. V tem primeru je vsaka točka S oblike $\sum_{i=1}^{|E|} |\alpha_i| \vec{s}_i$, kjer je $E \subset S$ množica ekstremnih točk množice S in velja $\vec{s}_i \in E$, $\sum_i \alpha_i = 1, 0 \leq \alpha_i \leq 1$. Simpleks je konveksna ovojnica svojih oglišč.

Računska geometrija se tudi ukvarja s problemom najbližjih sosedov. Najpomembnejša struktura za ta namen je **Voronojev diagram**, ki ge je opisal Voronoi leta 1908, in je prikazan na sliki 2.2. Sicer je bil koncept neodvisno razvit tudi na drugih področjih pod imeni Thiessenov diagram, Wigner-Seitzev diagram, Blumova transformacija in pa že leta 1850 kot Dirichletovo tlakovanje *(tesselation)*, kar le še poudarja njegovo izredno pomembnost in vsestransko uporabnost.

Če vzamemo neko množico točk $P \subset \mathbb{R}^n$ z močjo m, poimenujmo jih **položaji** *(sites)*, lahko točke v prostoru priredimo najbližjemu položaju in s tem ustvarimo m **Voronojevih območij** *(Voronoi regions)*. Vsako Voronojevo območje je lahko predstavljeno kot presek neke množice polprostorov, zato lahko za poljubno točko enostavno ugotovimo, kateremu območju pripada. Tiste točke, ki pa jih ne moremo tako prirediti, ker so enako oddaljene od dveh ali več položajev, sestavljajo Voronojev diagram $\mathcal{V}(\mathcal{P})$. Čeprav se bomo ves čas ukvarjali le z evklidsko metriko, lahko Voronojev diagram definiramo za poljubno metriko pa tudi za primere, ko je posamezen položaj lahko sam množica točk.

V $\mathbb{R}^n$ so **Voronojeva oglišča** *(Voronoi vertices)* točke, kjer se stika vsaj $n+1$ Voronojevih območij, torej točke, ki so enako blizu $n+1$ ali več položajem. Mejam med območji, torej točkam, ki so enako blizu natanko dveh položajev, pa pravimo **Voronojeve stranice** *(Voronoi facets)*, v dveh

dimenzijah tudi Voronojevi robovi. Za Voronojeve stranice velja, da vedno ležijo na hiperravnini, ekvidistantni od para položajev, in so vedno konveksne. Nekatere stranice so neskončne, takim pravimo poltraki ali **žarki** *(rays)*. Končne Voronojeve stranice so konveksne ovojnice oglišč, ki jih določajo. **Naravni sosedje** *(natural neighbors)* nekega položaja so vsi položaji, s katerimi si ta deli kako Voronojevo stranico.



Slika 2.2.: Voronojev diagram in Delaunayeva triangulacija, ki je na tej skici črtkana. Položaji so polne, Voronojeva oglišča pa prazne točke. Zunanji rob Delaunayeve triangulacije predstavlja konveksno ovojnico množice položajev.

Voronojevemu diagramu dualna struktura je **Delaunayeva triangulacija** *(Delaunay triangulation)*, ki si jo oglejte na sliki 2.2. Dualnost izhaja iz tega, da vsakemu Voronojevemu oglišču v ravnini ustreza natanko en trikotnik v Delaunayevi triangulaciji, Voronojevo oglišče je središče kroga, očrtanega ogliščem trikotnika. Položaji natanko ustrezajo ogliščem v Delaunayevi triangulaciji. Poleg tega vsaki stranici med dvema sosednjima položajema v Voronojevem diagramu ustreza povezanost dveh trikotnikov v Delaunayevi triangulaciji preko skupnega glavnega lica.

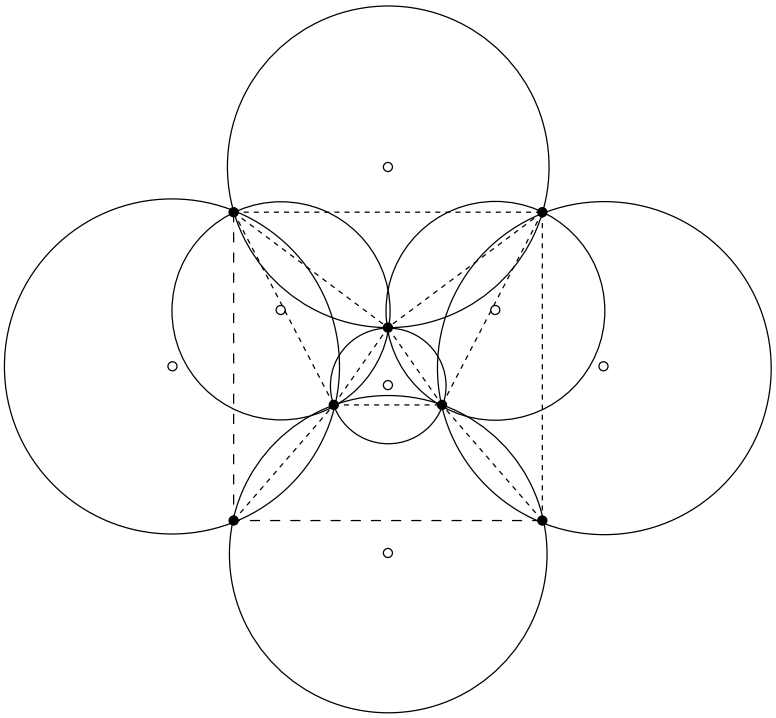
V $\mathbb{R}^n$ je osnovni element Delaunayeve triangulacije namesto trikotnika nek n-simpleks, Delaunayeva triangulacija pa je simplicialni kompleks, ki je unija samih n-simpleksov. Zato bomo uporabljali izraz **Delaunayev kompleks**. V treh dimenzijah se na primer uporablja izraz Delaunayeva tetraedralizacija. Zunanji rob Delaunayeve triangulacije je konveksna ovojnica množice položajev.

Delaunayev kompleks za neko končno množico položajev je končen in mu lahko pripišemo tudi graf, kjer povezave v njem ustrezajo glavnim licem n-simpleksov v kompleksu, vozlišča pa ogliščem Delaunayevih simpleksov, oziroma položajem. Torej je vozlišče v Delaunayevem grafu povezano z vsemi vozlišči, ki ustrezajo sosednjim položajem v Voronojevem diagramu.

Delaunayev kompleks je tudi bolj enostavno računati kot pa Voronojev diagram, saj se lahko uporabi kar algoritem za računanje konveksne ovojnice v za ena višji dimenziji, tako da položaje projeciramo na paraboloid. Sicer pa se s temi algoritmi bolj podrobno tukaj ne bomo ukvarjali, saj jih lahko najdemo v [56].

Značilno za Delaunayev kompleks je, da znotraj n-krogle, očrtane ogliščem nekega Delaunay-

evega simpleksa, ni nobenega drugega položaja, vsakemu položaju pa ustreza vsaj ena n-krogla. Ta lastnost je uporabna za dinamično spreminjanje Delaunayevega kompleksa v smislu dodajanja, brisanja ali premikanja položajev, saj se spremembe v kompleksu lahko zgodijo le znotraj krogel, ki vsebujejo spremembo [68, 49].



Slika 2.3.: Delaunayeva triangulacija s krožnicami, ki so očrtane Delaunayevim trikotnikom.

V linearnem času lahko na podlagi dualnosti iz Delaunayevega kompleksa pridemo do Voronojevega diagrama tako, da najprej izračunamo za vsak Delaunayev simpleks ustrezno Voronojevo oglišče, tako da najdemo središče krogle, očrtane vsem oglišči simpleksa. Zdaj moramo Voronojeva oglišča le še povezati skupaj v Voronojeve stranice za vsak par sosednjih položajev. To naredimo tako, da dodamo v Voronojevo stranico vsa Voronojeva oglišča, ki pripadajo Delaunayevim simpleksom, ki vsebujejo kot lice povezavo med položajema, ki ju v Voronojevem diagramu loči ta stranica.

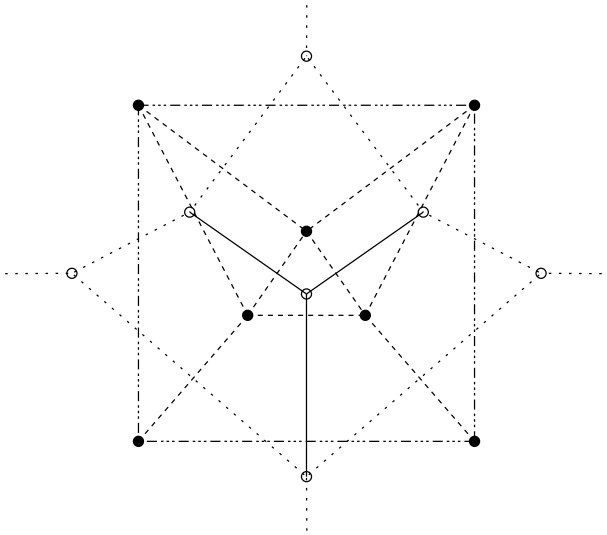
Seveda pa ni vse tako enostavno. Pozorni bralec je mogoče že ugotovil, da Voronojev diagram v $\mathbb{R}^n$ obstaja in je vedno enoličen tudi takrat, ko Delaunayev kompleks ne obstaja, ko ta ni enoličen ali ko vsebuje izrojene simplekse. Delaunayev kompleks vsebuje izrojene simplekse, če kompleksa ne moremo sestaviti, ne da bi na eni n-krogli ležalo več kot $n+1$ položajev (na primer pri kvadratu), če bi morali dopustiti krogle z neskončnim radijem (na primer v primeru več koplanarnih točk), ali pa če je položajev manj kot $n + 1$ (kar ne zadošča za n-simpleks).

Problematično je tudi računanje Voronojevih žarkov, saj jih iz Delaunayevega kompleksa na enostaven in hiter način ne moremo dobiti. Žarki povečujejo kompleksnost algoritmov kot nov tip podatkov, ki ga moramo upoštevati za vse operacije, ki bi jih na Voronojevem diagramu hoteli opravljati. Zato bi se jim radi izognili. Moramo se sicer zavedati, da nekateri Voronojevi diagrami brez žarkov niso več aproksimacije za orientabilne mnogoterosti, ki prostor separirajo, in posledično sami po sebi niso uporabni kot mejne ploskve.

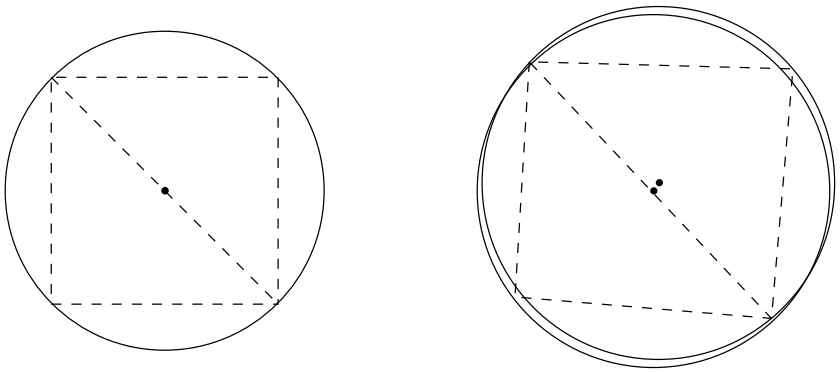
Kljub temu bi si želeli za vse primere zaradi enostavnosti uporabljati le zgoraj opisani postopek

generiranja Voronojevih diagramov z uporabo dualnosti. Za to najprej zapremo vse položaje v dovolj veliko hiperkocko, ki jih vse vsebuje *(bounding box)*, s čemer ustvarimo dovolj točk za Delaunayevo triangulacijo in "prirežemo" Voronojeve žarke. Žarki so potem namreč le med oglišči te n-kocke in ti nas ne zanimajo, saj se ukvarjamo le z dano množico položajev. Ob tem lokalno Voronojevih stranic ne pokvarimo, saj te vedno leže na ekvidistantni hiperravnini med položajema. Pozorni pa moramo biti na eksponentno rast kompleksnosti glede na dimenzijo, saj ima n-kocka 2^n oglišč.

Da se izognemo izrojenim simpleksom, naključno za zelo majhno vrednost perturbiramo ali stresemo *(joggling, perturbation)* položaje, dokler izrojenih Delaunayevih simpleksov ni več. Ko enkrat uspešno dobimo Delaunayev kompleks, ponovno z uporabo originalnih, nepremaknjenih položajev in dobljenega kompleksa izračunamo Voronojev diagram.



Slika 2.4.: Za tri položaje lahko z Delaunayevo triangulacijo izračunamo omejen Voronojev diagram brez žarkov, če jih zapremo v škatlo.



Slika 2.5.: Premikanje položajev za majhne naključne vrednosti preprečuje, da bi se pojavile prekrivajoče se ali neskončno velike krožnice. V desnem primeru sta Voronojevi oglišči tako blizu skupaj, da jih bomo kasneje združili. V praksi bomo položaje premikali manj kot v zgornji skici.

Stranice, ki vsebujejo več kot n oglišč, lahko vedno razbijemo na kompleks h-simpleksov po

algoritmu "rezanja ušes" *(ear snipping)* tako, da bo kompleks natanko ustrezal stranici. Ob tem se spomnimo, da so stranice vedno končne (ker nas žarki ne zanimajo več) in konveksne, ter da je v n dimenzijah vsako oglišče povezano z natanko $n-1$ drugimi oglišči v stranici. Nekemu oglišču sosednja so tista oglišča, katerih ustrezni Delaunayevi simpleksi imajo za skupno lice nek h-simpleks. Če ste pozabili, v $\mathbb{R}^2$ je h-simpleks daljica, v $\mathbb{R}^3$ pa trikotnik. Naključno si izberimo neko oglišče, ki ima natanko $n-1$ sosed, ter ustvarimo h-simpleks z njimi. Sosednja oglišča so zdaj zgubila eno sosedo, ob tem pa jih vse povežemo med seboj. Po taki operaciji stranica ostane še vedno konveksna. Če v preostanku stranice še vedno dovolj oglišč, operacijo ponovimo.

Lahko se zgodi, da so proizvedeni simpleksi zelo ozki in tanki — če imajo majhno ploščino in velik radij očrtanega trikotnika. Poseben primer tega so izrojeni simpleksi z ničelno ploščino. Temu bi se radi izzognili, saj izrojeni simpleksi niso uporabni za klasifikacijo, ozki simpleksi pa s sabo prinašajo različne težave z numerično natančnostjo. V primeru, da ob razbitju naletimo na tak simpleks, poskušamo stranico razbiti drugače, tako da uporabimo drugačno zaporedje odstranjevanja oglišč. Izrojenih simpleksov ne moremo kar odstraniti, saj bi se potem lahko zgodilo, da več sosednjih stranic ne bi mogli povezati skupaj v simplicialni kompleks.

S spreminjanjem vrstnega reda rezanja oglišč ne moremo vedno rešiti takih problemov. V taki situaciji si pomagamo tako, da ustvarimo posebna oglišča na primernih mestih znotraj stranice, potem pa vse skupaj razbijemo. V takem primeru je treba algoritem za razbijanje prilagoditi, saj postopek z rezanjem ušes ne deluje. Čeprav obstajajo enostavnejši postopki, bi končno lahko uporabljali kar Delaunayevo triangulacijo.

Po teh operacijah odstranjevanja neskončnih žarkov ter razbitja stranic na simplekse, nam ostane od Voronojevega diagrama simplicialni kompleks samih h-simpleksov, ki pa vključuje vse, kar rabimo za klasifikacijo.

3. Uresničitev

V prejšnjem razdelku smo spoznali matematična orodja, ki jih bomo potrebovali za realizacijo koncepta mejnih ploskev. Simplicialni kompleks si namreč moramo predstavljati kot končno unijo h-simpleksov, ki je poljubno natančen približek za neko mejno ploskev. Sam Voronojev diagram ter ustrezni simplicialni kompleks še nista primerna za klasifikacijsko učenje po definiciji iz Uvoda.

V Uresničitvi je iz računalniške perspektive predstavljena sama implementacija postopkov učenja in klasifikacije, skupaj s programskimi orodji in podatkovnimi strukturami. Omenjene bodo težave ter kako se jim izognemo. Pripravili bomo rešitve, ki ne omogočijo le učenja, temveč tudi sprotno spreminjanje in prilagajanje hipotez, kot tudi različne razširitve osnovnega postopka.

3.1. Razvojno okolje

Predstavitev matrik, vektorjev in drugih struktur, ki se uporabljajo v geometriji s poljubno dimenzijami, je relativno kompleksna. To zelo uspešno rešuje objektno usmerjeni programski jezik C++. Namesto posebnih funkcij za operacije z vektorji in matrikami zdaj lahko uporabljamo kar običajne operatorje za množenje, seštevanje, ipd., če le prej pripravimo razrede zanje in če se sprijaznimo z manjšim zmanjšanjem hitrosti izvajanja. Robert Davies *(http://webnz.com/robert/)* je izdelal knjižnico newmat09, ki vse to zelo dobro opravlja, ponuja pa tudi več numeričnih algoritmov in je v javni lasti.

Čeprav so algoritmi v računski geometriji dokaj dobro raziskani, so implementacije kompleksne in imajo velikokrat težave z robustnostjo, natančnostjo ter višjimi dimenzijami. Uporabljeno je bilo verjetno najkvalitetnejše orodje QHull avtorjev C. Bradforda Barberja in H. Huhdanpaa iz The Geometry Center pri University of Minnesota v Minneapolisu *(http://www.geom.umn.edu/locate/qhull)* [8]. Ker je QHull napisan v programskem jeziku C z neobjektnimi podatkovnimi strukturami, je bil izdelan poseben vmesnik, ki komunicira s QHull in pretvori izhod v interni format, uporabljen v aplikaciji.

Interne podatkovne strukture uporabljajo knjižnico standardnih podatkovnih struktur in algoritmov STL podjetja SGI. Ta knjižnica nudi robustne podatkovne strukture, ki same skrbijo za alokacijo in realokacijo pomnilnika in je bistveno skrajšala čas razvoja.

Za prebiranje datotek z učnimi in testnimi primeri je bila uporabljena knjižnica v javni lasti podjetja RuleQuest Research *(http://www.rulequest.com)* iz paketa See5.0.

Uporabljena prevajalnika sta bila Microsoft Visual Studio 6.0 ter Watcom C/C++ 10.6.

S temi orodji je bila razvita aplikacija za ukazno vrstico SICC *(SImplicial Complex Classification)*, ki omogoča izgradnjo klasifikacijskih simplicialnih kompleksov ter klasifikacijo. Interaktivni uporabniški vmesnik zaradi obsežnosti postopkov vizualizacije v treh in neizvedljivosti v višjih dimenzijah ni bil izdelan, zato pa SICC dovoljuje izvoz domen z dvema ali tremi atributi v format VRML *(Virtual Reality Markup Language)*, kar si lahko ogledate na sliki B.5 na strani 47. Čeprav SICC zaenkrat še ne dovoljuje poenostavljanja hipotez, bi se lahko v dveh in treh dimenzijah uporabilo že obstoječe postopke in aplikacije, na primer Garlandov QSlim [33] za poenostavljanje s simplicialnimi kompleksi opisanih grafičnih modelov, saj je način predstavitve hipotez in grafičnih

modelov v obliki množice trikotnikov skoraj identičen.

Leta 1998 razvita aplikacija CCW (slika B.1, stran 45), ki omogoča interaktivno ustvarjanje domen z dvema atributoma, klasifikacijo ter enostavno poenostavljanje mejnih krivulj z odstranjevanjem točk iz klasifikacijskega simplicialnega kompleksa ob ohranjanju popolne konsistentnosti na učnih primerih. CCW omogoča tudi nekatere spremembe množice učnih primerov, kot na primer dodajanje šuma atributom ter njihovo koreliranje, za analizo vplivov takih sprememb na uspešnost učenja. CCW vsebuje tudi postopek k-NN za primerjavo rezultatov. Sicer CCW ne gradi mejnih krivulj na podlagi Voronojevega diagrama, ampak uporablja podoben a enostavnejši postopek.

3.2. Priprava učnih primerov

Kot vemo, so učni primeri lahko predstavljeni kot točke v prostoru. Voronojevo območje omejuje del prostora, ki je najbližje nekemu položaju. Najočitnejša povezava med konceptoma je, da kot položaje pri gradnji Voronojevega diagrama uporabimo kar učne primere.

Voronojevi diagrami so zgrajeni na podlagi evklidske metrike, ki deluje kot način primerjanja podobnosti med dvema točkama v prostoru z upoštevanjem vseh atributov. Metrika je torej način združevanja razlik med vsemi atributi v eno samo vrednost. Če ne predpostavimo neodvisnosti med atributi, se moramo soočiti s problemom, kako različne atribute primerjati med sabo. Na primer, pri domeni napovedovanja vremena bi moral postopek enako učinkovito delovati v primeru, da so temperature v stopinjah Celzija ali če so v stopinjah Fahrenheita. Postopek mora hkrati upoštevati zelo različne količine, na primer težo v tonah in pa temperaturo v stopinjah.

Ta problem je neizogiben. Če hočemo naenkrat upoštevati večje število atributov, moramo tudi vedeti, koliko vsak od njih prispeva h končnemu rezultatu. Ko to vemo, imamo neko splošno mero podobnosti in lahko uspešno posplošujemo naučeno hipotezo na primere, ki jih še nismo videli. Ta problem obravnava večina metod, ki upoštevajo več zveznih atributov naenkrat, od različnih variant k-NN, do nevronskih mrež in SVM.

Razširjen in praktičen način za realizacijo tega koncepta so uteži. Vsakemu atributu lahko pripišemo neko mero pomembnosti na enoto, temu pravimo **utež** in za i-ti atribut jo označimo z w_i. Obtežena evklidska metrika definirana kot $d(\vec{a},\vec{b}) \equiv \sqrt{\sum_{i=1}^{n}(w_i\vec{a}_i - w_i\vec{b}_i)^2}$. Osnovna metoda z utežmi torej zahteva jemanje metrike kot spremenljive funkcije, kar v večini primerov ni zaželjeno. Alternativna rešitev je transformacija prostora atributov, kar omogoča identične rezultate kot obtežena metrika. Namesto da bi spreminjali metriko v algoritmih računske geometrije, dosežemo enake rezultate tako, da prostor atributov raztegnemo *(scale)* z vektorjem $[w_1, w_2, \ldots, w_n]$, kar moramo pred uporabo storiti za vse učne in testne primere.

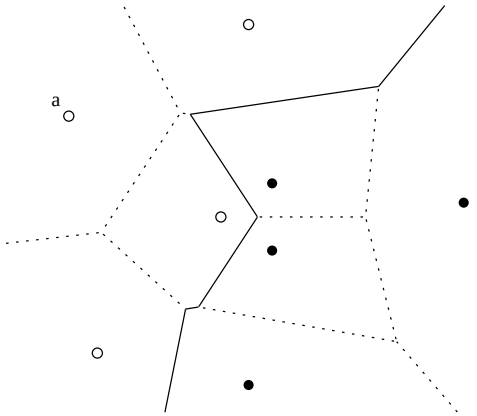
Poseben in skrajno netrivialen problem predstavlja določanje uteži, od česar je odvisna uspešnost klasifikacije. Zelo razširjena hevristika je skaliranje vsakega atributa na neko fiksno območje, na primer na $[0, 1]$, kar počne tudi SICC. Ambicioznejše metode uporabljajo PCA *(principal component analysis)* in še kaj drugega, vendar pa rezultati niso signifikantno boljši. Drugi postopki, ki spominjajo na učenje enostavnih nevronskih mrež, se celo sami učijo katere uteži so najbolj odgovorne za napako, na podlagi napačno klasificiranih primerov. Bolj natančno je ta tematika opisana v [71]. Sicer pa lahko tudi na druge načine spreminjamo metriko, da le dobimo kar najboljši klasifikator.

3.3. Klasifikacija

Zaenkrat bomo predpostavili, da imamo le dva razreda, kar pa zadošča za poljuben klasifikacijski problem. Namreč, neko omejeno množico razredov z močjo večjo od dva lahko vedno razbijemo na dve podmnožici, ter se naučimo ločevati med njima. Ti podmnožici naprej rekurzivno razbijamo

in se učimo. Na koncu dobimo iskalno drevo in $\log_2 n$ klasifikatorjev, ne da bi se sploh dotaknili osnovnega postopka za učenje in klasifikacijo. Sicer se da doseči boljše rezultate, še posebej, ko obstaja med razredi neka urejenost.

Voronojev diagram ločuje Voronojevo območje vsakega posameznega položaja od območij, ki pripadajo vsem ostalim položajem. Seveda pa ima v praksi pri klasifikaciji veliko učnih primerov enak razred in želimo ločevati le med območji, ki pripadajo različnim razredom. Zato nas zanima tudi le del Voronojevega simplicialnega kompleksa in to le tiste stranice, ki ločijo regije, ki pripadajo učnim primerom različnih razredov. Sam algoritem se torej sprehodi skozi vse Voronojeve stranice ter doda v simplicialni kompleks le tiste, ki ustrezajo zgornjemu pogoju. Na tak način pridobljen **mejni simplicialni kompleks**, sestavljen iz samih h-simpleksov, je približek za poljubno mejno ploskev med razredoma.



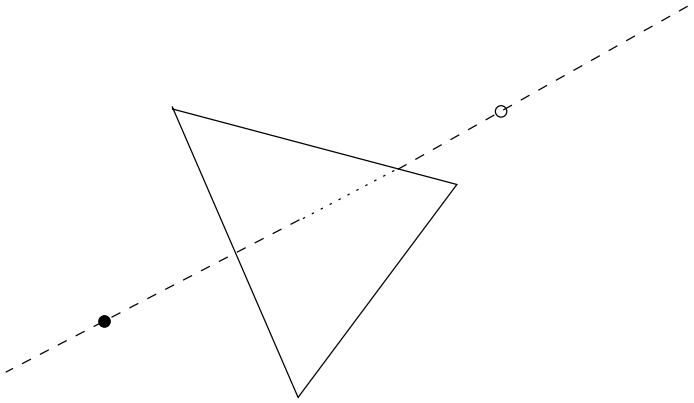
Slika 3.1.: Stranice, ki ločijo regije pripadajoče različnim razredom, so od Voronojevega diagrama vse, kar rabimo za klasifikacijo. Mejni simplicialni kompleks bi ostal enak, tudi če bi položaj a odstranili.

Zastavita se pomembni vprašanji — kako izvajati klasifikacijo zdaj, ko ni več ne enostavnih konveksnih regij, omejenih s polprostori, ne žarkov, zaradi česar je kompleks vedno omejen in sam po sebi ni več ustrezna mejna ploskev.

Že prej smo omenili, da h-simpleks leži v natanko določeni hiperravnini in zato lahko služi kot osnova za klasifikacijo. Seveda pa hiperravnina še ne zadošča za klasifikacijo, saj ne vemo, kateremu polprostoru pripada nek razred. Ker smo predpostavili obstoj le dveh razredov, bo usmeritev h-simpleksa opredelila tudi razred. Naj negativni polprostor ustreza razredu 0 in pozitivni razredu 1. Vsak h-simpleks vedno leži na neki stranici Voronojevega diagrama. Vemo, da vsaka stranica ločuje regiji, ki pripadata dvema sosednjima učnima primeroma in ta stranica je v Voronojevem diagramu tista, na podlagi katere se odločitev opravi.

Iz tega sledi naslednji postopek: s h-simpleksom izračunamo parametre hiperravnine, ter z ustreznima polprostoroma poskusimo klasificirati enega od teh dveh učnih primerov. V primeru, da klasifikacija ni pravilna, normalni vektor $\vec{n}$ ter skalar d pomnožimo z (-1). Če želimo, namesto tega obrnemo vrstni red točk v simpleksu, se pa bolj splača kar shraniti parametre hiperravnine v podatkovno strukturo, ki pripada simpleksu, saj jih bomo potrebovali tudi pri klasifikaciji. To operacijo obračanja izvedemo za vse simplekse v kompleksu. S tem smo dobili klasifikacijski simplicialni kompleks.

Ker smo klasifikacijski simplicialni kompleks gradili na podlagi Voronojevega diagrama, nam je zagotovljeno, da je ta orientabilna mnogoterost, ki prostor separira. Težava pa se pojavi, ker smo žarke odrezali in posledično klasifikacijski simplicialni kompleks prostora ne separira več. To popra-



Slika 3.2.: h-simpleks vedno leži na ekvidistantni hiperravnini med dvema sosednjima primeroma različnih razredov, ni pa nujno da seka premico, ki povezuje oba primera. Ta dva primera tudi uporabimo, da pravilno obrnemo hiperravnino, ki pripada simpleksu.

vimo tako, da h-simplekse na robovih, torej ostanke žarkov, implicitno podaljšamo v neskončnost s tem, da klasificiramo z hiperravninami, ki jih določajo sicer omejeni h-simpleksi. Torej, čeprav je h-simpleks omejen, določa neomejeno hiperravnino, ki jo zato uporabljamo za klasifikacijo.

Izmed vseh moramo za nek testni primer izbrati tak h-simpleks, s katerim bomo primer klasificirali prav tako, kot bi ga z uporabo originalnega Voronojevega diagrama. Ker simplicialni kompleks ni sklenjen in ni konveksen, si ne moremo pomagati z enostavnimi algoritmi iz računske geometrije za ugotavljanje položaja točke. Očitna rešitev je najti tako metriko, da nam bo nek dani primer $\vec{p}$ pravilno klasificiral njemu najbližji simpleks.

Geometrijsko gledano je simpleks konveksna množica točk. Izberimo si intuitiven koncept metrike kot razdalje med dano točko $\vec{p}$ in pa najbližjo točko simpleksa. Zdaj se spomnimo izreka, da je projekcija $\vec{p}'$ na hiperravnini izmed vseh točk hiperravnine najbližja $\vec{p}$. Sam h-simpleks je konveksna množica na hiperravnini. Zdaj sta možnosti dve — da projekcija $\vec{p}'$ leži znotraj simpleksa ali pa da ne. V prvem primeru je rešitev oddaljenost $\vec{p}$ od hiperravnine, na kateri leži simpleks. V drugem primeru pa najbližja točka simpleksa leži na kakem od njegovih glavnih lic, ki so $(n-2)$-simpleksi.

Kako pa vemo, da je $\vec{p}'$ znotraj simpleksa? Kot vemo, je konveksna množica, ki ustreza n-simpleksu, presek $n+1$ pozitivnih polprostorov, vsakega od njih pa določa eno izmed glavnih lic n-simpleksa. Hiperravnine, ki pripadajo glavnim licem n-simpleksa, moramo pravilno obrniti in to tako, da v pozitivnem polprostoru leži preostalo oglišče n-simpleksa, tisto, ki ga ni na obravnavanem glavnem licu.

Težava je, da lahko zgornji postopek uporabimo le za n-simplekse, saj lica nižje dimenzije ne vsebujejo dovolj oglišč, da bi lahko določile hiperravnino v n-dimenzionalnem prostoru. Lahko pa zahtevamo, da mora biti hiperravnina vsakega glavnega lica pravokotna na hiperravnino, na kateri leži tak simpleks. Tej zahtevi zadostimo tako, da v posplošenem vektorskem produktu, s katerim bomo računali normalni vektor hiperravnine, med parametre dodamo še normalni vektor hiperravnine.

Na ta način lahko priredimo hiperravnine vsem licem, ne le glavnim: ko se osredotočimo na glavno lice nekega m-simpleksa M, iz množice vektorjev, iz katerih računamo vektorski produkt, odstranimo vektor $\vec{v}_a - \vec{v}_b$, kjer a označuje oglišče, ki smo ga odstranili iz M, b pa označuje eno izmed oglišč, ki ostaja v glavnem licu in je izbrano kot konica za računanje vektorskega produkta. Namesto odstranjenega vektorja na isto mesto postavimo normalni vektor, ki pripada M. Zapišimo še formulo za normalni vektor, ki pripada l-simpleksu $L = \delta\langle\vec{v}_0, \vec{v}_1, \ldots, \vec{v}_l\rangle$ v $\mathbb{R}^n$, ki je glavno lice

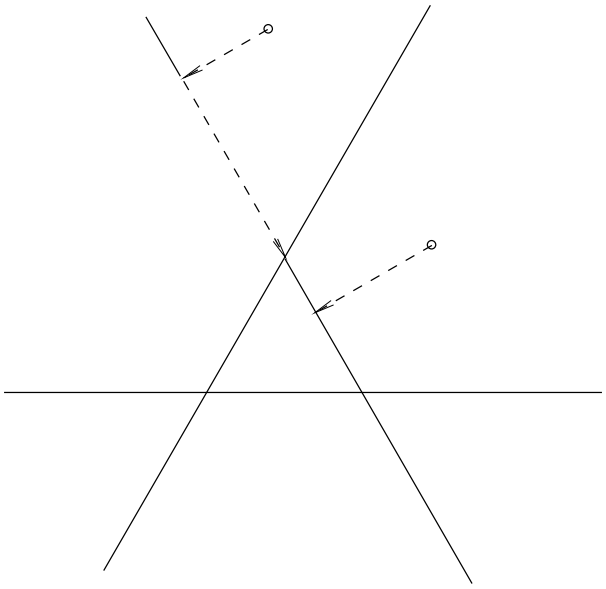
m-simpleksa M, oba pa sta lici h-simpleksa H:

$$\vec{n}_L = \times(\vec{v}_1 - \vec{v}_0, \vec{v}_2 - \vec{v}_0, \ldots, \vec{v}_l - \vec{v}_0, \vec{n}_M, \ldots, \vec{n}_H), |M| = |L| + 1, L \subset M \subset H, |H| = n$$

Očitno je tudi, da se oddaljenost od hiperravnine ne spremeni, če točki prištejemo poljuben mnogokratnik normale $\vec{n}_M$, kar pomeni, da sta točka $\vec{p}$ in njena projekcija $\vec{p}'$ na M enako oddaljeni od vseh hiperravnin lic M, če so te pravokotne na hiperravnino M. To nam zagotavlja zgornja formula za normalni vektor.

Zdaj, ko znamo prirediti polprostore vsakemu podsimpleksu, se vrnimo k problemu računanja razdalje. Vsa glavna lica nekega h-simpleksa predstavljajo kompleks $(n-2)$-simpleksov v $(n-1)$-dimenzionalnem prostoru. Ta problem pa že poznamo! Z rekurzijo, razdalja projekcije $\vec{p}'$ do simpleksa S je razdalja $\vec{p}'$ do najbližjega glavnega lica T. Obe razdalji združimo tako, da uporabimo Pitagorov izrek, saj lahko upoštevamo pravokotnost hiperravnine simpleksa in hiperravnin, ki pripadajo vsem njegovim licem:

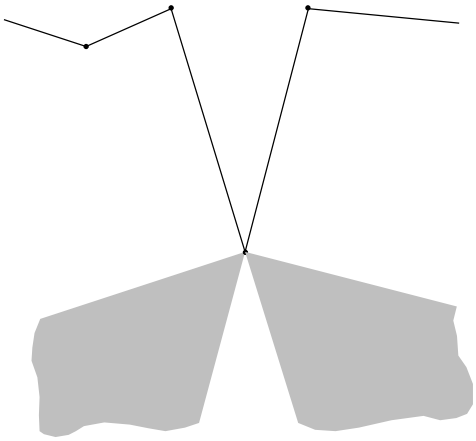
$$d(\vec{p}, S)^2 = d(\vec{p}, \langle \vec{n}_S, d_S \rangle)^2 + \begin{cases} 0 & \text{za } \vec{\mathrm{p}}' \in \mathrm{S} \\ \min_{T \subset S} d(\vec{p}, \langle \vec{n}_T, d_T \rangle)^2 & \text{za } \vec{\mathrm{p}}' \notin \mathrm{S}, |\mathrm{T}| = |\mathrm{S}| - 1, \mathrm{T} \subset \mathrm{S} \end{cases}$$



Slika 3.3.: Rekurzivno računanje razdalje od točke do 2-simpleksa s projekcijami na med seboj ortogonalne hiperravnine. Črtkane puščice označujejo projekcije, simpleks pa je trikotnik med premicami, ki označujejo hiperravnine, na katerih ležijo glavna lica 2-simpleksa.

Kaj pa naredimo, ko je več simpleksov enako oddaljenih od dane točke? Njihove klasifikacije so lahko v primeru ostrega kota med simpleksi namreč različne. Dvoumje rešimo tako, da izberemo za klasifikacijo simpleks, katerega hiperravnina je najdlje od točke. Lahko se zgodi, da ima več enako oddaljenih simpleksov tudi enako oddaljene hiperravnine, vendar se morajo v taki situaciji klasifikacije po vseh takih simpleksih ujemati, sicer imamo izrojen simplicialni kompleks, kjer se različno obrnjeni simpleksi sekajo ali prekrivajo.

Dlakocepsko bi se lahko vprašali, čemu smo toliko komplicirali, da bi na koncu dobili končni rezultat, ki je identičen trivialnemu algoritmu najbližjega soseda. To, da je identičen, predstavlja veliko korist — enostavno lahko preverimo pravilnost algoritmov s tem, da primerjamo rezultate



Slika 3.4.: Točke v označenem območje so enako oddaljene od dveh 1-simpleksov v kompleksu, ki sestavljata konico, a njihove klasifikacije po različnih simpleksih v konici se razlikujejo.

klasifikacije z tistimi, ki jih dobimo z algoritmom najbližjega soseda, saj morajo biti rezultati identični. Pri takem preverjanju ali verifikaciji moramo paziti na nekatere posebne primere. Ko je testni primer enako oddaljen od dveh ali več primerov z različnimi razredi, bo namreč klasifikator na osnovi simplicialnih kompleksov, ki upošteva le dva polprostora, izbral razred, ki pripada pozitivnemu polprostoru, čeprav bi v resnici moral biti neodločen. Sicer je to le posledica naše definicije pozitivnega polprostora. Zaradi stresanja pri računanju Delaunayevega kompleksa lahko pride še do napak, a na zelo omejenih območjih. Take napake lahko vnaprej odkrijemo, saj se zgodijo le takrat, ko več učnih primerov z različnimi vrednostmi razreda leži blizu skupaj (bližje kot je radij stresanja). Take primere lahko označimo z novim razredom, ki označuje dvomljive vrednosti razreda.

Druga prednost generiranja klasifikacijskega simplicialnega kompleksa iz Voronojevega diagrama je v tem, da je ta vedno neizrojen in zato uporaben za klasifikacijo. Simplicialni kompleks lahko spreminjamo z operacijami, ki ohranjajo neizrojenost, kar odpre marsikatero novo možnost. Tretja prednost leži v izraznih možnostih klasifikacijskega simplicialnega kompleksa, saj lahko iz njega dobimo marsikatero uporabno informacijo o domeni.

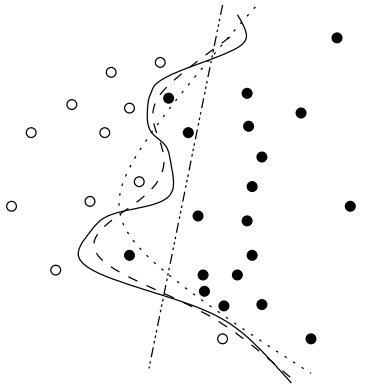
4. Perspektive

V tem razdelku si bomo pogledali možnosti uporabe in razširitve simplicialnih kompleksov v strojnem učenju. Kljub temu, da navedeni algoritmi niso bili preverjeni ali implementirani, so večinoma intuitivno izvedljivi in praktični. Opisi algoritmov so le shematični, saj ne upoštevajo vseh možnosti, ki jih je treba upoštevati, da bi popolnoma pravilno delovali.

4.1. Spremembe simplicialnih kompleksov

Glavna prednost mejnih simplicialnih kompleksov je v enostavnosti njihovega spreminjanja ob ohranjanju uporabnosti za klasifikacijo. Konkretni primeri sprememb simplicialnega kompleksa so odstranjevanje, premikanje in dodajanje oglišč. V nižjih dimenzijah je spreminjanje simplicialnih kompleksov in drugih struktur za aproksimacijo volumnov ali površin aktivno področje raziskovanja na več področjih, spreminjanje v poljubnih dimenzijah pa še ni dovolj raziskano. V tem razdelku bomo predstavili osnovne tehnike, uporabne v poljubno dimenzionalnih prostorih.

V računalniški grafiki je cilj poenostaviti kompleksne modele, ki so tudi predstavljeni z množico h-simpleksov, da bi jih lahko hitreje vizualizirali, ali pa da bi jih s čim manjšo porabo pomnilnika shranili v več različnih stopnjah podrobnosti *(levels of detail)* [55]. Pri tem je predvsem važno, da so poenostavljeni modeli še vedno vizualno čim bolj podobni originalu [33], nekatere metode poenostavljanja pa poskušajo ohraniti tudi topologijo modelov [43]. Stopnjam podrobnosti podoben koncept **večločljivosti** *(multiresolution)* [39] je uporaben za učinkovito obdelavo podatkov, saj lahko pri eni stopnji podrobnosti povzročimo spremembo, ki se potem odraža na vseh stopnjah [37]. Pri analizi podatkov z večločljivostjo lahko uporabljamo le podatke najbolj ustrezne ločljivosti. Če nas za neko topografsko predstavitev zemljišča zanima povprečna nadmorska višina in pa približen obris pokrajine, lahko uporabimo nizko ločljivost, če pa hočemo oceniti čas izkopa temeljev za stavbo ali pripraviti reklamno animacijo, se poslužimo natančnejšega, a počasnejšega modela.



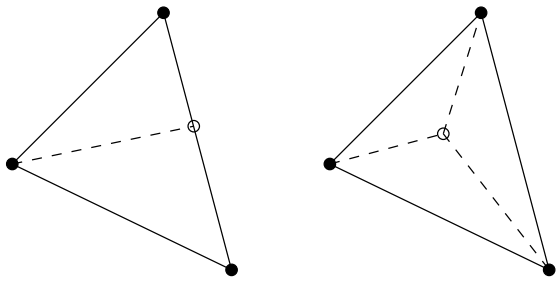
Slika 4.1.: Več stopenj podrobnosti neke krivulje, definirane kot zlepek z 11, 6, 3 in 2 točkama.

Na področjih računalniškega vida in rekonstrukcije modelov ter obrisov *(contours)* poskušajo

množici točk prirediti ploskev [1], na kateri te točke ležijo. V zvezi s tem je predvsem važno ugotavljanje, ali sta dve točki med sabo v strukturi povezani ali ne, ter kako "olepšati" to ploskev, kar lahko pomeni glajenje [48] ali pa poenostavljanje [6]. Konkretni primeri so rekonstrukcija strukture kosti na podlagi več rentgenskih posnetkov in ločevanje posameznih organov iz 3D posnetkov, dobljenih z magnetno rezonanco.

Osnovni princip pri spreminjanju simplicialnih kompleksov je osredotočen na simplekse — v simpleksu lahko ustvarimo novo oglišče in simpleks okoli njega **razbijemo** *(subdivide)* ali subdividiramo, neko oglišče (ki je lice skupno več simpleksom) lahko **premaknemo**, lahko pa tudi **odstranimo** izrojene simplekse. S temi osnovnimi operacijami bomo dosegli skoraj vse, kar bi želeli. Poglejmo si jih bolj podrobno!

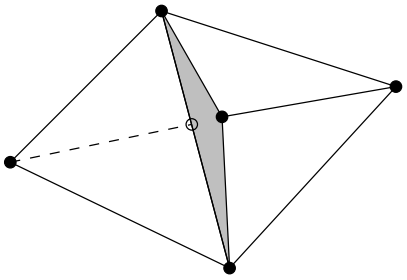
Mogoče najenostavnejša je operacija razbitja. Novo oglišče ustvarimo na poljubni točki znotraj simpleksa. Zdaj ustvarimo nov simpleks za vsako od glavnih lic simpleksa, na katerih novo oglišče ne leži, tako, da ogliščem lica dodamo še novo oglišče. Na koncu odstranimo začetni simpleks. Takrat, ko novo oglišče ne leži na nobenem izmed lic n-simpleksa dobimo $n+1$, ko pa leži na nekem licu, ki je m-simpleks, pa $m+1$ novih simpleksov. V primeru, da bi ustvarili novo oglišče kar na enem izmed obstoječih oglišč, dobimo kar originalni simpleks. Ko razbijemo lice nekega simpeksa znotraj kompleksa, moramo razbiti tudi vse simplekse, ki si delijo to lice.



Slika 4.2.: Razbitje za primer 2-simpleksa — v primerih, da novo oglišče leži na enem od glavnih lic ali pa da ne leži na nobenem.

Operacija odstranjevanja izrojenih simpleksov je malce bolj zapletena, saj včasih simpleksa ne moremo na enostaven način odstraniti, ne da bi njemu sosednje simplekse razbili. Ločiti moramo primer, ko dve oglišči ležita tako blizu skupaj, da ju lahko skrčimo v eno samo *(pair contraction)*. Ta postopek uporablja večina algoritmov za poenostavljanje v računalniški grafiki. Vsi simpleksi, ki vsebujejo obe oglišči, bodo postali izrojeni in se jih lahko na isti način pozneje odstrani. Vsi simpleksi, ki vsebujejo eno od dveh oglišč pa se navežejo na tisto novo. Drugi primer je, ko je simpleks zelo tanek, a so njegova oglišča med seboj oddaljena. V takem primeru simpleks in njemu sosednje primerno razbijemo, tako da ustvarimo bližnje oglišče na glavnem licu, nasproti oglišča, ki ga želimo odstraniti. Nato lahko obe oglišči brez težav skrčimo. To prikazuje slika 4.3. Tudi neizrojene simplekse lahko odstranimo tako, da premaknemo skupaj med seboj najbližji oglišči v simpleksu in jih potem skrčimo, oziroma da vsa oglišča premaknemo skupaj in jih vsa naenkrat skrčimo. Pri tem moramo biti previdni, saj smo uporabili operacijo premikanja oglišč.

Operacija premikanja oglišča je že precej bolj problematična, saj moramo ohranjati lastnosti simplicialnega kompleksa. Upoštevati moramo, da si lahko več simpleksov deli isto oglišče in to povezanost moramo ohranjati. Ne smemo povzročiti, da bi se po premiku dvoje ali več simpleksov sekalo. Nočemo niti pokvariti klasifikacijskih lastnosti kompleksa s tem, da bi nek simpleks obrnili ali ga premaknili v drugo območje. Deloma smo si poenostavili nalogo s tem, da naenkrat premikamo le eno oglišče, ob tem pa ohranjamo vse te lastnosti. Seveda pa lahko za bolj kompleksne operacije sestavimo zaporedje iz osnovnih operacij premikanja, saj bo s tem tudi celotna operacija



Slika 4.3.: Temni (skoraj) izrojeni simpleks bi radi odstranili, vendar moramo prej simpleks na levi razbiti, da bi ohranili simplicialni kompleks.

ohranjala konsistentnost.

Za premikanje rabimo oglišče ter njegov končni položaj. Oglišča ne smemo premakniti kar naenkrat na končni položaj, saj s tem ne bi odkrili možnega prebadanja drugih simpleksov na sami poti. Ob premikanju oglišča premikamo tudi vse simplekse, ki ga vsebujejo kot lice, in ob tem lahko pride do sekanja dveh simpleksov četudi oglišče samo ni trčilo ob noben simpleks. Zato potrebujemo splošen algoritem, ki nam bi nam izračunal oddaljenost med dvema h-simpleksoma. Ob tem se zavedamo tudi, da bo tak algoritem počasnejši kot pa tisti za klasifikacijo, kjer obravnavamo oddaljenost točke od h-simpleksa. K sreči so bili na področju robotike razviti algoritmi, ki ta problem obravnavajo v prostoru s poljubno dimenzijami, na primer GJK [34], za katerega je na voljo tudi učinkovita implementacija Stephena Camerona [19].

Da lahko izvedemo nek premik oglišča $\vec{x}$, se moramo najprej prepričati, da ne pride do sekanja. To naredimo tako, da sestavimo za vsakega od h-simpleksov, katerih lice je oglišče $\vec{x}$, nek **n-simpleks premika**, ki ga sestavljajo oglišča h-simpleksa ter končni položaj oglišča $\vec{x}$. n-simpleks premika združuje množico vseh točk, na katerih je v času premikanja ležal h-simpleks. Zdaj preverimo, ali se s tem n-simpleksom seka kak statični h-simpleks v simplicialnem kompleksu. Sečišče je lahko vsebovano v tistih glavnih licih premikajočih se simpleksov, ki so po premiku ostala statična in so deljena z sosednjimi h-simpleksi v kompleksu — tam se simpleksi v kompleksu vedno dotikajo. Ko pa leži sečišče še kje drugje, premika v celoti ne smemo izvesti. Takrat lahko poskusimo s krajšim premikom v isti smeri, ali pa odmaknemo simplekse, ki nam ležijo na poti. Mogoče je narediti tudi malo bolj zapleten postopek, ki deterministično izračuna maksimalni a še vedno dopustni premik v dani smeri. Ob premiku ohranimo konsistentnost na množici učnih primerov, če v nobenem n-simpleksu premika ni vsebovan noben učni primer.

Pozorni moramo biti tudi na robove simplicialnega kompleksa, kar so tista glavna lica v simplicialnem kompleksu, ki so vsebovana le v enem simpleksu in kjer smo odrezali žarke. Teh lic ne odstranjujemo, oglišča na njih pa premikamo kvečjemu le za s skalarjem pomnoženo vsoto (normaliziranih) normalnih vektorjev simpleksov, v katerih je oglišče vsebovano kot lice. Če se teh dveh pravil (in mogoče še kakega drugega, ki ga tukaj nismo omenili) ne bi držali, bi lahko povzročili nekonsistentnost hipoteze, ki je z zgornjimi postopki sekanja ne bi odkrili. Ta pravila so znana tudi v računalniški grafiki kot ohranjanje robov ali silhuet.

4.2. Podatkovne strukture

To, da simpleks določajo njegova oglišča, že vemo. V prejšnjih razdelkih navedene operacije za spreminjanje in klasifikacijo bi želeli čimbolj učinkovito izvajati. Zelo važno je predvsem poznati med sabo sosednje simplekse, saj potrebujemo podatke o sosednosti za vse zgornje operacije razen

za razbijanje simpleksov. Tudi postopek iskanja najbližjega simpleksa bi lahko optimizirali s poznavanjem sosednosti. Ob tem želimo za vsak h-simpleks v kompleksu poznati njegovo hiperravnino v smislu para $\langle \vec{n}, d \rangle$, saj tega ne bi hoteli večkrat računati. Lahko si zamislimo tudi koristne operacije, ki bi uporabljale podatek, med katerima dvema učnima primeroma leži nek simpleks.

Sosednost med simpleksi temelji na skupnih licih. Posameznemu simpleksu ustreza množica simpleksov N_0, s katerimi si ta deli oglišča. Potem obstaja množica $N_1 \subseteq N_0$ simpleksov, s katerimi si dani simpleks deli 1-lica. Enostavno je videti, da če imata dva simpleksa v skupnem neko m-lice, morata imeti tudi skupna $(m-1)$-lica, saj je m-lice vendar sestavljeno iz $(m-1)$-lic. Končno obstaja množica simpleksov, s katerimi si dani simpleks deli glavna lica, naj bo to $N_{n-1} \subseteq N_{n-2}$. Ob tem moramo omeniti, da vsako glavno lice simpleksa določimo, če odstranimo eno natanko določeno oglišče simpleksa. Torej lahko vsak element N_{n-1} priredimo tistemu oglišču, ki ga moramo odstraniti, da dobimo skupno glavno lice.

Čeprav bi lahko za vsak simpleks eksplicitno navedli vse množice N_i, nam pri simpleksih, ki opisujejo mejne ploskve, zadoščata že N_0 in N_{n-1} — ob premikanju oglišč uporabljamo ravno njiju. V primeru, da nas zanima nek N_i, je naše iskanje omejeno na N_0, kar že prihrani veliko časa. Ob tem pa nam ob vsaki spremembi ni treba shranjevati in obnavljati vseh teh množic.

Ker si vsako oglišče ponavadi deli več simpleksov, se nam splača koordinate oglišč shranjevati posebej in ne podvojevati teh podatkov v vsakem simpleksu. Zato tudi ne potrebujemo N_0 za vsak simpleks, saj je dovolj, da to shranimo v oglišču. S tem smo prišli do naslednjih dveh struktur, ki jih zapišimo v C++:

```
class Vertex {
  ColumnVector      p;      // položaj
  vector<Simplex *> s;      // vsebovanost v simpleksih
}

class Simplex {
  vector<Vertex *>  v;      // oglišča
  vector<Simplex *> sosed;  // N_{n-1}
  ColumnVector      n;      // hiperravnina:
  double            d;
  Case              *a,     // učni primer poz. polprostora
                    *b;     // učni primer neg. polprostora
}
```

Pri postopku izgradnje simplicialnega kompleksa smo ločeno računali oglišča in simplekse. Ko takrat ustvarimo nek nov simpleks, ga registriramo za vsa oglišča v razredu *Vertex* v seznamu s, ob tem pa tudi inicializiramo vsa polja razen *sosed*. Ko končamo z nepovezanim dodajanjem simpleksov in oglišč v seznam, poskrbimo še za povezanost med sosedi. Te za posamezni m-simpleks S dobimo tako, da sestavimo seznam vseh simpleksov, ki vsebujejo kako od oglišč S. Sosednji simpleksi so tisti, ki se v tem seznamu pojavijo m-krat. Vektor *sosed* ima $m+1$ polj, vsako polje z indeksom i kaže na simpleks, ki si z simpleksom S deli vsa oglišča razen oglišča z indeksom i. V primeru, da simpleks preko nekega glavnega lica ni povezan z nobenih drugim simpleksom, je na ustreznem mestu v *sosed* vrednost *NULL*.

4.3. Poenostavljanje

Najbolj očitna korist poenostavljanja klasifikacijskega simplicialnega kompleksa je odstranjevanje nepomembnih, majhnih in izrojenih simpleksov, ki so nastali kot posledice operacije stresanja pri pripravi učnih primerov za gradnjo Delaunayevega kompleksa. Druga uporaba je ustvarjanje večjega števila stopenj podrobnosti, da bi omogočili hitrejše delo s klasifikatorji. Verjetno najvažnejša pa je uporaba poenostavljanja hipotez kot načina za obvladovanje šuma.

Temelj poenostavljanja simplicialnega kompleksa je, da v simpleksu dve izmed njegovih oglišč premaknemo na eno samo mesto in ju zlijemo v novo oglišče. S tem postane en ali več simpleksov izrojenih in jih lahko enostavno odstranimo. Ob tem se lahko vprašamo, kam bi bilo novo oglišče najbolje postaviti. Lahko bi premaknili eno oglišče kar na mesto drugega ali nekam na njuno skupno lice, vendar bi se pri tem spremenilo razmerje med deležem prostora, ki pripada posamezni vrednosti razreda.

Iz tega vidika je ohranjanje razmerja med deležem prostora koristna hevristika: ob vsaki spremembi kompleksa poskušamo ohraniti razmerje med velikostjo območij, ki pripadajo različnim razredom. Ko združimo dve oglišči, moramo novo oglišče premakniti tako, da bo volumen območja, ujetega med prejšnjimi simpleksi in začetnimi simpleksi iz Voronojevega diagrama, enakomerno porazdeljen med obe strani začetnega simplicialnega kompleksa, kar prikazuje slika 4.4. To ujeto območje imenujmo **območje odstopanja**: za testne primere, ki ležijo v njem se namreč rezultat klasifikacije spremeni.

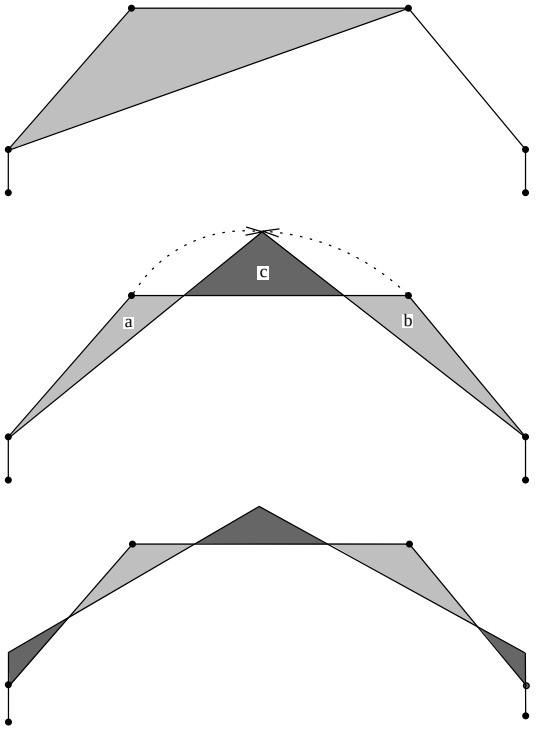
Še boljše rezultate lahko dobimo, če premikamo še sosednja oglišča simpleksov, da bi tako zmanjšali maksimalni odmik oglišč od prejšnjega simplicialnega kompleksa, s tem pa tudi zmanjšamo tudi skupno območje odstopanja, kar je tudi prikazano na sliki 4.4. Seveda ob tem poskušamo spremembe simplicialnega kompleksa zaradi učinkovitosti in lokalnosti omejiti — izogniti se hočemo situaciji, da je premikanje sosednjih oglišč ustvarilo nova območja odstopanja, za uravnoteženje katerih spet premaknemo dodatna oglišča, itd.

Pri poenostavljanju celotnega kompleksa si moramo pripraviti nek vrstni red ali prednostni seznam oglišč, v skladu s katerim jih bomo poskušali pri poenostavljanju odstraniti. Želimo si namreč odstraniti le nepotrebne podrobnosti, ki so značilne za preveliko prileganje učnim primerom, ne pa tudi klasifikacijske točnosti. Smiselni hevristiki sta izbiranje tistih oglišč, ki bodo povzročile minimalno odstopanje ali minimalen volumen območja odstopanja, izognile pa se bodo odstranjevanju oglišč, ki bi povzročila velike spremembe v rezultatih klasifikacije. Obe hevristiki bosta zmanjšali število simpleksov tako, da se bo klasifikacija spremenila le na majhnem območju.

4.3.1. Značilnosti simpleksov

Da bi zgornje hevristike izboljšali, poskusimo posameznim simpleksom pripisati **značilnosti**, na primer:

- število učnih primerov, katerim je najbližji ravno ta simpleks (to je teža simpleksa)
- parametri porazdelitve oddaljenosti učnih primerov od simpleksa
- površina simpleksa
- razmerje med številom učnih primerov različnih razredov za simpleks (na začetku vedno 1:1)
- verjetnostna porazdelitev ocene klasifikacijske napake v odvisnosti od oddaljenosti do simpleksa



Slika 4.4.: Če dve oglišči zlijemo kar na mesto enega, bomo na eni strani simplicialnega kompleksa izgubili, na drugi strani pa pridobili nek volumen, kar prikazuje skica na vrhu in kar ni preveč zaželjeno. Zato moramo novo oglišče premakniti tako, da bo volumen območja c enak vsoti volumnov območij a in b. Da ohranimo konsistentnost, na območjih a, b in c ne sme biti nobenega učnega primera. Kot je prikazano na spodnji skici, lahko premaknemo tudi sosednja oglišča, da bi minimizirali skupni volumen območja odstopanja, ki ga na obeh slikah označujejo obarvana območja.

Ko simplekse odstranjujemo, ustrezno priredimo značilnosti tudi njihovim naslednikom. Značilnosti so nam lahko v pomoč pri poenostavljanju simplicialnega kompleksa, pa tudi pri klasifikaciji. Če poznamo verjetnostno porazdelitev napake, potem lahko ocenimo gotovost naše klasifikacije, saj je ta verjetnostna porazdelitev lokalna aproksimacija porazdelitve klasifikacijske napake v domeni.

Pri poenostavljanju nas zanimajo očitni primeri šuma. Na primer, osamljeni učni primeri v skupinah učnih primerov drugega razreda so ena izmed pogostih posledic šuma. Simpleksi, ki te osamelce ločijo od množice, imajo majhno ploščino in zato majhno pomembnost, zato lahko simplekse s takimi značilnostmi pri poenostavljanju obravnavamo prednostno. Po poenostavitvi se potem pri njih pojavi nesorazmerje v številu pripadajočih učnih primerov za različne razrede. To je ena izmed značilnosti tipičnih za simplekse, ki ločujejo osamljene učne primere od večjega števila sosedov drugega razreda, zato lahko tudi te simplekse pri poenostavljanju obravnavamo prednostno.

Po drugi strani, če je v bližini nekega simpleksa veliko število učnih primerov, mu to da večjo težo, četudi bi odstranitev simpleksa vključevala le majhno območje odstopanja. Zato je smiselno težo pri tem upoštevati. Torej je volumen območja odstopanja (kar smo prej omenili kot možno hevristiko za izbiranje oglišč, ki bi jih radi odstranili) obravnavamo v skladu s težo simpleksa, ki je večja v primeru, da so učni primeri blizu in jih je dosti. Podobno velja tudi za verjetnost napake, ki je nočemo večati: če je klasifikacija s simpleksom nedvoumna, z majhno verjetnostjo napake, bomo

tak simpleks manj spreminjali.

Značilnosti lahko uporabimo tudi pri učenju koristnih hevristik za učenje, o sami hipotezi ali o lastnostih domene. S tem se izzognemo neštetim možnim algoritmom za poenostavljanje, a primernim le v nekaterih primerih. Učimo se, kako enostavno izračunljive značilnosti vplivajo na druge manj enostavno izračunljive. Predmet učenja ali optimizacije je seveda lahko katerikoli parameter, tudi metrika.

Poglejmo si verjetnostno porazdelitev napake za nek simpleks: v primeru, da simpleks po poenostavitvi ni več konsistenten z učnimi primeri, lahko to upoštevam pri porazdelitvi napake. Odkrili pa bi lahko možno posredno povezavo med večjo težo simpleksa in manjšo verjetnostjo napake, na podlagi česar bi lahko pri poenostavljanju prednostno obravnavali simplekse z manjšo težo.

Končno lahko uporabimo značilnosti za aktivno učenje. Možna hevristika bi lahko predlagala poskuse v okolici tistih simpleksov, ki imajo veliko verjetnost napake, veliko ceno zmote, majhno težo ter veliko ploščino. Članek [38] na primer omenja hevristiko na osnovi Voronojevih diagramov in trdi, da so primerni položaji za nove eksperimente ravno v Voronojevih ogliščih. Če to hevristiko prilagodimo simplicialnim kompleksom, bi bili primerni položaji za ostrenje točnosti hipoteze na samem klasifikacijskem simplicialnem kompleksu. Seveda obstajajo tudi druge smiselne hevristike za aktivno učenje.

4.3.2. Jedra

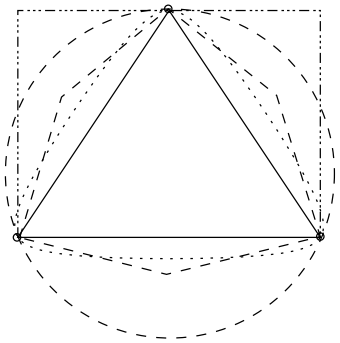
Za nekatere domene zaradi predznanja vemo, da v njih obstajajo zakonitosti, ki nimajo nujno enakih lastnosti kot simplicialni kompleksi. Primer take zakonitosti je pričakovana zvezno odvedljiva mejna ploskev. V takih primerih lahko na simplekse kot na ogrodje položimo primernejšo krivuljo ali ploskev, ob tem pa simplekse uporabimo za izračun parametrov ploskve. Če vemo, da domena temelji na oddaljenosti od neke točke, lahko na simplekse položimo krožne ali elipsoidne izseke. Če vemo, da so atributi med seboj neodvisni, ali če želimo simplicialni kompleks aproksimirati z običajnim odločitvenim drevesom, lahko uporabimo tudi s koordinatnim sistemom poravnane ploskve. Če vemo, da je mejna ploskev gladka, lahko uporabimo zlepke.

Tip tako uporabljene ploskve označujemo z izrazom **jedro** *(kernel)*. Kot jedro lahko pravzaprav uporabimo poljubno ploskev, parametri katere se lahko izračunajo neposredno iz nekega dela simplicialnega kompleksa. Če bi na primer kot ploskev hoteli uporabiti le en polinom nekega določenega reda, bi za izračun parametrov, katerih število ustreza redu polinoma, uporabili celoten simplicialni kompleks. V primeru zlepkov z lokalnimi lastnostmi računamo parametre zlepka le s posameznim simpleksom ter njegovimi sosedi.

Jedro se splača uvajati potem, ko smo simplicialni kompleks že poenostavili. Seveda lahko simplicialni kompleks po uvedbi jedra še dodatno poenostavimo, saj nam primerno jedro omogoča konsistentne hipoteze z manjšim številom simpleksov. Pri poenostavljanju po uvedbi jedra uporabljamo podobne postopke, kot smo jih pred tem: poenostavljanje s premikanjem oglišč, ob ohranjanju konsistentnosti in razmerja med volumnom območij, ki pripadajo različnim razredom.

4.3.3. Occamova britev

Šum lahko obravnavamo tako, da že na samem začetku zgradimo drugačno strukturo, iz katere potem izluščimo klasifikacijski simplicialni kompleks. Voronojeve diagrame lahko posplošimo z uvedbo koncepta reda, kar lepo opisuje [56]. Navaden Voronojev diagram je reda 1, kar pomeni, da vsako Voronojevo območje ustreza natanko enemu položaju. Vsako območje v Voronojevem diagramu reda k pa ustreza neki kombinaciji k položajev. Območja so ravno tako konveksna. Tudi z Voronojevimi diagrami poljubnega reda lahko zgradimo klasifikacijski simplicialni kompleks



Slika 4.5.: Uvajanje različnih jeder na podlago trikotnega simplicialnega kompleksa: krožno, ortogonalno ter Bezierovo jedro in pa aproksimacija zlepka z razbitim simplicialnim kompleksom.

tako, da vsakemu območju pripišemo razred, ki ga ima večina primerov, katerim območje pripada. Rezultati pri klasifikaciji s takimi simplicialnimi kompleksi bodo identični metodi k-NN. Znano je, da k-NN dosega v povprečju boljše rezultate od 1-NN v domenah, kjer je veliko šuma. Še ena smiselna možnost za obravnavanje šuma z glasovanjem temelji na posplošitvi same klasifikacije: ob klasifikaciji upoštevamo klasifikacije k najbližjih simpleksov testnemu primeru.

Sicer vemo, da s poenostavljanjem dosežemo odpornost proti šumu, vendar moramo poenostavljanje nekje ustaviti, saj bi sicer začeli izgubljati in ne pridobivati klasifikacijske točnosti. Večina metod, ki temeljijo na sicer filozofski Occamovi trditvi, poskuša na različne načine določiti stopnjo optimalnega obsega poenostavljanja. Verjetno najbolj popularen postopek, ki temelji na Occamovi britvi, je Rissanenov MDL *(minimum description length)* [60]. Ta pravi, da je hipoteza najboljša (in s tem tudi obseg poenostavljanja) takrat, ko pridemo do **minimalnega opisa** hipoteze ter učnih primerov, izraženih ob upoštevanju hipoteze. Prednost uporabe principa MDL je v tem, da nam ni treba ocenjevati pričakovane kompleksnosti domene, ker za to poskrbi metoda sama. Mimogrede, v večini literature se v obravnavi principa MDL namesto izraza hipoteza uprablja izraz model, mi pa terminologije ne bomo spreminjali samo zaradi tega — naš model je sicer tako splošen, da je naša hipoteza skoraj enakovredna modelu pri drugih metodah, samega modela pa tako ali tako ne spreminjamo. Sama hipoteza poleg simplicialnega kompleksa vsebuje še nekaj parametrov, ki jih moramo posebej upoštevati pri izračunu kompleksnosti, na primer tip jedra.

Že prej smo brez kakšnega posebnega reda omenili več hevristik pri poenostavljanju in zdaj jih bomo poskušali združiti v celoto tako, da bomo poenostavljanju postavili kriterije in omejitve. Ko smo drugod iskali najboljšo hipotezo po kriteriju največje klasifikacijske točnosti, lahko namesto tega uporabimo kot merilo kar dolžino minimalnega opisa, ki v sebi posredno vključuje tudi klasifikacijsko točnost. Da torej najdemo optimalno hipotezo po principu MDL, na poljuben način iščemo hipotezo ter ocenjujemo kompleksnost opisa. To iskanje ni nič drugega kot optimizacija ob spreminjanju parametrov, na primer metrike, jedra, stopnje poenostavljanja, itd. Še posebej zanimiv je način za iskanje optimalne metrike — namreč, boljša metrika bo omogočila krajši opis hipoteze. S konkretnimi algoritmi za optimizacijo se tukaj ne bomo ukvarjali. Za uporabo MDL je bistveno le definirati, kako se bo računala dolžina opisa hipoteze. Tu lahko predpostavimo, da nam v opis ni treba vključiti atributov učnih primerov, saj jih že imamo podane — sicer pa bi bile vse hipoteze pri tem enako uspešne, saj se same ne ukvarjajo s predstavitvijo atributov.

Definirajmo način opisa klasifikacij množice učnih primerov, s čemer bomo razlikovali med hipotezami različnih stopenj konsistentnosti. Pomembno je, da je dolžina te komponente opisa uravnotežena z dolžino opisa same hipoteze. S slabo izbranim načinom računanja kompleksnosti lahko

napačne klasifikacije opisujemo z relativno kratkimi kodami. S tem bi imeli nagnjenje preveč poenostaviti hipotezo. Zato je mogoče boljše, če namesto klasifikacij učnih primerov raje kodiramo za vsak simpleks verjetnostno porazdelitev ocene napake, ki je lahko izračunana iz napačno klasificiranih učnih primerov in verifikacijske podmnožice učnih primerov ali pa posredno ocenjena na podlagi značilnosti, kot smo to opisovali prej — v taki oceni lahko na podlagi značilnosti upoštevamo tudi zelo pomembno statistično signifikantnost. Če je simpleks negotov, če z veliko verjetnostjo pričakuje napako in je slabo ocenjen, bo imela verjetnostna porazdelitev na podlagi entropije dolg opis, sicer pa kratkega. Tako smo z MDL v eno samo številko združili signifikantnost in klasifikacijsko točnost.

Posamezen simpleks torej znamo kodirati, kaj pa kodiranje celotnih simplicialnih kompleksov? Vsakemu simpleksu lahko pripišemo neko fiksno dolžino opisa, kar bo povzročilo, da se bodo ob optimizaciji najprej odstranjevali majhni simpleksi. V dolžino opisa lahko vključimo uravnoteženost — simpleksi, ki so ob poenostavljanju kršili zahteve o uravnoteženosti volumna območja odstopanja, bodo opisani z daljšimi kodami. V dolžini opisa lahko upoštevamo zaželjeno spreminjanje kotov v odvisnosti od sosednjih simpleksov, s čemer prilagodimo pristranskost glede na gladkost oziroma oglatost simplicialnega kompleksa, in še marsikaj drugega. Težava pri veliki fleksibilnosti je počasnost učenja. Bolj praktična enostavna alternativa bi delovala tako, da na podlagi predznanja predpišemo smiselne metode poenostavljanja (na primer hevristike ohranjanja razmerja med volumni, ki pripadajo različnim razredom), potem pa ocenjujemo kompleksnost s fiksno dolžino opisa simpleksa.

Radi bi se tudi izognili nerealnim modelom in hipotezam, saj so ti lahko enostavno opisljivi v simbolični predstavitvi. To je eden izmed glavnih argumentov nasprotnikov MDL. Iz te zagate se lahko rešimo tako, da se ne ukvarjamo kratkovidno le z izbrano domeno, temveč upoštevamo tudi njej podobne domene in predznanje ter konservativno obnašamo. Nagibamo se k temu, da v izbrani domeni raje uporabimo tiste metode, ki so bile potrjene kot učinkovite tudi pri podobnih domenah. To realiziramo tako, da uspešnim hevristikam in lastnostim, ki se pogosto izkažejo kot pravilne, določimo relativno krajši opis v primerjavi z ostalimi hevristikami in lastnostmi. Kljub temu pa se bo še vedno splačalo izbrati nek popolnoma nov model, če se ta izkaže kot bistveno bolj učinkovit in ga potrjuje veliko število primerov, saj bo to odtehtalo naše konservativno nagibanje k preizkušenim rešitvam. Končno pa si lahko stopnjo konservativnosti izberemo sami.

Pristop k MDL smo opisali zelo na splošno. Za praktično uporabnost ga je treba bistveno bolj natančno formulirati. Čeprav teoretično ni ovir, da ne bi mogli znotraj ene same hipoteze upoštevali več različnih jeder, metrik, ipd., bi bili v praksi taki algoritmi računsko precej neučinkoviti, kljub potencialno boljšim rezultatom. Zato je včasih boljše omejiti fleksibilnost opisa in se zanašati na specifične, neposredno programirane hevristike.

4.3.4. Razumljivost

Ena izmed prednosti MDL je tudi ta, da je enostavna hipoteza ponavadi bolj razumljiva tudi človeku. Kriterije za ocenjevanje kompleksnosti hipoteze lahko prilagodimo še za ta namen, ne le za doseganje klasifikacijski točnosti. Če nam je pomembna razumljivost, iščemo najbolj informativno hipotezo za dano domeno v dveh ali treh dimenzijah. Ko jo dobimo, lahko na podlagi nje ugotovimo, iz katerega pogleda bo hipoteza najbolj razumljiva človeku. Do hipotez nižjih dimenzij lahko pridemo s poljubnimi projekcijami atributov v podprostor nižje dimenzije, ne le z običajnim zmanjševanjem števila atributov. Barvna slika B.2 na strani 45 prikazuje, kako se poveča razumljivost domene, če le nanjo gledamo iz primerne smeri, tudi če ne povečamo števila atributov.

Sam postopek za iskanje informativnega pogleda na hipotezo bi deloval tako, da najprej v višji dimenziji najdemo učinkovito hipotezo. Na primer, domena na omenjeni sliki v treh dimenzijah ob upoštevanju vseh atributov ne povzroča učnemu algoritmu nobenih težav. Ko dobimo hipotezo

v obliki simplicialnega kompleksa, lahko poiščemo tak pogled nanj, da bo simplicialni kompleks izgledal čimbolj tanek. To nam bo omogočilo, da ga informativno prikažemo v nižji dimenziji kot črto v dveh ali ploskev v treh dimenzijah, ne da bi ob tem povzročili nekonsistentnost. Seveda si možne tudi drugačne hevristike za iskanje razumljivega pogleda.

Omenimo še to, da so značilnosti simpleksov, izračunane, ocenjene ali naučene, kot smo jih prej opisali, lahko zelo uporabno orodje strokovnjakom za analizo domene.

4.4. Učinkovitost

Glavna težava klasifikacije s simplicialnimi kompleksi je eksponentno povečevanje kompleksnosti z dimenzijo. Po [27] je kompleksnost izgradnje Voronojevega diagrama linearna za neko fiksno dimenzijo, torej $O(n)$, kjer je n število položajev. Želeli pa bi primerjati, koliko bolj kompleksen je postopek v višjih dimenzijah. Naj bo $T_d(n)$ čas, ki ga porabimo za izračun nekega Voronojevega diagrama za n položajev v d dimenzijah. Naj bo $T_2(n) = 2$. Potem je po [27] $T_3(n) \approx 6.77$, $T_4(n) \approx 31.78$. Vidimo torej, da se konstanta eksponentno povečuje z dimenzijo.

Trenutno najnižja Chazellova zgornja meja za kompleksnost računanja konveksne ovojnice je $O\left(n \log n + n^{\lfloor \frac{d}{2} \rfloor}\right)$, kjer je d dimenzija prostora [35, 4]. Kompleksnost same konveksne ovojnice v smislu števila stranic je $O\left(\frac{n^{\lfloor d/2 \rfloor}}{\lceil d/2 \rceil !}\right)$. V praksi program QHull v prostorih dimenzije večje od 5 z nekaj sto učnimi primeri porabi več kot 10^8 zlogov RAM, da sploh deluje. Tudi število oglišč hiperkocke, ki smo jo uporabili za prirezovanje žarkov, raste eksponentno s številom dimenzij. Na današnji strojni opremi je zato postopek izgradnje klasifikacijskih simplicialnih kompleksov omejen na 5 do 8 atributov.

4.4.1. Podproblemi

Včasih so v neki domeni med seboj odvisni le nekateri atributi in le take, med seboj odvisne atribute, nam je treba hkrati upoštevati pri gradnji simplicialnega kompleksa. To nam pomaga zmanjšati največje število dimenzij. V primeru, da so atributi med seboj neodvisni, nam klasifikacija s simplicialnimi kompleksi ne prinaša nobenih prednosti pred enostavnejšimi metodami, ki predpostavijo neodvisnost atributov. Take metode ponavadi kot mejne ploskve obravnavajo le hiperravnine, poravnane z osjo koordinatnega sistema, torej tiste, katerih normalni vektorji so bazni vektorji kanonične baze.

Že prej smo omenili ugotovitev, da se kompleksnost hipoteze lahko zmanjša, klasifikacijska točnost pa poveča, če vključimo vanjo dodatne atribute. Na tej podlagi lahko izdelamo postopek za pohlepno *(greedy)* iskanje najboljše kombinacije atributov. Začnemo z enim atributom in eno-dimenzionalnim klasifikacijskim problemom ter za vse posamezne atribute ustvarimo optimalno hipotezo. Zdaj izberemo tistega, ki je dal po kriteriju MDL najboljšo hipotezo, kar pomeni, da ima najboljšo kombinacijo klasifikacijske točnosti ter kompleksnosti opisa, torej je ta atribut za domeno najbolj informativen. Za vsakega od preostalih atributov ustvarimo nov dvo-dimenzionalni klasifikacijski problem tako, da atributu, izbranem v prvem krogu, dodamo posamezno vsakega od preostalih atributov. Spet izberemo najboljšo hipotezo in postopek ponavljamo, dokler ne dosežemo največjega še sprejemljivega števila atributov in bi ob nadaljevanju postal postopek prepočasen, ali pa ko dosežemo zadovoljivo klasifikacijsko točnost. Sam pohlepni algoritem lahko brez večjih sprememb izboljšamo tako, da uporabimo iskanje v žarku *(beam search)*. S tem algoritmom lahko pridemo tudi do podatkov o odvisnosti atributov — neodvisni atributi ob sočasnem upoštevanju ne bodo dali boljših rezultatov, kot če bi jih obravnavali posebej. Ta postopek lahko kombiniramo z zgoraj omenjenim postopkom projekcijskega urejanja.

Možno je tudi učenje z dekompozicijo [72], tako da problem razbijemo na več podproblemov, kjer so v posameznemu podproblemu atributi med seboj odvisni, velikost posameznega podproblema pa je po njihovem številu čim manjša. Podprobleme lahko gradimo z zgoraj opisanim pohlepnim postopkom. Posamezen podproblem rešujemo s simplicialnimi kompleksi, rezultate podproblemov pa lahko obravnavamo in združujemo tudi s simboličnimi metodami. Tudi pri izgradnji odločitvenih dreves bi namesto enostavnih primerjav ($>, <$) lahko uporabljali splošnejše klasifikacijske simplicialne komplekse.

4.4.2. Pohitritev klasifikacije

Nekateri algoritmi, opisani v prejšnjih razdelkih bolj kot ilustracije, so v praksi zelo neučinkoviti, še posebej klasifikacija. Klasifikacijo lahko pospešimo z usmerjenim iskanjem po kompleksu, z izogibanjem redundantnim izračunom ter z večločljivostjo. Če smo se pripravljeni posloviti od simplicialnih kompleksov, lahko uporabimo postopek, ki temelji na razbitju prostora *(spatial subdivision)*, na podlagi klasifikacijskega simplicialnega kompleksa ustvarimo podatkovno strukturo, podobno odločitvenemu drevesu.

Ob računanju oddaljenosti točke od simpleksov velikokrat izračunamo parametre istih hiperravnin. Vemo namreč, da je večina lic v simplicialnem kompleksu vsebovana v več simpleksih v kompleksu in tudi znotraj istega simpleksa si eno lice med seboj deli veliko glavnih lic. V našem rekurzivnem postopku znotraj enega samega simpleksa večkrat izračunamo parametre istega lica. Odveč je sploh omenjati, da tudi za vsak nadaljnji testni primer vse te izračune ponovimo. Izračunane parametre bi lahko shranjevali v posebno podatkovno strukturo, organizirano kot predpomnilnik. Ob klasifikaciji bi za optimalni izkoristek predpomnilnika raje skupaj klasificirali podobne in bližnje kot pa različne in oddaljene testne primere.

Usmerjeno iskanje lahko realiziramo z iskanjem v širino. Vemo namreč, da se z rekurzijo v lica simpleksa oddaljenost od primera kvečjemu poveča v primerjavi z oddaljenostjo od hiperravnine, na kateri leži simpleks. Vozlišča iskalnega drevesa stanj torej ustrezajo simpleksom, h-simpleksi so vozlišča na prvem nivoju, poddrevesa nekega vozlišča so glavna lica tistega simpleksa, ki ustreza vozlišču, listi drevesa ustrezajo ogliščem, cene premika pa temeljijo na povečani razdalji točke do projekcije na glavno lice v primerjavi z razdaljo do projekcije na hiperravnino simpleksa. Najbližji simpleks lahko torej iščemo tako, da najprej izračunamo oddaljenost testnega primera do hiperravnin, na katerih ležijo posamezni h-simpleksi. Nato se spustimo v najbližji simpleks in če projekcija leži znotraj simpleksa smo že dospeli na cilj po najkrajši poti. Če nismo, izračunamo razdaljo do vseh njegovih glavnih lic ter ponovimo postopek za trenutno najbližje vozlišče oziroma simpleks.

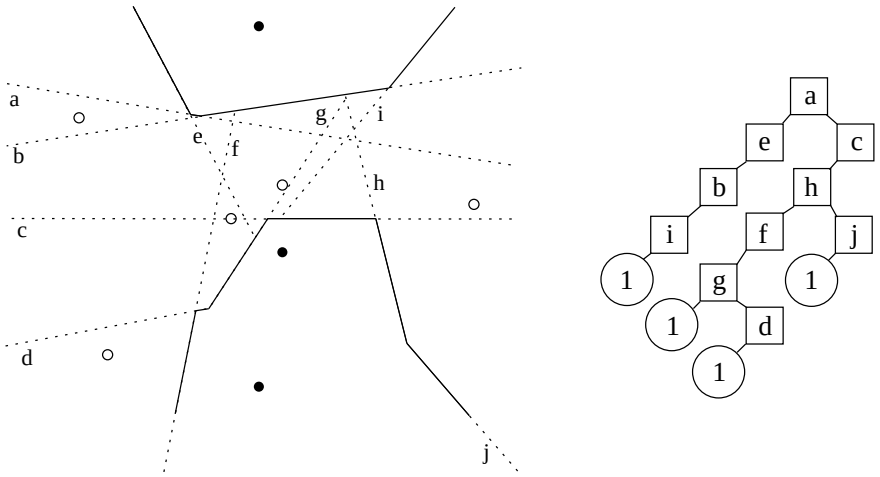
Najbližji simpleks lahko najdemo tudi z usmerjenim sprehajanjem po strogo konveksnem ter povezanem simplicialnem kompleksu. Naključno si izberemo nek simpleks ter ga dodamo v prej prazno množico simpleksov, ki jih želimo obiskati. Vzamemo enega izmed simpleksov v množici ter preverimo, če obstajajo njemu sosednji in še neobiskani simpleksi, ki so enako blizu ali bližje kot izbrani simpleks. V primeru, da taki simpleksi obstajajo, v množico simpleksov, ki jih hočemo obiskati, dodamo vse tiste simplekse, ki so enako blizu kot najbližji. Potem iteracijo ponovimo za vse simplekse v množici. Če je množica prazna, je trenutni simpleks tudi najbližji.

Ko simplicialni kompleks ni konveksen, ga lahko vedno razbijemo na več podkompleksov, sestavljenih iz med seboj konveksnih in povezanih simpleksov. S postopkom usmerjenega sprehajanja za vsak podkompleks najdemo najbližji simpleks, nato pa med najdenimi izberemo najbližjega za celoten simplicialni kompleks. Obstajajo tudi druge podatkovne strukture, primerne za hitro iskanje najbližjega simpleksa, na primer večdimenzionalna iskalna drevesa *(multidimensional search trees)* [56], med katerimi so najbolj poznana k-D drevesa, ki bi se jih dalo prilagoditi za iskanje najbližjega simpleksa.

Za hitrejšo klasifikacijo lahko razbijemo prostor na območja, ki jim pripadajo različni razredi. Razbitje bi lahko ustrezalo izgradnji posplošenega Voronojevega diagrama, kjer so položaji individualni simpleksi, vendar pa obstajajo enostavnejši in bolj učinkoviti postopki. Popularen način razbitja prostora so drevesna razbitja s polprostori. Podatkovna struktura, ki temu ustreza, je v računalniški grafiki poznana kot BSP drevo *(binary space partitioning tree)* [29], v strojnem učenju pa kot perceptronsko odločitveno drevo (*perceptron decision tree*) [63]. Vsakemu vozlišču v drevesu ustreza nek h-simpleks iz klasifikacijskega simplicialnega kompleksa. Klasifikacija nekega testnega primera poteka tako, da se po drevesu sprehajamo, dokler ne pridemo do lista, kjer je zapisan razred. Če testni primer leži v negativnem polprostoru nekega simpleksa, stopimo v levo poddrevo, sicer v desnega. Pri taki strukturi sicer ne izvemo, kateri simpleks je najbližji. Sam algoritem za gradnjo ni zapleten, a presega okvire tega besedila. Bralec naj se obrne na literaturo, na primer na [29]. V primeru, da uporabljamo ortogonalno jedro, bi lahko na ta način izdelali tudi običajno odločitveno drevo. Na podlagi simplicialnega kompleksa bi lahko tako določili tudi topologijo in uteži ustreznih nevronskih mrež.

Ob tem je mogoča tudi nasprotna pot — iz odločitvenega drevesa in hipotez drugih modelov lahko ustvarimo simplicialni kompleks, ki ga lahko z navedenimi metodami poenostavimo, obdelamo, analiziramo, ter po potrebi novo hipotezo nazaj pretvorimo v odločitveno drevo. Končno smo prej kot osnovno metodo opisovali izgradnjo simplicialnega kompleksa na podlagi modela 1-NN.

Podobna zanimiva možnost bi bila na podlagi simplicialnega kompleksa ustvariti množico klasifikacijskih primerov. Ti primeri morajo biti razpostavljeni tako, da dobimo identične rezultate, če na njih uporabimo algoritem najbližjega soseda 1-NN, ali pa če klasificiramo neposredno s simplicialnim kompleksom. Intuitivno jasno je, da to lahko vedno storimo do poljubne natančnosti, če le vzamemo dovolj klasifikacijskih primerov. Prednost tega pristopa je predvsem v tem, da za 1-NN že obstaja veliko učinkovitih algoritmov.



Slika 4.6.: Klasifikacijski simplicialni kompleks na levi lahko pretvorimo v BSP drevo. Polnim krogcem ustreza razred 1, praznim pa 0. Vsakemu simpleksu ustreza hiperravnina, ki je označena s črko in je pravilno obrnjena. Iz hiperravnin lahko sestavimo klasifikator izražen z BSP drevesom na desni. Na tem primeru smo označili le liste, ki pripadajo razredu 1, ki tudi ustreza negativnemu polprostoru. V primeru, da bi se hoteli spustiti po desnem poddrevesu, ki v vozlišču ne obstaja, je pravilna klasifikacija razred 0.

5. Zaključek

Čeprav je koncept mejnih ploskev v strojnem učenju zelo razširjen in uveljavljen, do sedaj ni bilo poglobljenega dela na predstavitvi in obdelavi eksplicitno predstavljenih mejnih ploskev poljubne kompleksnosti. V tem besedilu smo poskusili predstaviti algoritme in podatkovne strukture, nujne za učenje in uporabo klasifikatorjev, ki temeljijo na geometrijski predstavitvi mejnih ploskev s simplicialnimi kompleksi.

Prednosti strojnega učenja s simplicialnimi kompleksi ležijo predvsem v fleksibilnosti ter enostavnosti za nadaljnjo obdelavo, to pa zato, ker je dobljena metoda hkrati zagnana, lokalna in nehierarhična. Te tri lastnosti olajšajo poenostavljanje hipotez, različne načine predstavitve hipotez, raziskovanje domen ter lokalno spreminjanje hipotez. To omogoči uporabnost te tehnike tudi na področjih aktivnega učenja, učenja z ojačanjem ter celo vizualizacije, hkrati pa tudi večjo uspešnost pri domenah z odvisnimi zveznimi atributi.

Seveda pa obstajajo tudi slabosti. Pri domenah z veliko šuma bodo na začetku imele lokalne metode, ki delujejo od spodaj navzgor, težave z zelo kompleksnimi hipotezami, katerih poenostavljanje je lahko dolga in počasna operacija. Rešitev bi lahko iskali predvsem v drugačnih metodah izgradnje simplicialnih kompleksov, ki že na začetku upoštevajo šum. Kompleksnost simplicialnih kompleksov narašča z njihovo dimenzijo, torej s številom atributov, zato omenimo postopka dekompozicije ter iterativnega dodajanja atributov. Ta postopka lahko poenostavita problem in zato izboljšata učinkovitost.

Dosedanje delo odpira vrsto možnosti za nadaljnje izboljšanje opisanih postopkov. Predvsem je bistveno implementirati metodo tako, da bo neposredno praktično uporabna pri resničnih problemih. SICC trenutno dosega rezultate, ki so enaki tistim, ki so pridobljeni z metodo najbližjega soseda. Ta metoda je sicer po klasifikacijski točnosti med najbolj uspešnimi, vendar pa da bi te rezultate presegli, bi se bilo najbolje osredotočiti na izvedbo ocenjevanja kompleksnosti hipotez, mogoče na podlagi pristopa MDL, ki je nakazan v tem besedilu. Bolj zaokroženo je nadaljnje delo na izboljšanju učinkovitosti algoritmov ter integracija z drugimi metodami, kar bi doprineslo tudi k bolj učinkoviti obravnavi domen z veliko atributi. Končno bi bilo smiselno uporabiti prednosti eksplicitnega zapisa hipotez na področjih analize domen, kar je bilo opisano v prejšnjih odstavkih.

Literatura

[1] N. Amenta, M. Bern, and D. Eppstein. The crust and the β-skeleton: Combinatorial curve reconstruction. Research Report, Xerox PARC, 1997.

[2] N. Amenta, M. Bern, and D. Eppstein. Optimal point placement for mesh smoothing. *Journal of Algorithms*, 30(2):302–322, February 1999.

[3] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, September 1991.

[4] Chazelle B. An optimal convex hull algorithm in any fixed dimension. *GEOMETRY: Discrete & Computational Geometry*, 10, 1993.

[5] Dasarathy B. and White L.J. A characterization of nearest neighbor rule decision surfaces and a new approach to generate them. *Pattern Recognition*, 10:41–46, 1978.

[6] Chandrajit Bajaj and Daniel Schikore. Error-bounded reduction of triangle meshes with multivariate data. *SPIE*, 2656:34–45, 1996.

[7] Chandrajit L. Bajaj and Daniel R. Schikore. Topology preserving data simplification with error bounds. *Computers & Graphics*, 22(1):3–12, February 1998. ISSN 0097-8493.

[8] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The Quickhull algorithm for convex hull. Technical Report GCG53, Geometry Center, Univ. of Minnesota, July 1993.

[9] Bruce G. Baumgart. Winged Edge Polyhedron Representation. Artificial Intelligence Project Memo AIM-179 (CS-TR-74-320), Computer Science Department, Stanford University, Palo Alto, CA, October 1972.

[10] Bruce G. Baumgart. A Polyhedron Representation for Computer Vision. In *Proceedings of the National Computer Conference*, pages 589–596, 1975.

[11] Hilan Benususan. God doesn't always shave with Occam's razor – learning when and how to prune. In C. Nedellec and C. Rouveirol, editors, *Proceedings of the 9th European Conference on Machine Learning*, Chemnitz, Germany, April 1998. Springer-Verlag.

[12] Marshall Bern, Paul Chew, David Eppstein, and Jim Ruppert. Dihedral bounds for mesh generation in high dimensions. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 189–196, San Francisco, California, January 1995.

[13] Marshall Bern, David Eppstein, and John Gilbert. Provably good mesh generation. *Journal of Computer and System Sciences*, 48(3):384–409, June 1994.

[14] B. K. Bhattacharya. Application of computational geometry to pattern recognition problems. Technical Report 82-3, School of Computing Science, Simon Fraser University, 1982. (McGill Ph.D. Thesis).

[15] B. K. Bhattacharya. An algorithm for computing order k Voronoi diagrams in the plane. In *ALLERTON: 22th Annual Allerton Conference on Communication, Control, and Computing*. Allerton House, Monticello, Illinois, 1984.

[16] B. K. Bhattacharya, Ronald S. Poulsen, and Godfried T. Toussaint. Application of proximity graphs to editing nearest neighbor decision rule. In *International Symposium on Information Theory*, Santa Monica, CA, 1991.

[17] L. Bottou and V. Vapnik. Local learning algorithms. *Neural Computation*, 4(6):888–900, 1992.

[18] B. Büeler, A. Enge, K. Fukuda, and H.-J. Lüthi. Exact volume computations for polytopes: a practical study. In *Abstracts 12th European Workshop Comput. Geom.*, pages 57–64, 1996.

[19] S. Cameron. A comparison of two fast algorithms for computing the distance between convex polyhedra. *IEEE Trans Robotics and Automation*, 13(6):915–920, 1997.

[20] Stephen Cameron. Enhancing GJK: Computing minimum and penetration distances between convex polyhedra. *Int Conf Robotics and Automation*, April 1997.

[21] David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan. Active learning with statistical models. Technical Memo AIM-1522, Massachusetts Institute of Technology, Artificial Intelligence Laboratory.

[22] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IT*, 13(1):21–27, January 1967.

[23] D. Cubanski and D. Cyganski. Multivariate classification through adaptive Delaunay-based C-0 spline approximation. *PAMI*, 17(4):403–417, April 1995.

[24] B. V. Dasarathy. All you needed to know about the neighbors (but never could figure out how!) recognition under partial exposure and imperfect supervision. 1979.

[25] T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 22(1), 1998.

[26] Harris Drucker, Chris J. C. Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. Support vector regression machines. In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 155. The MIT Press, 1997.

[27] R. A. Dwyer. Higher-dimensional Voronoi diagrams in linear expected time. *Discrete Comput. Geom.*, 6:343–367, 1991.

[28] Carl Erikson. Polygonal simplification: An overview. Technical Report TR96-016, Department of Computer Science, University of North Carolina - Chapel Hill, February 16 1996. Wed, 26 Jun 1996 18:07:48 GMT.

[29] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, 1990. second edition.

[30] Lori A. Freitag and Carl Ollivier-Gooch. A comparison of tetrahedral mesh improvement techniques. In *Proceedings of the Fifth International Meshing Roundtable*, pages 87–100, Pittsburgh, Pennsylvania, October 1996.

[31] Lori A. Freitag and Carl Ollivier-Gooch. Tetrahedral mesh improvement using face swapping and smoothing. *International Journal for Numerical Methods in Engineering*, 40, 1997.

[32] A. Gammerman, V. Vovk, and V. Vapnik. Learning by transduction. In Gregory F. Cooper and Serafín Moral, editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 148–155, San Francisco, July 24–26 1998. Morgan Kaufmann.

[33] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 209–216. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[34] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects. *Internat. J. Robot. Autom.*, 4(2):193–203, 1988.

[35] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press LLC, Boca Raton, FL, 1997.

[36] J. Grasselli. *Linearna Algebra*. Društvo matematikov, fizikov in astronomov Slovenije, 1994.

[37] I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. Technical Report TR-99-01, Program in Applied and Computational Mathematics, Princeton University, January 1999.

[38] Martina Hasenjäger and Helge Ritter. Active learning with local models. *Neural Processing Letters*, 7(2):107–117, 1998.

[39] Paul S. Heckbert and Michael Garland. Multiresolution modeling for fast rendering. In *Proc. Graphics Interface '94*, pages 43–50, Banff, Canada, May 1994. Canadian Inf. Proc. Soc.

[40] N. Hitschfeld. Algorithms and data structures for handling a fully flexible refinement approach in mesh generation. In *Proceedings of the 4th International Meshing Roundtable*, pages 265–276, Sandia National Laboratories, October 1995.

[41] H. Hoppe. Progressive meshes. *Computer Graphics*, 30(Annual Conference Series):99–108, 1996.

[42] Hugues Hoppe. Efficient implementation of progressive meshes. Technical Report MSR-TR-98-02, Microsoft Research, January 1998. Available from http://www.research.microsoft.com/research/graphics/hoppe.

[43] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 19–26, August 1993.

[44] R. Jamnik. *Verjetnosti račun in statistika*. Društvo matematikov, fizikov in astronomov Slovenije, 1995.

[45] L. C. Kinsey. *Topology of Surfaces*. Springer-Verlag, New York, 1993.

[46] I. Kononenko. *Strojno učenje*. Založba FE in FRI, Ljubljana, 1997.

[47] E. Kreyszig. *Advanced Engineering Mathematics*. Wiley, 7th edition, 1993.

[48] Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 313–324. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[49] Terje Midtbo. Removing points from a Delaunay triangulation. In *Sixth International Symposium on Spatial Data Handling*, volume 2, pages 739–750, Edinburgh, Scotland, 1994.

[50] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[51] S. M. Omohundro. The Delaunay triangulation and function learning. Technical Report 90-001, International Computer Science Institute, Berekeley, CA, 1989.

[52] S. M Omohundro. Geometric learning algorithms. Technical Report ICSI-TR-89-041, International Computer Science Institute, Berkeley, 89.

[53] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1994.

[54] M.H. Overmars and J. Vleugels. Approximating Voronoi diagrams of convex sites in any dimension. *Int. J. of Comp. Geom. and Appl.*, 8:201–221, 1998.

[55] Jovan Popović and Hugues Hoppe. Progressive simplicial complexes. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 217–224. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[56] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 3rd edition, October 1990.

[57] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1992.

[58] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[59] D. A. Raikov. *Vector Spaces*. Noordhoff, Groningen, 1965.

[60] J. Rissanen. Minimum description length principle. In S. Kotz and N. L. Johnson, editors, *Encyclopedia of Statistical Sciences, 5*, pages 523–527, New York, 1985. Wiley.

[61] A. J. Robinson. *Dynamic Error Propagation Networks*. PhD thesis, Cambridge University, 1989.

[62] Malcolm Sambridge, Jean Braun, and Herbert McQueen. Geophysical parameterization and interpolation of irregular data using natural neighbours. *Geophysical Journal International*, 122, 1995.

[63] J. Shawe-Taylor and N. Cristianini. Data-dependent structural risk minimisation for perceptron decision trees. Technical report, NeuroCOLT2 Technical Report Series, May 1998.

[64] R. Sibson. A brief description of the natural neighbour interpolant. Technical report, Math Department, U. of Bath, 1980.

[65] R. Sibson. A brief description of natural neighbour interpolation. In Vic Barnet, editor, *Interpreting Multivariate Data*, pages 21–36. Wiley, Chichester, 1981.

[66] B. Stroustrup. *The C++ Programming Language*. Addison Wesley, 2nd edition, 1991.

[67] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Academic Press, London, 1998.

[68] Godfried Toussaint and Hossam El Gindy. Efficient algorithms for inserting and deleting edges from triangulations. In *Proc. International Conference on Foundations of Data Organization*. Kyoto University, May 1985.

[69] Godfried Toussaint and Jerzy W. Jaromczyk. Relative Neighborhood Graphs and their Relatives. *Proceedings of the IEEE*, 80(9):1502–1517, September 1992.

[70] D. F. Watson. *nngridr: An implementation of natural neighbor interpolation*. David Watson, 1994.

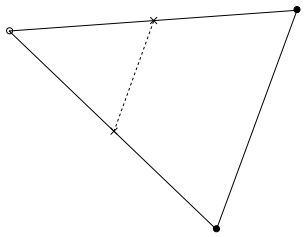
[71] Dietrich Wettschereck. *A Study of Distance-Based Machine Learning Algorithms*. PhD thesis, Oregon State University, June 1994.

[72] B. Zupan, M. Bohanec, J. Demšar, and I. Bratko. Learning by discovering concept hierarchies. *Artificial Intelligence*, 109:211–242, 1999.

A. Implementacija CCW

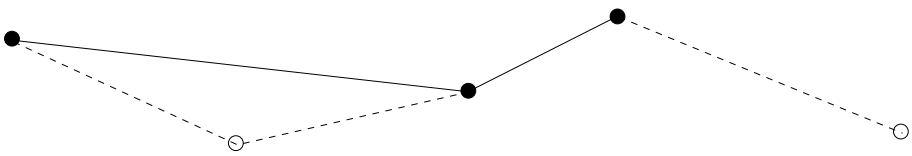
Program CCW uporabniku omogoča interaktivno delo s klasifikatorji za domene z dvema zveznima atributoma ter dvema vrednostima razreda. Te omejitve omogočajo, da se lahko domeno predstavi kot diagram točk v ravnini. Mejni simplicialni kompleks je pri CCW množica daljic (1-simpleksov), ki se med seboj stikajo kvečjemu v ogliščih.

Za razliko od SICC, CCW ustvarja klasifikatorje, katerih rezultati niso ekvivalentni metodi najbližjega soseda. Postopek izgradnje v CCW ustvari mejno daljico za vsak trikotnik (2-simpleks) Delaunayeve triangulacije (slika B.4), ki vsebuje položaje različnih razredov, na sredini tistih dveh stranic, ki povezujeta položaja različnih vrednosti razreda (slika A.1). Postopek klasifikacije in podatkovne strukture so identični tistim v SICC. Tudi v CCW moramo biti previdni pri situacijah, ko več učnih primerov sovpada, ali ko so manj kot trije učni primeri, saj takrat postopek ne bo nujno deloval pravilno.



Slika A.1.: Za nek Delaunayev trikotnik ustvarimo mejno daljico tako, da povežemo razpolovišča tistih dveh stranic v trikotniku, ki povezujeta položaja različnih vrednosti razreda.

CCW vključuje tudi enostaven postopek poenostavljanja klasifikacijskih simplicialnih kompleksov, ki temelji na ohranjanju konsistentnosti na učnih primerih. Deluje tako, da postopoma odstrani vsa tista oglišča (0-simplekse) v simplicialnem kompleksu, katerih odstranitev ohrani popolno konsistentnost na učnih primerih. Popolno konsistentnost enostavno preverimo tako, da poskusimo s poenostavljenim kompleksom klasificirati vse učne primere, njihova klasifikacija pa mora biti identična njihovemu razredu. Pri uporabljenem postopku izgradnje je neko oglišče vedno vsebovano v kvečjemu dveh daljicah. V primeru, da je oglišče vsebovano v eni daljici, jo skupaj z ogliščem odstranimo. V primeru, da je oglišče stičišče dveh daljic, ju obe odstranimo ter ustvarimo novo med preostalima ogliščema odstranjenih daljic (slika A.2).



Slika A.2.: Odstranjevanje daljic pri CCW: črtkano so prikazane odstranjene daljice, prazni krogci pa odstranjene točke.

Na postopek poenostavljanja lahko vplivamo z izbiro vrstnega reda točk. Najprej bi namreč hoteli porezati tiste, ki so bolj verjetno rezultat šuma ali prevelikega prilagajanja učnim primerom. Ena od smiselnih hevristik bi recimo najprej odstranjevala točke, ki ležijo med zelo oddaljenimi učnimi primeri.

Če bi princip Occamove britve vedno veljal, potem bi morali ob takem poenostavljanju vedno doseči boljše rezultate. Namreč, hipotezo poenostavimo tako, da jo vedno ohranjamo identično po klasifikacijski točnosti na učnih primerih. Ker to na podlagi eksperimentalnih testov ni res, lahko vidimo tudi to, da Occamova britev sama po sebi ne zagotavlja uspeha. Poleg kompleksnosti v smislu števila simpleksov moramo upoštevati še marsikaj, na primer gladkost mejne ploskve in ohranjanje razmerja med volumnom ob poenostavljanju, kar smo poudarjali v prejšnjih poglavjih. V skladu z metodo najbližjih sosedov si želimo, da naj bo mejna ploskev enako oddaljena od njej najbližjih primerov. Seveda se moramo zavedati, da si te zahteve včasih nasprotujejo.

CCW dovoljuje še nekaj interaktivnih operacij, kot na primer spreminjanje metrike, dodajanje šuma, dodajanje učnih primerov, povečevanje korelacije ter še nekaj drugih funkcij. Lahko si ogledamo Delaunayevo triangulacijo, lahko preizkusimo klasifikacijo z 1-NN in metodo lokalno obtežene regresije. Interaktivno lahko preizkušamo klasifikacijo za poljuben primer, pri čemer tudi opazujemo, kateri simpleks mu je najbližji. Zaradi vseh teh funkcij je lahko CCW tudi orodje za vizualizacijo in študijo domene ter klasifikatorjev, čeprav je sam po sebi precej omejen.

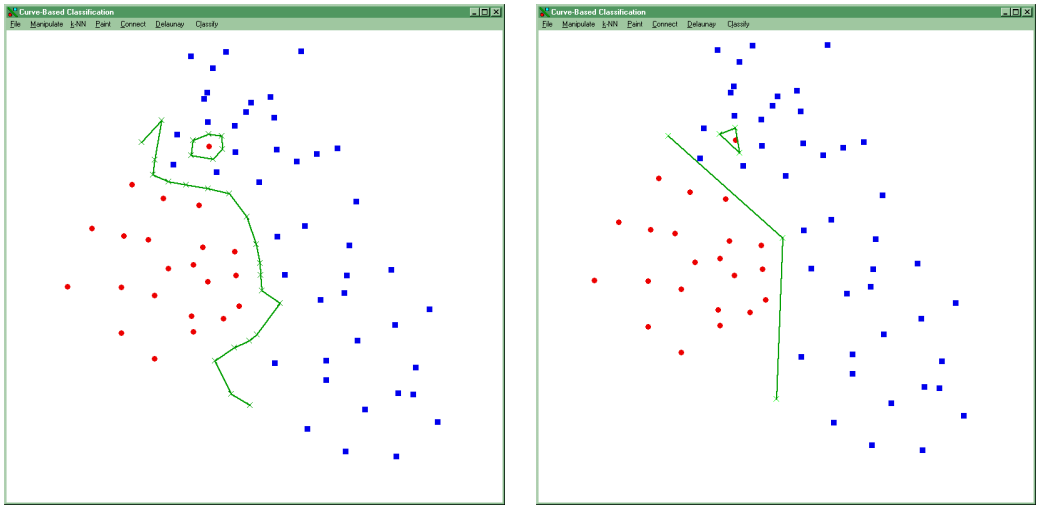
Zaradi svojih omejitev je CCW le primeren le za omejeno število domen, obnese pa se približno tako dobro kot metoda najbližjega soseda, saj sta si pristopa zelo podobna. CCW smo preizkusili le okvirno na domeni prepoznavanja samoglasnikov [61] iz *CMU Neural Network Benchmark Archive*. Ta vsebuje 10 različnih zveznih atributov. V tabeli A.1 in na sliki B.3 so prikazani rezultati CCW pri križnem preverjanju v primerjavi s poznanim postopkom C4.5 [58], vendar pa sta bila zaradi omejitev CCW uporabljena le dva atributa, čeprav bi jih lahko C4.5 upošteval več.

Ta preizkus je bil le ilustrativen. Namen je prikazati delovanje postopka ter izgled klasifikacijskih simplicialnih kompleksov. Za resno preizkušanje uspešnosti je potrebno obravnavati skoraj poljubno število atributov, kar počne SICC. Da pa bi presegli učinkovitost metode najbližjega soseda, je treba v SICC še vključiti poenostavljanje ter obravnavo šuma.

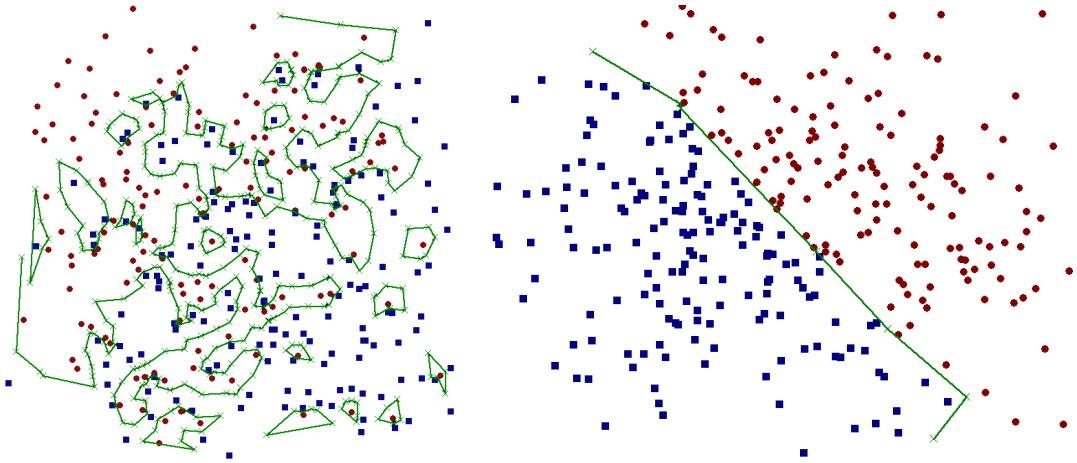
	CCW	**CCW-p**	**C4.5**
	8.1	10.1	12.1
	12.1	16.2	18.2
	12.1	11.1	9.1
	16.2	12.1	18.2
	16.2	15.2	10.1
	14.1	16.2	16.2
	6.1	7.1	12.1
	11.1	10.1	16.2
	12.1	12.1	18.2
	14.1	14.1	10.1
Povprečno	**12.2**	**12.4**	**14.0**
Najboljši	**4x+1**	**2x+1**	**3x**

Tabela A.1.: Rezultati metode CCW s poenostavljanjem (CCW-p) in brez v primerjavi z metodo C4.5. V tabeli so prikazani deleži napačno klasificiranih testnih primerov, dobljeni s križnim preverjanjem. Uporabljena je bila domena *"Vowel Recognition"* iz *CMU Neural Network Benchmark Archive*. Izbrana atributa sta 1. in 2., v prvi razred spadajo $\{O, o, U, u\}$, v drugega pa vsi ostali.

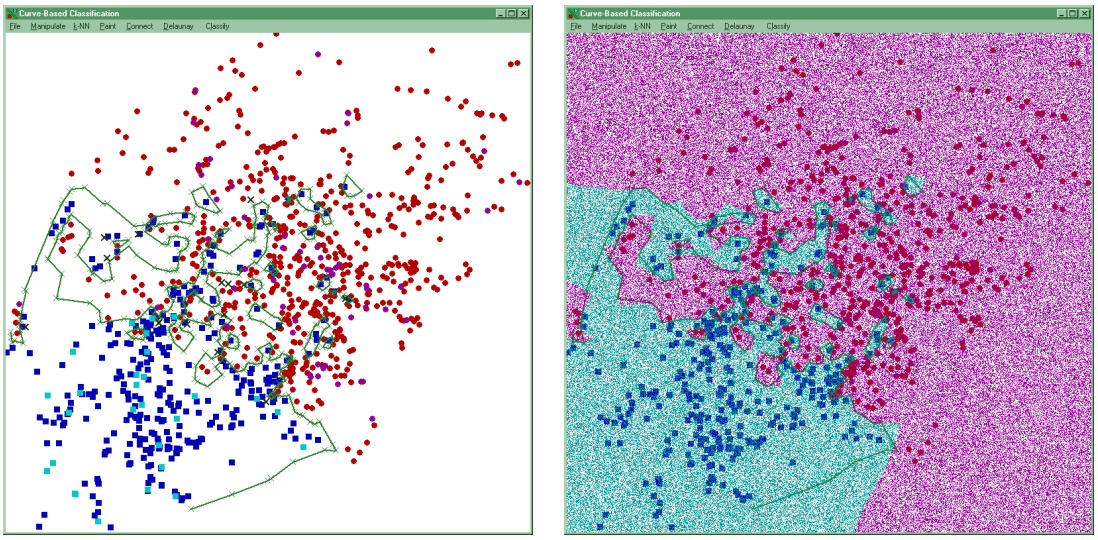
B. Barvna priloga



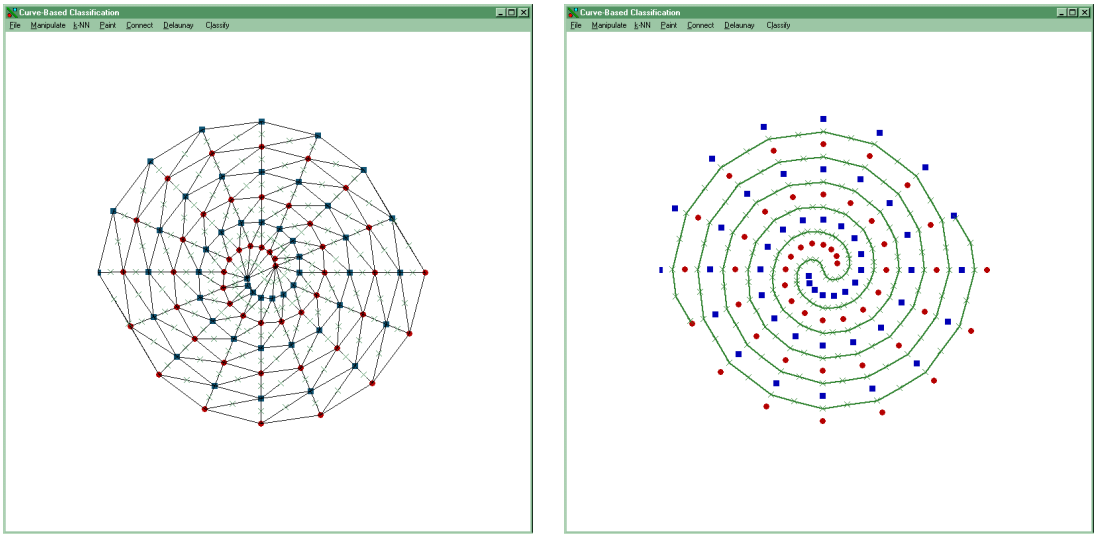
Slika B.1.: Prikaz poenostavljanja mejne krivulje z združevanjem oglišč ob ohranjanju skladnosti z vsemi učnimi primeri v aplikaciji CCW.



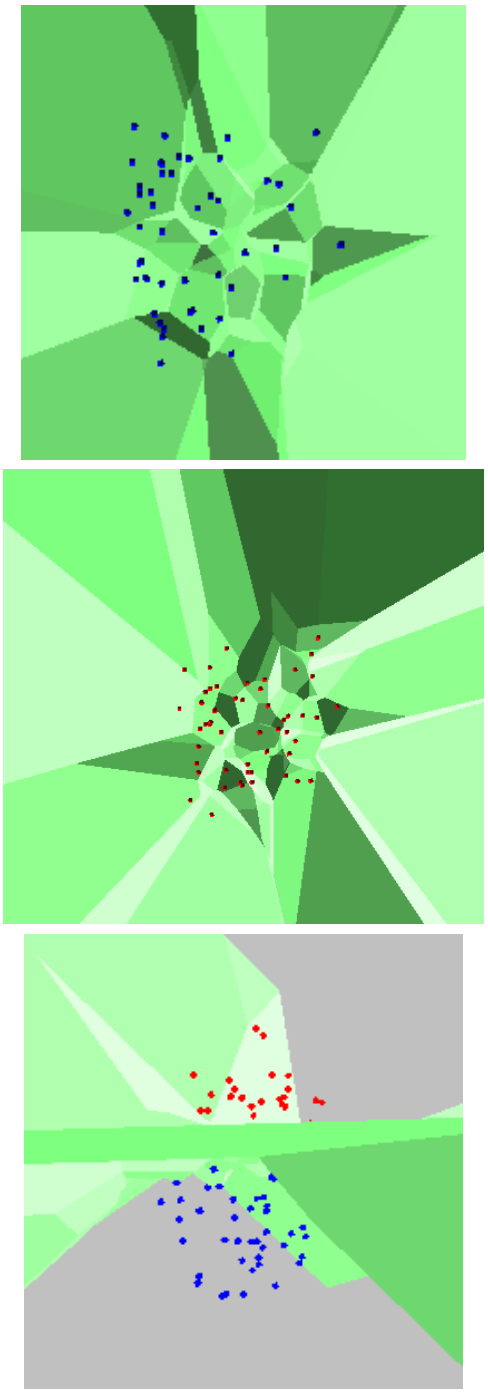
Slika B.2.: Domena $z < x + y$ je lahko zelo kompliciranа, če izpustimo atribut z (na prvi sliki), ali pa zelo enostavna, če prikazujemo transformirana atributa $2z$ in $x + y$ (na drugi sliki).



Slika B.3.: Domena "Vowel Recognition" iz *CMU Neural Network Benchmark Archive* pri interaktivnem delu s CCW. Črni križci predstavljajo napačno, vijolične in sinje točke pa pravilno klasificirane primere. Na prvi sliki so klasifikacije s CCW, na drugi pa je prikazan klasifikator po metodi najbližjega soseda za veliko naključno izbranih točk pri isti domeni.



Slika B.4.: Obnašanje programa CCW na domeni spirale, ki je za večino algoritmov strojnega učenja trd oreh. Na prvi sliki je prikazana Delaunayeva triangulacija domene, na drugi pa ustrezni simplicialni kompleks.



Slika B.5.: Pogledi iz različnih položajev na zeleno mejno ploskev, predstavljeno z 2-simpleksi v 3D prostoru, v domeni s tremi zveznimi atributi. Mejno ploskev je izdelala aplikacija SICC in je prikazana s pregledovalnikom datotek VRML. Osvetlitev posameznega simpleksa je odvisna od lege v prostoru, učni primeri pa so rdeči in modri.