České vysoké učení technické
Fakulta elekrotechnická
Katedra kybernetiky

Diplomová práce

PROGRAMOVÉ NÁSTROJE PRO ROZPOZNÁVÁNÍ

Praha, 2000 Vojtěch Franc

České vysoké učení technické
Fakulta elekrotechnická
Katedra kybernetiky

Diplomová práce

Programové nástroje pro rozpoznávání

Vojtěch Franc

Praha, 28. ledna 2000

Vedoucí práce: Prof. Ing. Václav Hlaváč, CSc.

Prohlašuji, že předloženou práci jsem vypracoval sám pouze za použití uvedené literatury. Souhlasím s volným užitím výsledků této práce fakultou elektrotechniky na ČVUT v Praze.

V Praze dne 28. ledna 2000 Vojtěch Franc

Anotace

Tématem této diplomové práce je návrh a realizace programových nástrojů pro rozpoznávání v programu Matlabu. Implementovány jsou vybrané metody z knihy M.I. Schlesinger a V. Hlaváč, Deset přednášek z teorie statistického a strukturního rozpoznávání. Tato kniha velmi obecně popisuje klasické i méně známé metody rozpoznávání a snaží se je sjednotit do ucelených rámců. Programový balík obsahuje metody z první části knihy, která se zabývá příznakovými metodami statistického rozpoznávání. Mezi vybrané metody patří lineární diskriminační funkce, kvadratická diskriminační funkce, algoritmy učení bez učitele a algoritmy mimimaxního učení. Usnadnit pochopení důležitých algoritmů z uvedené knihy si klade za cíl sada nástrojů, jejichž návrhem a implementací se zabývá tato práce. Mezi nejdůležitější vlastnosti programového balíku patří možnost vizualizace a interaktivní ovládání procesu řešení, porovnání různých algoritmů na jednom typu úlohy, dále snadná možnost generování syntetických dat a provádění samostatných experimentů s implementovanými algoritmy rozpoznávání. Mezi metodami jsme se nejvíce zaměřili na zobecněnou Andersonovu úlohu, přičemž se nám podařilo navrhnout a implementovat malé vylepšení algoritmu jejího řešení.

Annotation

The aim of the diploma thesis is to develop and to implement a statistical pattern recognition toolbox for Matlab. The toolbox contains selected methods published in the recently appeared monograph M. I. Schlesinger and V. Hlaváč, Ten lectures from the statistical and structural pattern recognition theory. Several of the reported methods are not widely known, provide solution to a more general tasks than before, and give a new systemathic insight to the statistical pattern recognition tasks. The toolbox covers first part of the monograph which deals with feature-based algorithms for synthesis of linear or quadratic discriminant function, minimax learning, and unsupervised learning. The toolbox should help to understand relevant algorithms better and to demonstrate their functionality. The visualisation of the process leading to the solution and experimentation feasibility is stresed for this reason. The demonstrator environment is provided that allows the user to choose different algorithms, compare their behavior, provides tools to control the algorithm run interactively, creates synthetic input data or uses real ones. A substantial attention was devoted to the generalized Anderson's task. In addition, there is a small contribution in the diploma thesis, namely in the algorithm solving the generalized Anderson's task.

Poděkování

Rád bych poděkoval panu profesorovi Václavu Hlaváčovi za jeho vedení a cenné rady a panu profesorovi M.I. Schlesingerovi, který mi ochotně odpovídal na otázky z rozpoznávání. Také bych rád poděkoval členům Centra strojového vnímání za pomoc a rady při psaní v typografickém systému LaTeX.

Obsah

1. Úvod

Tématem této diplomové práce je vývoj sady nástrojů (toolbox) pro Matlab, které mají demonstrovat vybrané algoritmy z knihy [8] (dále jen Kniha) zabývající se statistickými metodami rozpoznávání. Objektem našeho zájmu jsou metody z první části Knihy, v níž jsou popsány příznakové metody statistického rozpoznávání. Mezi témata vybraná k demostraci patří syntéza lineární diskriminační funkce, kvadratické diskriminační funkce, metody učení bez učitele a minimaxní učení.

Kniha se snaží sjednotit známé metody rozpoznávání do ucelených rámců. Většina popisovaných algoritmů je navržena pro řešení velmi obecných úloh, které jsou v Knize podrobně matematicky analyzovány. Usnadnit pochopení důležitých algoritmů z Knihy formou ukázky řešení konkrétních úloh si klade za cíl sada nástrojů, jejichž návrhem a implementací se zabývá tato práce.

Mezi nejdůležitější vlastnosti této sady nástrojů patří možnost vizualizace a interaktivní ovládání procesu řešení, porovnání různých algoritmů na jednom typu úlohy, dále snadná možnost generování syntetický dat a provádění samostatných experimentů s implementovanými algoritmy rozpoznávání.

Pro implentaci byl zvolen program Matlab, který je ideálním nástrojem umožňujícím vývoj a vizualizaci řešení technických úloh. Jeho standardní součástí jsou propracované vizualizualizační funkce, účinné algoritmy pro počítátní s maticemi a v neposlední řadě nástroje pro optimalizaci. Další velké plus je jeho podpora na mnoha počítačových platformách, takže v něm napsané programy jsou nezávislé na konkrétním operačním systému.

Přesný výčet metod, které se mají demonstrovat v našem programovém balíku, je uveden v následující specifikaci úloh:

1. Lineární diskriminační funkce.
   a) Andersonova úloha pro dvě třídy.
   b) Srovnání původního řešení s metodou gradientního růstu a s ε-řešením.
   c) Perceptron a Kozincův algoritmus, srovnání.
   d) Nalezení Fisherova klasifikátoru pomocí perceptronu a Kozincova algoritmu.
2. Učení bez učitele.
   a) Normální rozdělení, nezávislé příznaky.
   b) Normální rozdělení, závislé příznaky.
3. Algoritmy minimaxního učení.
4. Syntéza kvadratické diskriminační funkce.

Během studia metod rozpoznávání jsme kromě Knihy použili publikaci [6] a především knihu [4], která nám byla hodnotnou učebnicí při získávání základů z tohoto oboru.

2. Existující nástroje pro rozpoznávání

V této kapitole popíšeme balíky nástrojů pro Matlab, které se zabývají rozpoznáváním. Jako zdroj informací jsme použili Internet a informace poskytované firmou MathWork (výrobce Matlabu). V popisu těchto produktů se věnujeme především výčtu funkcí, vztahujících se k našemu problému, tedy metodám statistického rozpoznávání a jejich vizualizaci.

2.1 Ahlea - Pattern Recognition Toolbox

Autorem je Ahlea Systems Corp., http://www.ahlea.com. Jedná se o plně komerční produkt. Infromace jsme čerpali z prezentace na Internetu. Zkušební verze není k dispozici.

Tento balík nástrojů obsahuje funkce pro vizualizaci, pro reprezentaci dat a práci s nimi, metody shlukování a funkce pro odhad chyby klasifikace.

Vizualizace: Funkce pro vykreslování elips znázorňující normálně rozloženou hustotu pravděpodobnosti. Funkce vykreslující rozdělující nadplochu pro bayesovskou rozhodovací strategii. Dále je zde umožněno zobrazit datové množiny ve 2D a 3D.

Shlukování: Pro shlukování (učení bez učitele) je zde obsažena klasická metoda k-průměru a její modifikace pro případ dat s normálně rozdělenou hustotou pravděpodobnosti.

Popisovaný balík nástrojů obsahuje poměrně malý počet metod rozpoznávání, které nejsou zaměřeny na žádnou specifickou oblast rozpoznávání.

2.2 B. Duin - PRTools, A Matlab Toolbox for Pattern Recognition

Autorem je Bob Duin, Delft University of Technology, The Netherlands, http://www.ph.tn.tudelft.nl/prtools. Tento balík vznikl pro akademické účely a je volně šiřitelný na Internetu. Informace jsme čerpali jak z dokumentace, tak z provedených experimentů.

Balík obsahuje více jak 100 funkcí pokrývající velkou část statistických metod rozpoznávání. Jsou zde obsaženy funkce pro generování syntetických dat, včetně několika reálných datových množin. Dále jsou zde implementovány metody pro předzpracování dat a jejich analýzu, jako je Fisherův lineární diskriminant nebo metody zakřivení příznakového prostoru. Kromě klasických metod rozpoznávání, jsou v něm obsaženy metody používající neuronových síti, funkce pro stavbu rozhodovacích stromů, metody pro analýzu chyb klasifikace atd.

Vizualizace: Jsou zde funkce pro vykreslování diskriminačních funkcí ve 2D, pro vykreslování datových množin, pro kreslení dendogramů (výsledek shlukové analýzy), atd.

Klasifikátory: Většina klasických metod, jako např. bayesovská rozhodovací strategie, klasifikace podle k-nejbližších sousedů, Fisherovy klasifikátory, Percepron, neuronové sítě a mnoho dalších.

Záběr popisovaného balíku je opravdu široký. Po porovnání s ostatními produkty, které jsme objevili, se zdá být nejkomplexnější co se týče implementovaným metod statistického rozpoznávání.

2.3 Innovative Dynamic, Inc. - Pattern Recognition Toolbox

Autorem je firma Innovative Dynamics, Inc., http:/www.innodyn.com. Jedná se o plně komerční produkt. Infromace jsme čerpali z prezentace na Internetu. Zkušební verze není k dispozici.

Tento balík je úzce zaměřen na předzpracování a analýzu průběhu jednorozměrné veličiny (signálu). Jsou zde obsaženy metody vytvářející příznaky pro rozpoznávání z časového průběhu signálu a frekvenčních charakteristik.

Vizualizace: Poskytované funkce jsou specializováné na zobrazení jednorozměrných signálů a jeho charakteristik.

Klasifikátory: Klasifikace podle k-sousedů, metoda shlukování k-průměrů, prahové klasifikátory a různé typy neuronových sítí.

2.4 Závěr o existujících nástrojích

Po zpracování všech dostupných informací jsme dospěli k následujícím závěrům. Ucelených programových balíků pro Matlab, které se zabývají statistickým rozpoznávání nebo jen obecně rozpoznáváním, je překvapivě málo a nebo jsou těžko dostupné. O něco více dostupné jsou separované programy (m-file), zabývající se nějakou konkrétní úlohou rozpoznávání. Všechny popsané produtky vyžadují, aby byl uživatel poměrně dobře seznámen s problematikou rozpoznávání. Mezi balíky nástrojů, nejvíce odpovídající požadavkům našeho zadání, patří B. Duin - PRTools popsaný v odstavci (2.2).

3. Návrh nástrojů pro rozpoznávání

Metody rozpoznávání jsou v našem programovém balíku členěny podle tématického rozdělení, které je uvedeno ve specifikaci vybraných úloh popsaných v úvodu. Pro každou skupinu algoritmů, řešících stejný typ úlohy, je vytvořen demostrační program. V něm lze tyto algoritmy porovnávat na řešení konkrétní úlohy. Průběh procesu řešení je zobrazován a je možné jej interaktivně ovládat. Každá úloha je popsána jedním datovým souborem jehož formát je velmi jednoduchý. Uživatel může algoritmy vyzkoušet na několika vzorových příkladech, které jsou k danému typu úloh vytvořeny, a nebo si vytvořit své vlastní zadání úlohy. Pro tento účel jsme vytvořili speciální program. Další možností je konverze dat z jiných programů, která je také podporována. Jednotlivé algoritmy rozpoznávání, vizualizační funkce a funkce pro práci s daty jsou samostatnými programy, takže je uživatel může používat ve vlastních experimentech.

Rozdělení programů na skupiny podle jejich účelu je následující:

1. Algoritmy statistického rozpoznávání.
2. Demonstrační programy.
3. Algoritmy vizualizace.
4. Program pro vytváření syntetických dat a funkce pro práci s nimi.

V dalších odstavcích se budeme věnovat jednotlivým skupinám programů, přičemž se soustředíme pouze na popis myšlenky. Pro konkrétní implementaci a syntaxi funkcí odkazujeme na přiložené zdrojové kódy a uživatelskou dokumentaci.

3.1 Algoritmy statistického rozpoznávání

Samotné algoritmy jsou v Knize vysvětleny a popsány velmi obecně, takže jsme při jejich impelentaci pro konkrétní typ úloh měli poměrně velkou volnost. Navíc je bylo třeba navrhnou tak, aby se daly spouštět a zastavovat v libolném stavu řešení, neboť tyto funkce jsou potřebné pro průběh vizualizace a pro interaktivní ovládání procesu řešení.

Při implementaci jsme se snažili o obecnost algoritmů a ne o optimalizaci na nějakou konktrétní úlohu. Většina algoritmů může zpracovávát data z příznakového prostoru libovolné konečné dimenze. Omezení je dáno pouze pamětí počítače a velikostí matice, kterou je Matlab schopen zpracovat. V případech, kdy to bylo možné, jsme převáděli výpočetních postupy do maticového zápisu, pro který je Matlab optimalizovaný.

Výsledné algoritmy, obsažené v našem programovém balíku, jsou popsány v další samostatných kapitolách. Uvádíme v nich jen nejdůležitější definice a postupy, nezbytně nutné pro vymezení daného problému a jeho řešení. Všechny metody, zmiňované v těchto kapitolách, jsou v našem programovém balíku obsaženy.

Navíc jsme implementovali několik algoritmů, které nejsou v Knize popsány. Především to je metoda Linear Support Vector Machine, která hledá optimální rozdělení konečných množin bodů. K zařazení této metody do našeho programového balíku nás vedly dva důvody. Zaprvé to, že se nám ji podařilo velice efektivně implementovat díky optimalizačním nástrojům Matlabu, a proto ji používáme v jiných úlohách, kde je nutné řešit optimální rozdělení množiny bodů. Zadruhé umožňuje uživateli jiný pohled na problém, který je v Knize řešen interačními algoritmy.

Dále jsme navrhli vylepšení jedné z podúloh algoritmu, který hledá optimální řešení zobecněné Andersonovy úlohy. Také jsme experimentovali z různými kombinacemi podúloh, jenž ovlivňují celkové vlastnosti uvedeného algoritmu. Podrodný popis je v další kapitole.

Nakonec odstavce uvádíme seznam algoritmů rozpoznávání, které jsou v našem programovém balíku obsaženy.

Implementované algoritmy rozpoznávání:

- Lineární diskriminační funkce.
  - Rozdělení konečných množin bodů.
    - ▷ Perceptron.
    - ▷ Kozincův algoritmus.
    - ▷ ε-řešení modifikovaným Kozincovým algoritmem.
    - ▷ Linear Support Vector Machine.
  - Fisherovy klasifikátory.
    - ▷ Řešení modifikovaným perceptronem.
    - ▷ Řešení modifikovaným Kozincovým algoritmem.
  - Zobecněná Andersonova úloha.
    - ▷ Původní Andersonovo řešení.
    - ▷ ε-řešení.
    - ▷ Naplnění kostry algoritmu pro obecné řešení Andersonovy úlohy metodami popsanými v Knize a jednou nově navrženou.
    - ▷ Řešení metodou zobecněné gradientní optimalizace.
- Kvadratická diskriminační funkce. Transformace vyrovnání příznakového prostoru pro kvadratickou funkci.
- Algoritmy učení.
  - Algoritmus učení bez učitele pro model s normálním rozdělením a korelovanými i nekorelovanými příznaky.
  - Algoritmus minimaxního učení pro model s normální rozdělením a korelovanými příznaky.

První úroveň uvedeného výčtu algoritmů je shodná s názvy kapitol, ve kterých jsme popsali řešení jednotlivých úloh.

3.2 Demonstrační programy

Demonstrační programy jsou samostatnou součástí našeho programového balíku. Pro každou skupinu algoritmů, řešících stejný typ úlohy, je obsažen jeden demonstrační program. Jsou napsány s využitím nástrojů Matlab User Interface, které umožňují použití stejného uživatelského prostředí a ovládání jako běžné aplikace na daném operačním systému.

Koncepci demonstračních programů (vzhled a funkce) jsme se snažili zachovat u všech úloh co možná nejvíce podobnou. Všechny textové popisy jsou v angličtině. Zde popíšeme ovládací prvky a funkce společné pro všechny demonstrační programy:

- V programu je možné nahrávat existující data nebo interaktivně vytvářet nové zadání úloh.
- Jedna úloha může být řešena více algoritmy, které lze takto porovnávat mezi sebou. Navíc, pokud to algoritmy dovolují, lze část úlohy vyřešit jednou metodou a část úlohy jinou metodou.
- Proces řešení úlohy lze interaktivně řídit povely:
  - Inicializuj proces řešení.
  - Spusť proces řešení.
  - Zastav proces řešení.
  - Proveď zadaný počet kroků.
- Současný stav řešení úlohy je zobrazován v reálném čase a to jak graficky, tak textovým popisem.
- Graficky zobrazený výsledek řešení lze přímo exportovat do formátu Encapsulated PostScript (*.eps), který podporuje většina programů (například LaTeX).
- Je možné vyvolat okno s kontextovou nápovědou vztahující se ke konkrétní úloze.

Všechny úlohy demonstrujeme ve dvourozměrném prostoru, tedy v rovině. Odlišnost jednotlivých demonstračních programů je dána způsobem zobrazování procesu řešení a možností zadávání údajů spojených s konkrétním typem úlohy. V následujících odstavcích popíšeme odlišnosti jednolivých demonstračních programů.

3.2.1 Rozdělení konečných množin bodů

V tomto demostračním programu je možné experimentovat s následující algoritmy:

- Perceptron.
- Kozincův algoritmus.
- ε-řešení modifikovaným Kozincovým algoritmem.
- Linear Support Vector Machine.

Cílem popisovaného typu úloh je rozdělit konečné množiny bodů pomocí nadroviny. Podrobnější popis a formulace úlohy je v odstavci 4.3.

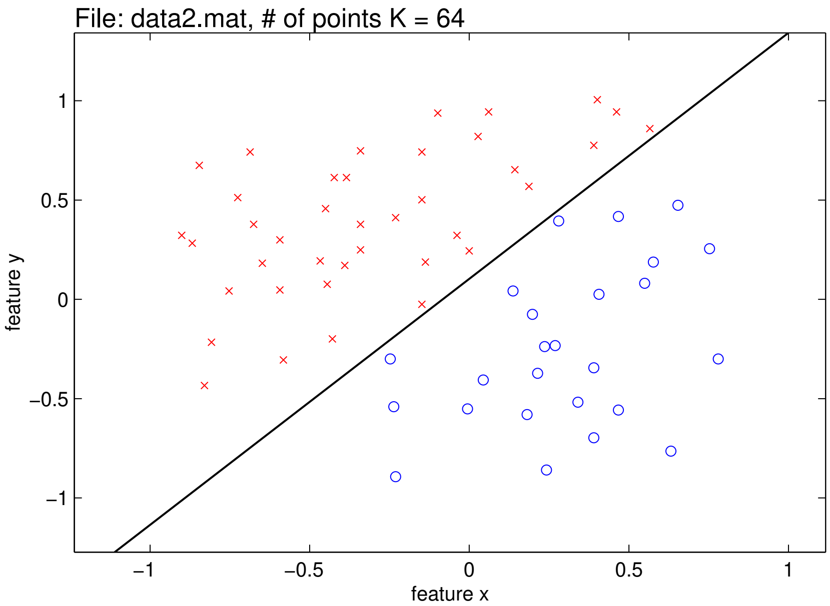
Objekty ze vstupní množiny jsou znázorněny jako body v rovině, přičemž příslušnost k dané třídě je určena barvou a také tvarem bodu pro případ černobílehé zobrazení nebo výstupu na tiskárnu.

Stav řešení úlohy je znázorněn polohou přímky, která během procesu řešení mění svoje umístění. Je možné zvolit plynulé nebo skokové změny polohy přímky. Výpis hodnoty současného řešení, parametrů přímky, je zobrazován v komentářovém okně. Zde je také informace o stavu řešení, tedy kolik iterací algoritmus provedl a zda-li je už úloha vyřešena nebo ne. Ilustrační úloha je na obrázku 3.1, pro jehož vygenerování jsme použily funkci exportu do EPS formátu.

U algoritmu *ε-řešení* je zapotřebí vkládat přesnost řešení, pro tento účel je zde speciální vstupní řádka.

3.2.2 Fisherovy klasifikátory

V tomto demostračním programu je možné experimentovat s následující algoritmy:



Obrázek 3.1 Rozdělení konečných množin bodů.

- Řešení Fisherova klasifikátoru modifikovaným perceptronem.
- Řešení Fisherova klasifikátoru modifikovaným Kozincovým algoritmem.

Cílem těchto algoritmů je rozdělit konečný počet konečných množin bodů pomocí linární diskriminační funkce. Každé třídě (množině bodů) odpovídá jeden vektor, který se musí určit tak, aby sklární součin každého bodu dané třídy s odpovídajícím třídním vektorem byl větší než skalární součin s jakýmkoli vektorem patřící jiné třídě. Podrobný popis je uveden v odstavci 4.4.

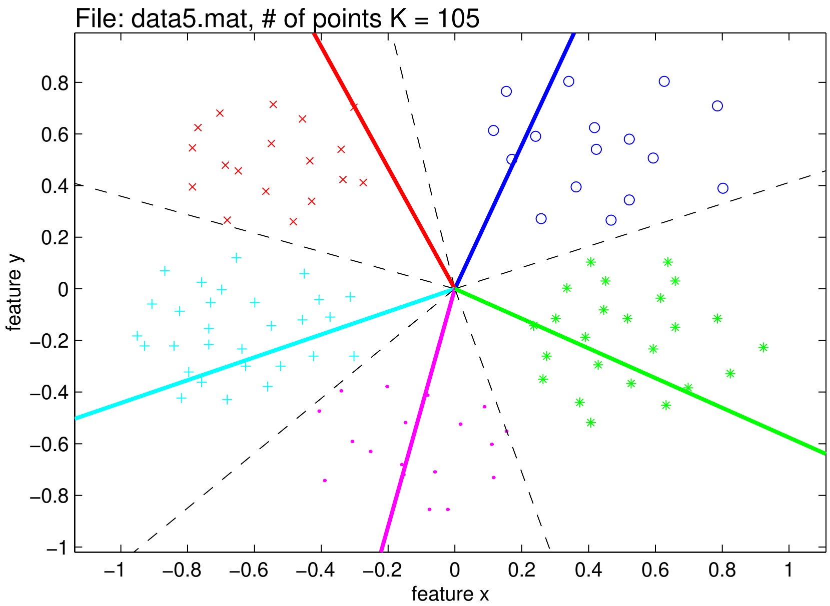
Vstupní množiny jsou znázorněny jako body v rovině, přičemž příslušnost k dané třídě je opět určena barvou a tvarem bodu. Naladění klasifikátoru určují třídní vektory, kdy každá třída má právě jeden. Vektory jsou zobrazeny jako přímky vycházející z počátku souřadnic a jejich barva odpovídá barvě bodů z příslušné třídy. Navíc jsme postupným porovnáváním vektorů pro dvě třídy určili rovnice rozdělující nadplochy mezi třídami, které jsou zobrazeny čárkovanou čarou. Textový komentář k úloze obsahuje počet provedených iterací a informaci o tom, je-li již řešení nalezeno nebo ne. Ilustrační úloha je na obrázku 3.2.

3.2.3 Kvadratická diskriminační funkce

V tomto demostračním programu se experimentuje se syntézou kvadratické diskriminační funkce, kterou hledáme ve vyrovnaném příznakovém prostoru. Díky vyrovnání příznakového prostoru můžeme pro syntézu kvadratické funkce použít všechny algoritmy, které hledají lineární rozhodovací strategii, což je také v tomto demonstračním programu provedeno.

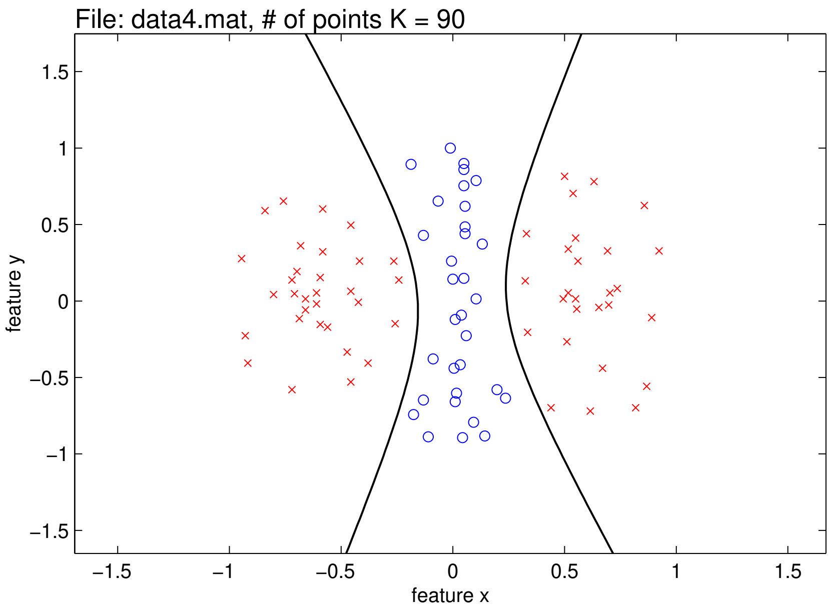
Uvedená úloha rozšiřuje lineární strategii rozhodování na kvadratickou, která je více obecná a samozřejmě může rozdělit i lineárně neseparovatelné množiny bodů. Navíc je kvadratická funkce pro rozpoznávání dosti významná. Například optimální bayesovská strategie pro normální rozdělení vede na kvadratickou rozdělující funkci.

Provedení demonstračního programu je stejné jako pro lineární diskriminační funkci



Obrázek 3.2 Nalezení Fisherova klasifikátoru pro 5 tříd. Třídní vektory jsou označeny plnou čarou a rozdělující přímky jsou čárkovaně.

až na to, že je rozdělující nadplocha kvadratická funkce. Nalezená nadrovina ve vyrovnaném 5D prostoru se transformuje na obecnou kvadratickou funkci ve 2D, která se zobrazuje v různých tvarech (elipsa, hyperboly, přímky, atd.). Podrobná analýza vizualizace této funkce je uvedena v odstavci 3.3.1. Ilustace nalezené kvadratické funkce v demonstračním programu ukazuje obrázek 3.3.



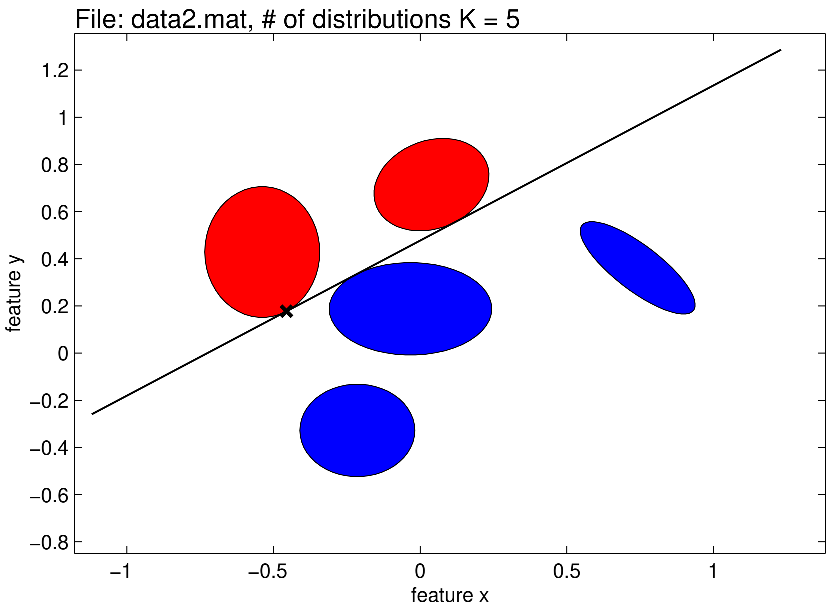
Obrázek 3.3 Rozdělení dvou konečných množin bodů pomocí kvadratické diskriminační funkce.

3.2.4 Zobecněná Andersonova úloha

V tomto demostračním programu je možné experimentovat s následující algoritmy:

- Původní Andersonovo řešení.
- ε-řešení.
- Naplnění kostry algoritmu pro obecné řešení Andersonovy úlohy metodami popsanými v Knize a jednou nově navrženou.
- Řešení metodou zobecněné gradientní optimalizace.

V Andersonově úloze se hledá rozdělující nadrovina, která od sebe co nejlépe oddělí dvě třídy, určené směsí normálních rozdělení. Tyto třídy zobrazujeme jako množinu elips, jejichž střed určuje střední hodnota a tvar kovarianční matice. Všechny elipsy mají stejný poloměr daný rozměrem nejmenší elipsy, která určuje kvalitu nalezeného řešení. Úloha má řešení ve stavu, kdy je rozdělující přímka přitisknuta elipsami tak, že se její pozice nemůže změnit aniž by se zmenšila velikost nejmenší z těchto elips. Podrobný popis je uveden v odstavci 4.5. Experimentováním s parametry matlabovských vizualizačních funkcí, se nám podařilo implemetovat poměrně rychlé zobrazování rozrůstajících se elips, které znázorňují přibližování stavu algoritmu k výslednému (optimálnímu) řešení. Příklad řešení konkrétní zobecněné Andersonovy úlohy je vidět na obrázku 3.4.



Obrázek 3.4 Řešení zobecněné Andersonovy úlohy ve 2D. První třídu popisují dvě a druhou tři normální rozdělení.

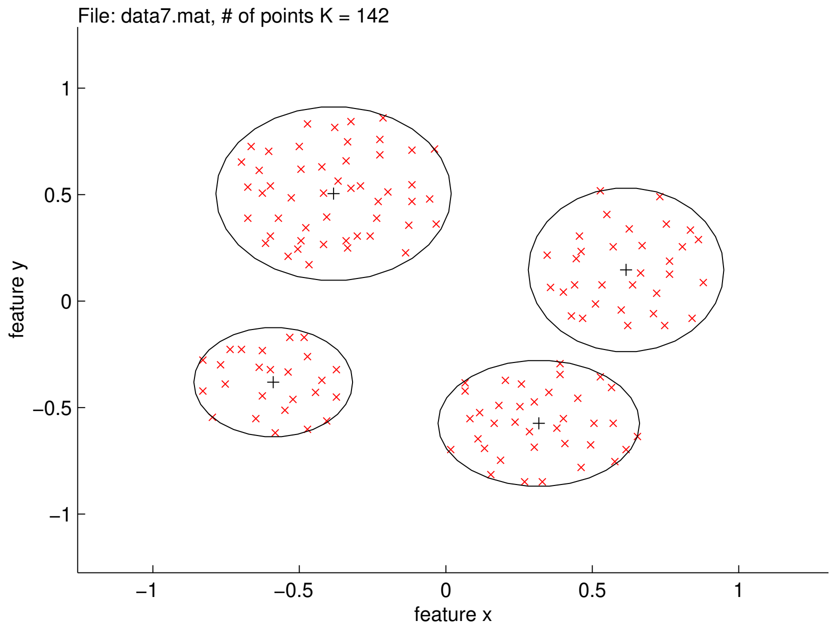
V komentářovém okně je zobrazen počet kroků algoritmu, parametry nalezeného řešení a také poloměr nejmenší elipsy (normálního rozdělení), který je nepřímo úměrný kvalitě řešení. Demonstrační program také obsahuje vstupní řádky, pro zadávání parametrů ukončovacího kritéria, které mají jednotlivé algoritmy odlišné.

3.2.5 Učení bez učitele

Tento demonstrační program umožňuje experimenty s algoritmem učení bez učitele pro model s normálním rozdělením a případ korelovaných a nekorelovaných příznaků.

Uvedený algoritmus se snaží doplnit informace o statistickém modelu předem zadaného počtu tříd z neohodnocené množiny dat (učení bez učitele). Kromě počtu tříd je další známou informací tvar modelu (hustota pravděpodobnosti s normálním rozdělením) a znalost o závislosti nebo nezávislosti příznaků. Algoritmus hledá neznámé parametry normálních rozdělení, jenž popisují modely jednotlivých tříd, a to provádí tak dlouho, dokud se nedostane do stacionárního bodu řešení, tzn. že se hledané modely tříd nezmění ve dvou po sobě následujících krocích. Podrobný popis algoritmu je v odstavci 6.1.

Stav řešení zobrazujeme množinou elips, kdy každé třídě odpovídá právě jedna. Střed a tvar elipsy je určen parametry statistického modelu pro danou třídu, tj. střední hodnotou a kovarianční maticí. Velikost elipsy je určena tak, aby obsahovala všechny body, které jsou na základě stavajícího statického modelu do této třídy klasifikovány, pomocí optimání bayesovské strategie. Zjednodušeně řečeno, elipsy se snaží rozdělit trénovací množinu mezi sebe, přičemž pravidla rozdělování určuje algoritmus učení bez učitele. Na obrázku 3.5 je zobrazeno nalezené řešení pro čtyři třídy.



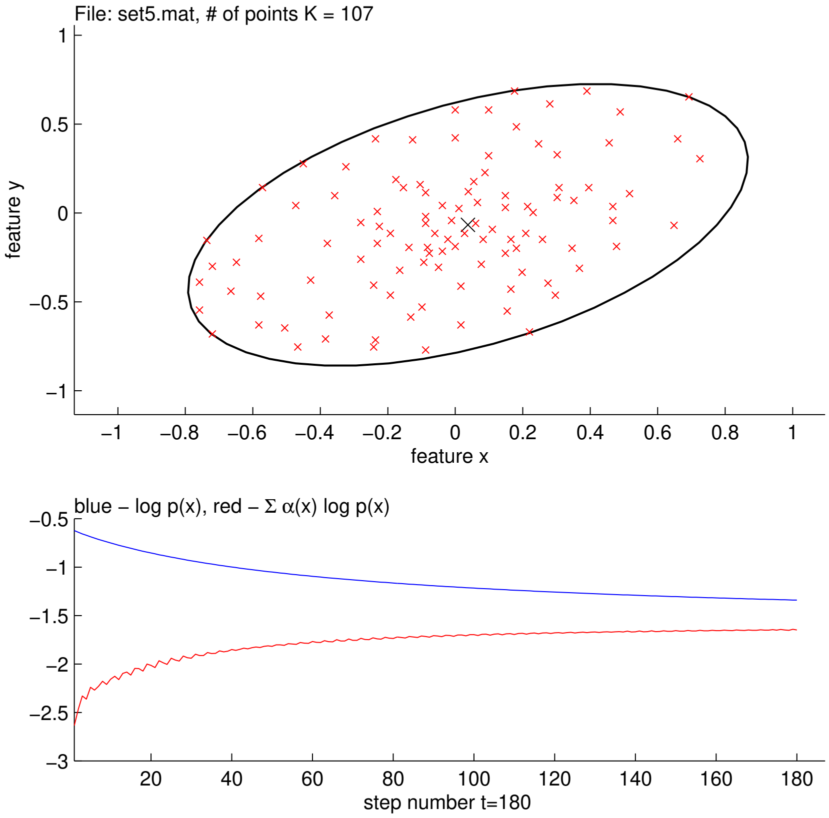
Obrázek 3.5 Algoritmus učení bez učitele pro 4 třídy s normálně rozděleným statistickým modelem a nezávislými příznaky.

3.2.6 Algoritmus minimaxního učení

Tento demonstrační program umožňuje experimenty s algoritmem minimaxního učení pro model s normální rozdělením a korelovanými příznaky.

Uvedený algoritmus hledá statistický model pro jednu třídu z trénovací množiny, jejíž prvky nejsou náhodně vybrány, ale mají to být významní reprezentanti své třídy. Opět se odhadují parametry statistického modelu s normálním rozdělením a korelovanými příznaky. Takto definovaná úloha je ekvivalentní s vyhledáváním elipsy s co nejmenším průměrem, která musí obsahovat všechny prvky z trénovací množiny. Podrobnější popis je uveden v odstavci 6.2.

Průběh řešení je zobrazován ve dvou grafech. První zobrazuje tvar a umístění hledané elipsy, jenž odpovídá statistickému modelu. V druhém grafu se zobrazují dvě funkce, v závislosti na počtu iterací minimaxního algoritmu. Jedna funkce udává současnou kvalitu řešení a druhá dolní odhad optimální kvality řešení. Algoritmus pracuje tak dlouho, dokud není rozdíl mezi odhadem optimální a nalezené kvality řešení menší, než zadaná mez, kterou je možno v demostračním programu libovolně měnit. Ilustrační úloha je vidět na obrázku 3.6.



Obrázek 3.6 Algoritmus minimaxního učení pro jednu třídu s normálně rozděleným statistickým modelem a závislými příznaky.

3.3 Algoritmy vizualizace

Po provedení několika experimentů jsme pro vizualizaci v demonstračních programech zvolili pouze dvourozměrný prostor (2D) na místo třírozměrného prostoru (3D), jehož použití bylo navrhováno v zadání u Andersonovy úlohy. Důvodem tohoto kroku byla nízká rychlost funkcí Matlabu pro zobrazování ve 3D a podstatně menší názornost než ve 2D. V našich demonstračních programech se snažíme zobrazovat řešení úlohy v reálném čase (plynule), proto potřebujeme rychlé vykreslování grafických objektů, které je u standardních matlabovských funkcí nedostatečné. Respektive čas stravený optimalizací vizualizačních funkcí by pravděpodobně neodpovídal dosaženému výsledku.

Náš programový balík obsahuje speciální nástroje, zobrazující vstupní data algoritmů a stav procesu řešení. Uvedené nástroje jsou složeny ze stantardních vizualizačních funkcí Matlabu. Jedinou vyjímkou je zobrazování elipsoidů a obecné kvadratické funkce, které v Matlabu nejsou obsaženy. Řešení těchto dvou problému zde uvedeme.

3.3.1 Zobrazování obecné kvadratické funkce ve 2D

Zde popíšeme zobrazení kvadratické funkce ve 2-rozměrném prostoru, která se dá obecně vyjádřit jako

$$\langle x^T, A_{N\times N}.x\rangle + \langle B_{N\times 1}, x\rangle + C_{1\times 1} = 0 \tag{3.1}$$

kde zápis $\langle x, y\rangle$ vyjadřuje skalární součin a zápis $x \cdot y$ je maticové násobení (používáme stejného zápisu jako v Knize). Prametry $A_{N\times N}, B_{N\times 1}$ a $C_{1\times 1}$ jsou matice daného rozměru a přirozené číslo N udává dimenzi prostoru.

Obecná kvadratická funkce (3.1) se ve 2D ($N = 2$) může v závislosti na parametrech $A_{2\times 2}, B_{2\times 1}, B_{1\times 1}$ zobrazit v několika různých tvarech:

1. Přímka.
2. Dvojice rovnoběžných přímek.
3. Dvojice protínajících se přímek.
4. Elipsa nebo kružnice.
5. Dvojice symetricky umístěných hyperbol.

Naším cílem je diskretizace kvadratické funkce na body pro všechny uvedené případy. Bodově vyjádřenou křivku již Matlab lehce zobrazí.

Pro další analýzu si kvadratickou funkci (3.1) rozepíšeme do tvaru, kde $x = x_1$ a $y = y_2$,

$$ax^2 + bx + cxy + dy + ey^2 + f = 0.$$

Nejprve zjistíme interval hodnot, které můžou nabývat nezávislé proměnné x a y. Kvadratickou fukci vyjádříme ve tvaru $y = f(x)$ jako

$$y = \frac{-(cx + d) \pm \sqrt{(cx + d)^2 - 4e(ax^2 + bx + f)}}{2e}$$

a podobně pro $x = f(y)$

$$x = \frac{-(b + cy) \pm \sqrt{(b + cy)^2 - 4a(ey^2 + dy + f)}}{2e}.$$

Protože požadujeme, aby proměnné x a y byly reálné, musí být determinanty obou uvedených funkcí nezáporná reálná čísla. Z tohoto požadavku určíme hledané intervaly x a y. Pro proměnnou x to je determinant

$$(cx + d)^2 - 4e(ax^2 + bx + f) = x^2(c^2 - 4ea) + x(2cd - 4eb) + d^2 - 4ef \geq 0 \tag{3.2}$$

a pro y

$$(b + cy)^2 - 4a(ey^2 + dy + f) = y^2(c^2 - 4ae) + y(2bc - 4ad) + b^2 - 4af \geq 0. \tag{3.3}$$

Jak je vidět, abychom tyto intervaly určili, musíme vyřešit dvojici jednorozměrných kvadratických funkcí. Zavedeme následující značení

$$\begin{array}{rcl} \alpha &=& (c^2 - 4ae), \\ \beta_1 &=& 2cd - 4eb, \qquad \beta_2 \;=\; 2bc - 4ad, \\ \gamma_1 &=& d^2 - 4ef, \qquad \gamma_2 \;=\; b^2 - 4af. \end{array}$$

Kvadratické funkce (3.2) a (3.3) se pak můžeme zapsat jako

$$\begin{array}{rcl} \alpha x^2 + \beta_1 x + \gamma_1 &=& 0, \\ \alpha y^2 + \beta_2 y + \gamma_2 &=& 0 \end{array}$$

Tabulka 3.1 Intervaly reálných funkčních hodnot kvadratické funkce ve 2D.

D_x	D_y	α	x	y
+	+	+	$(-\infty, x_1\rangle \cup \langle x_2, \infty)$	$(-\infty, y_1\rangle \cup \langle y_2, \infty)$
-	-	+	$(-\infty, \infty)$	$(-\infty, \infty)$
+	+	-	$\langle x_1, x_2\rangle$	$\langle y_1, y_2\rangle$
-	-	-	nic	nic

a jejich řešení se spočítá

$$D_x = \beta_1^2 - 4\alpha\gamma_1, \quad x_1 = \max\left(\frac{-\beta_1 \pm \sqrt{Dx}}{2\alpha}\right),\ x_2 = \min\left(\frac{-\beta_1 \pm \sqrt{Dx}}{2\alpha}\right),$$

$$D_y = \beta_2^2 - 4\alpha\gamma_2, \quad y_1 = \max\left(\frac{-\beta_2 \pm \sqrt{Dy}}{2\alpha}\right),\ y_2 = \min\left(\frac{-\beta_2 \pm \sqrt{Dy}}{2\alpha}\right).$$

Z těchto hodnot a z vlastností kvadratické funkce již určíme intervaly pro proměnné x a y. Výsledek je zapsán v tabulce 3.1.

Nekonečné intervaly v tabulce 3.1 jsou nahrazeny hodnotami, které odpovídájí rozměrům pracovního okna. Pomocí uvedené tabulky a dvojice rovnic (3.2), (3.3), lze spočítat body ve 2D, určující hranici obecné kvadratické funkce. Bodově vyjádřená křivka, respektive dvě křivky, již můžeme snadno zobrazit standardními funkcemi Matlabu.

3.3.2 Zobrazování elipsoidů

Elipsoid (elipsa ve 2D) je velice důležitým tvarem pro vizualizaci v rozpoznávání. Zde vyřešíme zobrazování elipsoidů ve 2D i ve 3D, jenž není v Matlabu implementováno.

Pro zobrazení grafických objektů je v prostředí Matlabu potřeba vyjádření jejich tvaru v diskretizovaných bodech. Pro 3D objekty je to matice bodů popisující drátěný model a ve 2D vektory popisující hranici křivky. My jsme se rozhodli využít funkcí Matlabu, jenž generují matice bodů pro kouli respektive kružnici. Vygenerované matice bodů transfromujeme do zakřiveného prostoru, kde se koule (kružnice) zobrazí jako elipsoid (elipsa) požadovaného tvaru a rozměru. Na následujících řádcích tuto transfromaci popíšeme.

Hranice elipsoidu je v lineárním prostoru X maticově vyjádřena jako

$$(x - \mu)^T \sigma^{-1} (x - \mu) = r^2, \tag{3.4}$$

kde σ je pozitivně definitní symetrická matice udávající tvar elipsoidu, vektor μ je jeho střed a číslo r udává poloměr. Předpokládáme, že se elipsoid (3.4) dá zobrazit ve vyrovnaném lineárním prostrou Y jako kruh, s poloměrem r a středem μ, popsaný rovnicí

$$(y - \mu)^T (y - \mu) = r^2, \tag{3.5}$$

Dále předpokládáme, že hledaná transformace je lineární, to znamená, že bod je v novém lineárním prostoru Y vyjádřen

$$y = Ax, \tag{3.6}$$

kde A je matice určující hledanou transformaci. Díky vlastnostem lineární zobrazení můžeme transformaci určit pro elipsoid a kruh se středem v počátku souřadnic, tj. $\mu = 0$, neboť změnu pozice objektů v obou prostorech lze provést pouhým posunutím o původní hodnotu vektoru μ. Takto specifikovaný elipsoid (3.4) a kruh (3.5) porovnáme díky shodným pravým stranám a po dosazení transfromace (3.6) za proměnou y dostaneme po úpravě výsledný vztah pro převodní matici A. Neboli

$$
\begin{aligned}
y^T y &= x^T \sigma^{-1} x & (3.7)\\
(Ax)^T Ax &= x^T \sigma^{-1} x & (3.8)\\
x^T A^T Ax &= x^T \sigma^{-1} x & (3.9)\\
A^T A &= \sigma^{-1}. & (3.10)
\end{aligned}
$$

Matici A získáme rozkladem matice σ pomocí Choleskyho trasfomace, která je standardní součástí Matlabu. Výsledný bodový popis elipsoidu X dostaneme inverzní transformací (3.6) z množiny bodů Y a posunutím o vektor μ, kde popis koule (kruhu) Y generuje standardní funkce Matlabu neboli

$$x = A^{-1} y + \mu, \quad y \in Y.$$

Popsaná transfomace funguje pro obecný konečně-rozměrný prostror, my jsme ji samozřejmě implementovali jen pro případ 2D a 3D prostoru.

3.4 Program pro vytváření syntetických dat a funkce pro práci s nimi

V našem programovém balíku pracujeme se dvěma typy tříd objektů. Zaprvé to jsou třídy popsané konečnými množinami bodů, tedy jejich výčtem. Druhým případem jsou třídy popsané směsí normálních rozdělení, respektive parametry těchto rozdělení. Pro zaznamenání uvedených tříd jsme navrhli jednoduchou datovou strukturu, která se skládá z matic a vektorů, jenž lze snadno zaznamenat do standardních datovým souborů Matlabu, označovaných příponou *.MAT. Na následujících řádcích ji stručně popíšeme.

Popis struktury datových souborů:

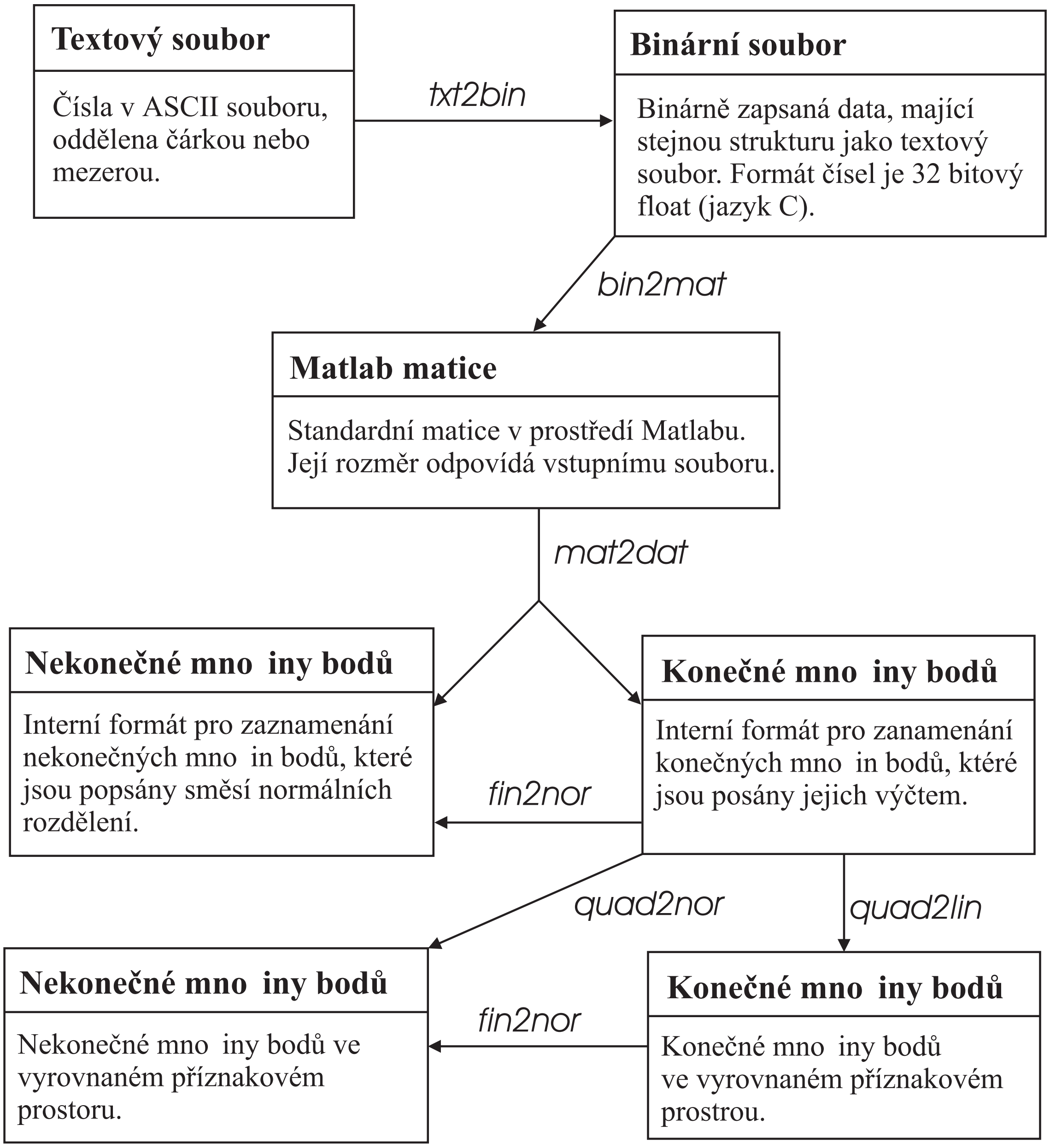
- Vektor *id*. Tento vektor obsahuje posloupnost znaků (string), která popisuje typ množiny. Vektor může nabývat dvou různých hodnot:
  1. *id*=„Finite sets, Enumeration“, pro případ konečných množin určených výčtem bodů.
  2. *id*=„Infinite sets, Normal distributions“, pro nekonečné množiny bodů popsané parametry směsí gaussovských rozdělení.
- Celé kladné číslo N. Udává dimenzi příznakového prostoru.
- Vektor K $[1 \times |K|]$. Číslo $|K|$ určuje počet tříd zaznamenaných v datové množině. Pro třídy určené výčtem bodů, udává číslo $K(j)$ počet bodů v j-té třídě a v případě nekonečných množin udává počet dvojic (μ^j, σ^j), což jsou parametry normálního rozdělení popisující danou třídu. Počet všech zaznamenaných bodů, respektive parametrů normálního rozdělení, je určen součtem složek vektoru K, neboli $M = \sum_{j=1}^{|K|} K(j)$.
- V závislosti na typu dat jsou třídy určeny:
  1. Výčet bodů v matici X $[N \times M]$. Matice je tvořena vektory x, popisující jednotlivé objekty. Sloupec popisuje jeden objekt (sample), číslo řádku odpovídá číslu příznaku.

2. Matice středních hodnoty MI $[N \times |K|]$ a kovarianční multimatice $SIGMA$ $[N \times (N \cdot |K|)]$. Střední hodnoty jsou v matici MI uspořádány stejně jako objekty v matici X, tedy co sloupec do střední hodnota pro danou třídy. Multimatici $SIGMA$ tvoří kovarianční matice umístěné vedle sebe.

- I, vektor $[1 \times |K|]$. Zde jsou uložena celá čísla (indexy) určující příslušnost jednotlivých objektů ke třídám. Hodnota čísla $I(j)$ udává index třídy, do které patří j-tý vektor z matice X, respektive j-tý parametr z matic MI a $SIGMA$. Indexy tříd jsou přirozená čísla od 1 do $|K|$.

Všechny demonstrační programy používají jako vstup pouze soubory s uvedenou datovou strukturu. Každá konkrétní úloha je zaznamenána právě v jednom takovém souboru. Náš programový balík obsahuje nástroj pro jejich interaktivní vytváření ve 2D prostoru, jenž lze spustit ze všech demostračních programů a může být i samostatně používán. Program je plně ovladatelný pomocí myši. V případě zadávání konečných množin uživatel specifikuje jednotlivé body kliknutím myši na požadovanou pozici. Při vytváření směsi normálních rozdělení se stejným způsobem určí jejich střední hodnota a pomocí vstupní řádky se vloží kovarianční matice. Zadaná data se okamžitě zobrazují buď ve formě bodů pro první případ a nebo v druhém případě jako elipsy s poloměrem vhodně zvoleným pro dané měřítko.

Dalším možným způsobem vytváření uvedených datových souborů je konverze z jiných formátů. Nutným přepokladem je, aby tato externí data bylo možné převést do matlabovské matice nebo do textového souboru. Pro takto zaznamenaná data jsme implementovali převodní funkce do našeho interního formátu. Na obrázku 3.7 jsou zobrazeny datové soubory, s kterými pracuje náš programový balík. Jednotlivé soubory jsou pospojovány orientovanými hranami, u nichž je uveden název funkce zajišťující převod daným směrem.



Obrázek 3.7 Používané datové soubory a funkce pro převod mezi nimi.

4. Lineární diskriminační funkce

V následujících kapitolách popíšeme algoritmy rozpoznávání, jak jsou implementovány v našem programovém balíku. Během popisu se budeme držet značení, které je používáno v Knize. Navíc jsme toto značení striktně dodržovali i při psaní zdrojových kódů pro Matlab. Na následujících řádcích uvádíme základní pojmy, jenž budeme bezprostředně potřebovat. Pro podrobnější popis samozřejmě odkazujeme na Knihu.

Objekt rozpoznávání je popsán vektorem příznaků x^j v lineárním n-rozměrném prostoru X. Horní index rozhoduje, do které třídy vektor x^j patří. Třídy jsou v Knize značeny dvěma způsoby. Zaprvé množinou bodů $x \in X_i$, která označuje i-tou třídu. Zadruhé je i-tá třída také značena jako množina indexů J_i a tedy zápis x^j pro $j \in J_1$ je to samé jako $x \in X_1$. V obou případech se jedná o objekty ze třídy 1. Jak jsme již uvedli v předchozí kapitole, je skalární součin vektorů x a y zapisován $\langle x, y \rangle$. Je-li A matice pak $A \cdot x$ vyjadřuje maticové násobení.

4.1 Lineární rozhodovací strategie

Obsahem této kapitoly jsou metody hledající lineární rozhodovací strategii ze zadané trénovací množiny. Data jsou v trénovací množině rozdělena podle tříd, do kterých patří, jedná se tedy o učení s učitelem. Hledanou rozhodovací strategii používá klasifikátor pro zařazování pozorovaného objektu do tříd. Strategii q lze také popsat jako zobrazení příznakového prostoru X na množinu indexů tříd J, neboli $q\colon X \to J$.

Převážně se budeme zabývat situací, kdy mohou objekty patřit pouze do dvou tříd $J = \{1, 2\}$, tzv. dichotomie. V tomto případě je lineární rozhodovací strategie ekvivalentní s rozdělením prostoru příznaků X, na dva podprostory pomocí nadroviny $\langle \alpha, x \rangle = \theta$. Souvislost mezi strategií rozhodování a rozdělující nadrovinou lze vyjádřit jako

$$q(x) = \begin{cases} 1, & \text{pro} \quad \langle \alpha, x \rangle \geq \theta, \\ 2, & \text{pro} \quad \langle \alpha, x \rangle < \theta. \end{cases} \tag{4.1}$$

kde je $\alpha \in X$ vektor a θ prahová hodnota.

Uveďme ještě, že v Knize není přesně definován pojem diskriminační funkce, my ho zde budeme používat ve smyslu definice rozhodovací strategie (4.1), kde se jedná o lineární diskriminační funkci. V další kapitole ještě popíšeme kvadratickou diskriminační funkci.

Všechny algoritmy popisované v této kapitole hledají rozdělující nadrovinu $\langle \alpha, x \rangle = \theta$, ale lze je rozdělit podle toho, jestli je jejich vstupní trénovací množina konečná nebo nekonečná. Rozdělení dvou konečných množin bodů řeší perceptron, Kozincův algoritmus plus jeho modifikace a Linear Support Vector Machine. Rozdělením více konečných množin bodů se zabývají Fisherovy klasifikátory, které jsme umístili do samostatného odstavce. Nekonečné množiny bodů jsou v našem případě určeny směsí parametrů normálního (gaussovského) rozdělení a tímto problémem se zabývá zobecněná Andersonova úloha.

Pro oba případy, konečné i nekonečné množiny X, lze lineární rozhodovací strategii (4.1) transformovat na ekvivalentní problém, který je jednodušší a v Knize je používán. V dalším odstavci tyto transformace popíšeme.

4.2 Ekvivalentní formulace lineární rozhodovací strategie

Rozhodovací strategii (4.1) lze vyjádřit ekvivalentním způsobem, který zde popíšeme. Zaprvé nebudeme hledat nadrovinu libovolně umístěnou v prostoru, ale pouze takovou, která prochází počátkem souřadnic. Pak už není třeba hledat normálu α a práh θ v původním prostoru X, ale pouze určovat normálu α' v novém příznakovém prostoru X'. Tato transformace zvýší dimenzi původního prostoru z n na $n+1$ a to tak, že každému vektoru x přidáme jednu konstantní souřadnici. Formálně lze tuto transformaci zapsat následovně.

Formulace 4.1 Přidání konstantní souřadnice.

Nechť je X n-rozměrný lineární prostor. Rozdělující nadrovina $\langle \alpha, x\rangle = \theta$ dělí prostor X na dvě množiny bodů X_1 a X_2 podle vztahu

$$\begin{aligned} \langle \alpha, x\rangle &\geq \theta, \quad x \in X_1, \\ \langle \alpha, x\rangle &< \theta, \quad x \in X_2. \end{aligned} \tag{4.2}$$

Rozdělení prostoru podle (4.2) lze převést na rovnocenou strategii, kdy se rozhoduje pouze podle znaménka lineární fukce $\langle \alpha', x'\rangle$, tj.

$$\begin{aligned} \langle \alpha', x'\rangle &\geq 0, \quad x' \in X_1', \\ \langle \alpha', x'\rangle &< 0, \quad x' \in X_2'. \end{aligned} \tag{4.3}$$

Toho se dosahuje následující transformací.
Všechny vektory z X se transformují na X' podle vztahu

$$x' = \{x_1, x_2, ..., x_n, 1\},$$

rozdělující nadrovina v novém prostoru X' bude procházet počátkem souřadnic a její normála se spočítá jako

$$\alpha' = \{\alpha_1, \alpha_2, ..., \alpha_n, -\theta\}.$$

Inverzní transformace z prostoru X' do X se provede analogicky.

▲

Upravená úloha (4.3) se skládá ze dvou soustav nerovnic. Protože transformovaná nadrovina prochází počátkem souřadnic, můžeme obě soustavy spojit do jedné a to tak, že body z jedné množiny prohodíme podle počátku souřadnic.

Formulace 4.2 Přehození bodů podle počátku souřadnic.

Nechť je X' lineární prostor a nadrovina určená normálou α' ho rozděluje na dvě množiny X_1' a X_2' podle vztahu (4.3). Zavedeme novou množinu X'', která obsahuje vektory, pro něž platí

$$x'' = \begin{cases} \;\; x', & \textit{pro} \quad x' \in X_1', \\ -x, & \textit{pro} \quad x' \in X_2' \end{cases}$$

Po této transformaci je úloha (4.3) ekvivalentní s úlohou nalezení α', které splňuje

$$\langle \alpha', x'' \rangle \geq 0, \quad x'' \in X''. \tag{4.4}$$

▲

4.3 Rozdělení konečných množin bodů

V této části popíšeme algoritmy řešící úlohu rozdělení konečných množin bodů. Uvedenou úlohu budeme dále dělit na jednoduché rozdělení, optimální rozdělení a ε-optimální rozdělení.

Ve formulaci použijeme ekvivalentní vyjádření lineární rozhodovací strategie podle (4.1), takže hledáme pouze vektor α, určující nadrovinu procházející počátkem souřadnic.

Úloha jednoduchého rozdělení bodů se snaží nalézt libovolnou rozdělující nadrovinu, která od sebe oddělí rozdělované množiny bodů.

Formulace 4.3 Jednoduché rozdělení konečných množin bodů.
Nechť je J konečná množina indexů, která je rozložena na dvě množiny J_1 a J_2, a $X = \{x^j, j \in J\}$ je konečná množina bodů v lineárním prostoru. Hledá se vektor α, který vyhovuje soustavě nerovnic

$$\begin{aligned} \langle \alpha, x^j \rangle &> 0, \quad j \in J_1, \\ \langle \alpha, x^j \rangle &< 0, \quad j \in J_2. \end{aligned} \tag{4.5}$$

Této úloze říkáme jednoduché rozdělení konečných množin bodů.

▲

Existuje-li jednoduché rozdělení, lze také nalézt rozdělení optimální. V tomto případě se snažíme umístit rozdělující nadplochu tak, aby ten nejbližší bod z obou konvexních obalů rozdělovaných množin byl od rozdělující nadplochy co nejdále. Uvedená úloha vede na problém kvadratického programování na konvexní množině.

Formulace 4.4 Optimální rozdělení konečných množin bodů.
Má-li soustava (4.5) řešení, pak lze také hledat α, které při co největší kladné hodnotě čísla r vyhovuje soustavě

$$\begin{aligned} \left\langle \tfrac{\alpha}{|\alpha|}, x^j \right\rangle &\geq r, \quad j \in J_1, \\ \left\langle \tfrac{\alpha}{|\alpha|}, x^j \right\rangle &\leq -r, \quad j \in J_2. \end{aligned}$$

Jinak řečeno, hledá se vektor α vyhovující kritériu

$$\alpha = \operatorname*{argmax}_{\alpha} \min \left(\min_{j \in J_1} \frac{\langle \alpha, x^j \rangle}{|\alpha|}, \min_{j \in J_2} \frac{-\langle \alpha, x^j \rangle}{|\alpha|} \right). \tag{4.6}$$

Úloze (4.6) říkáme optimální rozdělení konečných množin bodů.

▲

V některých případech nepotřebujeme nalézt optimální rozdělující nadrovinu, ale stačí nám nalézt takové řešení, které zajišťuje, že nejbližší bod z konvexních obalů množin nebude od rozdělující nadroviny vzdálen méně než zadaná vzdálenost ε. Takovéto rozdělení definuje následující úloha.

Formulace 4.5 ε-optimální rozdělení konečných množin bodů.

Nechť je α^* řešením úlohy optimálního rozdělení konečných množin bodů *(4.6). Pak vektor α, pro který platí*

$$\min_{j \in J} \left\langle \frac{\alpha^*}{|\alpha^*|}, x^j \right\rangle - \min_{j \in J} \left\langle \frac{\alpha}{|\alpha|}, x^j \right\rangle \leq \varepsilon, \tag{4.7}$$

bude ε-optimálním řešením úlohy *rozdělení konečných množin bodů. A úloze hledající α podle (4.7) říkáme* ε-optimální rozdělení konečných množin bodů.

▲

Dále popíšeme iterační algoritmy obsažené v našem programovém balíku, které řeší jednoduché rozdělení (Kozincův algoritmus, perceptron) a ε-optimální rozdělení (modifikace Kozincova algoritmu). Jejich podrobná analýza je uvedena v Knize.

Pro hledání optimálního řešení jsme implementovali metodu *Linear Support Vector Machine* (SVM), která tuto úlohu převádí na problém kvadratického programování. Uvedená metoda není v Knize popsána, přesto ji zde uvádme pro porovnání s iteračními algoritmy. Dale také proto, že Matlab obsahuje účinné optimalizační nástroje, pomocí nichž lze SVM implementovat velice jednoduše a efektivně. Navíc je problém optimáního rozdělení potřeba při řešení zobecněné Andersonovy úlohy.

4.3.1 Vstup a výstup algoritmů

Zde popíšeme vstupní a výstupní argumenty algoritmů řešící úlohu rozdělení konečných množin bodů. Algoritmy jsou navrženy tak, abychom je mohli spustit z libovolného stavu a řídit počet jejich iterací, pro tento účel jsou definovány speciální argumenty.

Vstupem algoritmů je konečná množina bodů X, ve které je každý bod označen indexem podle toho, do jaké třídy patří. První třída je určena množinou indexů J_1 a druhá třída J_2. Dálší vstupní argument t_{max} určuje maximální počet iterací algoritmu. To znamená, jestliže není řešení nalezeno do iterace t_{max}, pak v tomto kroku algoritmus nejpozději skončí. V případě, kdy chceme pustit algoritmus z nějakého předem definovaného stavu, můžeme použít trojici argumentu t_s, α_s a θ_s, které jednoznačně určují stav řešení.

Výstupem algoritmů je rozdělující nadrovina určená normálovým vektorem α a prahem θ. Protože lze algoritmus ukončit dříve než nalezne řešení, je další výstupní hodnotou proměnná *solution*. Je-li řešení nalzeno pak je $solution = 1$, v opačném případě je $solution = 0$. Číslo kroku (počet iterací), ve kterém algortimus skončil je určeno návratovou hodnotou proměnné t.

4.3.2 Perceptron

Tento algoritmus využívá adaptačního pravidla perceptronu, který popsal F. Fosenblatt v [7].

Algoritmus na začátku inicializuje řešení na hodnotu $\alpha_1 = 0$. V každém kroku iterace pak hledá takový bod x_t, který je špatně klasifikován, tj. nevyhovuje vztahu (4.5). Nové řešení se určí jako součet stavajícího řešení a špatně klasifikovaného bodu, neboli $\alpha_{t+1} = \alpha_t + x_t$. Algoritmus iteruje tak dlouho, dokud nejsou všechny body z X správně klasifikovány. Pokud řešení existuje, pak ho algoritmus nalezne v konečném počtu kroků. V opačném případě se neukončí.

Algoritmus 4.1 Perceptron

Vstup: $X, J \in J1 \cup J2, t_{max}, t_s, \alpha_s, \theta_s$

Výstup: $\alpha, \theta, solution, t$

1. Provede se transformace množiny X na X‘ jak je posána v (4.2).

$$x^{‘j} = \begin{cases} \{x_1^j, ...x_n^j, 1\} & \text{pro} \quad j \in J_1, \\ -\{x_1^j, ...x_n^j, 1\} & \text{pro} \quad j \in J_2. \end{cases}$$

   Je-li zadán počáteční stav $\{t_s, \alpha_s, \theta_s\}$, transformuje se také $\{\alpha_s, \theta_s\}$ na α'_s, jako

$$\alpha' = \{\alpha_{s_1}, ..., \alpha_{s_n}, -\theta_s\}$$

   Po této transformaci se původní úloha redukovala na nalezení α', které splňuje

$$\langle \alpha', x'^j \rangle \geq 0, \quad j \in J$$

   Pro přehlednost budeme α', X' a x' dále označovat bez čárky jako α, X a x.

2. Inicializace algoritmu. Je-li zadán počáteční stav $\{t_s, \alpha_s, \phi_s\}$ nastav

$$t = t_s, \alpha_t = \alpha_s,$$

   jinak

$$t = 1, \alpha_1 = 0.$$

3. Hledá se vektor $x^j, j \in J$, který vyhovuje podmínce

$$\langle \alpha_t, x^j \rangle < 0.$$

4. Pokud takový vektor x^j neexistuje, znamená to, že α_t je řešením ulohy. Nastav argument $solution = 1$ a přejdi na krok 7.

5. Pokud vektor x^j existuje, označíme ho x_t, a nalezne se nové α_{t+1} podle

$$\alpha_{t+1} = \alpha_t + x_t.$$

6. Zvětší se počítadlo kroku $t = t + 1$. Je-li $t < t_{max}$, a pokračuje se krokem 3, jinak se nastaví argument $solution = 0$.

7. Ukončení algoritmu. Provede se inverzní transformace z kroku 1. Nalezené α_t opět označíme jako α' a vypočteme výsledné řešení

$$\begin{aligned} \alpha &= \{\alpha'_1, ...\alpha'_n\}, \\ \theta &= -\alpha'_{n+1}. \end{aligned}$$

4.3.3 Kozincův algoritmus

Tento algoritmus je pojmenován po ruském matematikovi B. N. Kozincovi, který jej publikoval v [5].

Metoda modifikuje perceptron, jenž adaptuje stavající řešení α_t pouhým přičtením špatně klasifikovaného vektoru x_t. Kozincův algoritmus adaptuje řešení tak, aby nové α_{t+1} leželo na přímce určené body α_t a x_t, a aby bylo co nejblíže počátku souřadnic.

Algoritmus 4.2 Kozincův algoritmus

Vstup: $X, J \in J1 \cup J2, t_{max}, t_s, \alpha_s, \theta_s$

Výstup: $\alpha, \theta, solution, t$

1. Provede se transformace množiny X na X' jak je posána v (4.2).

$$x'^j = \begin{cases} \{x_1^j, ...x_n^j, 1\} & \text{pro} \quad j \in J_1, \\ -\{x_1^j, ...x_n^j, 1\} & \text{pro} \quad j \in J_2. \end{cases}$$

   Je-li zadán počáteční stav $\{t_s, \alpha_s, \theta_s\}$, transformuje se také $\{\alpha_s,\theta_s\}$ na α'_s, jako

$$\alpha' = \{\alpha_{s_1}, ..., \alpha_{s_n}, -\theta_s\}.$$

   Po této transformaci se původní úloha redukovala na nalezení α‘, které splňuje

$$\langle \alpha', x'^j \rangle \geq 0, \quad j \in J.$$

   Pro přehlednost budeme α', X' a x' dále označovat bez čárky jako α, X a x.

2. Inicializace algoritmu. Je-li zadán počáteční stav $\{t_s, \alpha_s, \theta_s\}$ nastav

$$t = t_s, \alpha_t = \alpha_s,$$

   jinak

$$t = 1, \alpha_1 = x.$$

   kde x je první bod z množiny X.

3. Hledá se vektor $x^j, j \in J$, který vyhovuje podmínce

$$\langle \alpha_t, x^j \rangle < 0.$$

4. Pokud takový vektor x^j neexistuje, znamená to, že α_t je řešením ulohy. Pak se nastaví argument $solution = 1$ a přejde se na krok 7.

5. Pokud vektor x^j existuje (označíme ho x_t), nalezne se α_{t+1} tak, aby to byl bod na přímce proložené body α_t a x_t, který je nejblíže počátku souřadnic, což se vypočte jako

$$\alpha_{t+1} = (1-k) \cdot \alpha_t + k \cdot x_t, \quad k \in R,$$

   kde

$$k = \operatorname*{argmin}_k |(1-k) \cdot \alpha_t + k.x_t|.$$

6. Zvětší se počítadlo kroku $t = t + 1$. Je-li $t < t_{max}$, pak se pokračuje krokem 3, jinak se nastaví argument $solution = 0$.

7. Ukončení algoritmu. Provede se inverzní transformace z kroku 1. Nalezené α_t opět označíme jako α' a vypočteme výsledné řešení

$$\begin{aligned}\alpha &= \{\alpha'_1, ...\alpha'_n\},\\ \theta &= -\alpha'_{n+1}.\end{aligned}$$

V pátém kroku algoritmu se hledá bod α_{t+1}, nejbližší počátku souřadnic, který leží na přímce určené dvěma body α_t a x_t. Tento problém jsme vyřešili v následujícím odstavci.

Hledání nejbližšího bodu od počátku souřadnic

Hledáme α_{t+1}, tak aby platilo

$$\alpha_{t+1} = (1-k)\cdot\alpha_t + k\cdot x_t, \quad k \in R,$$

kde

$$k = \operatorname*{argmin}_k |(1-k).\alpha_t + k.x_t|. \tag{4.8}$$

Minimalizovanou funkci označíme $f(k)$ a zapíšeme ji vektorově jako

$$\begin{aligned}f(k) &= [\alpha_t\cdot(1-k)+k\cdot x_t]^T\cdot[\alpha_t\cdot(1-k)+k\cdot x_t] =\\ &= \alpha_t^T\cdot\alpha_t + 2\cdot k\cdot(x_t-\alpha_t)^T\cdot\alpha_t + k^2\cdot(x_t-\alpha_t)^T\cdot(x_t-\alpha_t).\end{aligned}$$

Funkce $f(k)$ je konvexní, takže dosahuje minima v bodě k, kdy je její první derivace nulová

$$\begin{aligned}\frac{df(k)}{dk} &= 2\cdot(x_t-\alpha_t)^T\cdot\alpha_t + 2\cdot k\cdot(x_t-\alpha_t)^T\cdot(x_t-\alpha_t) \;=\; 0,\\ k &= -\frac{(x_t-\alpha_t)^T\cdot\alpha_t}{(x_t-\alpha_t)^T\cdot(x_t-\alpha_t)}.\end{aligned}$$

V ostatních algoritmech budeme tuto úlohu označovat její definicí (4.8), která je více názornější než její řešení.

4.3.4 Algoritmus ε-optimální rozdělení konečných množin bodů

Zde popíšeme algoritmus, který hledá ε-optimální rozdělení konečných množin bodů, jak jsme jej formulovali v 4.5. Uvedenou úlohu řeší modifikace Kozincova algoritmu v kroku 4.23, kde se kontroluje správnost klasifikace rozdělovaných bodů. Tato podmínka se zpřísní tak, aby klasifikované body vyhovovaly definici ε-optimálního rozdělení (4.5).

Algoritmus 4.3 **ε-optimální rozdělení konečných množin bodů**

Vstup: $X, J \in J1 \cup J2, \varepsilon, t_{max}, t_s, \alpha_s, \theta_s$

Výstup: $\alpha, \theta, solution, t$

1. Provede se transformace množiny X na X' jak je posána v (4.2).

$$x'^j = \begin{cases} \{x_1^j, ...x_n^j, 1\} & \text{pro} \quad j \in J_1, \\ -\{x_1^j, ...x_n^j, 1\} & \text{pro} \quad j \in J_2. \end{cases}$$

   Je-li zadán počáteční stav $\{t_s, \alpha_s, \theta_s\}$, transformuje se také $\{\alpha_s, \theta_s\}$ na α'_s, jako

$$\alpha' = \{\alpha_{s_1}, ..., \alpha_{s_n}, -\theta_s\}.$$

   Po této transformaci se původní úloha redukovala na nalezení α', které splňuje

$$\langle \alpha', x'^j \rangle \geq 0, \quad j \in J.$$

   Pro přehlednost budeme α', X' a x' dále označovat bez čárky jako α, X a x.

2. Inicializace algoritmu. Je-li zadán počáteční stav $\{t_s, \alpha_s, \phi_s\}$ nastav

$$t = t_s, \quad \alpha_t = \alpha_s,$$

   jinak

$$t = 1, \quad \alpha_1 = x,$$

   kde x je první bod z množiny X.

3. Nejprve se nalezne x^t, tak že

$$x_t = \min_{j \in J} \left\langle \frac{\alpha_t}{|\alpha_t|}, x^j \right\rangle$$

   a s takto nalezeným x_t se následně ohodnotí nerovnice

$$|\alpha_t| - \left\langle \frac{\alpha_t}{|\alpha_t|}, x_t \right\rangle \leq \varepsilon.$$

4. Když je předchozí nerovnice splněna, potom je α_t řešením úlohy. Nastav $solution = 1$ a přejdi na krok 7.

5. Není-li podmínka splněna vypočte se nová hodnota vektoru α_{t+1} jako

$$\alpha_{t+1} = \alpha_t(1-k) + kx_t,$$

   kde

$$k = arg \min_k |(1-k) \cdot \alpha_t + k \cdot x_t|. \tag{4.9}$$

6. Zvětši se počítadlo kroku $t = t + 1$. Je-li $t < t_{max}$, pak se pokračuje krokem 3, jinak se nastaví argument $solution = 0$.

7. Ukončení algoritmu. Provede se inverzní transformace z kroku 1. Nalezené α_t opět označíme jako α' a vypočteme výsledné řešení

$$\begin{aligned} \alpha &= \{\alpha'_1, ...\alpha'_n\}, \\ \theta &= -\alpha'_{n+1}. \end{aligned}$$

4.3.5 Linear Support Vectors Machine

Zde uvádíme metodu, jenž hledá optimální rozdělení konečných množin bodů. Tato metoda řeší problém rozdělení konečného počtu bodů jiným způsobem, než iterační algoritmy popsané výše, takže umožňuje uživateli jiný pohled na věc. Navíc, díky použití optimalizační toolboxu Matlabu, jí lze implemetovat velmi jednoduše a efektivně, což se nám bude hodit při řešení dalších algoritmů, především zobecněné Andersonovy úlohy, u které je tento problém jednou z podúloh jejího řešení. Autorem metody je Vapnik [11], my jsme však čerpali informace především z popisu od Burgese [3].

Jak jsme již popsali výše, lze použitím transfromací z odstavce (4.2) převést úlohu optimálního rozdělení na optimalizační problém

$$\max r \tag{4.10}$$

za podmínky

$$\left\langle \frac{\alpha}{|\alpha|}, x \right\rangle \geq r, \quad x \in X. \tag{4.11}$$

Vektor α určuje normálu rozdělující nadroviny procházející počátkem souřadnic. Číslo r je vzdálenost nejbližšího bodu z konvexního obalu množiny X. Maximalizace r znamená, že se snažíme umístit rozdělující nadrovinu tak, aby její vzdálenost od konvexního obalu X byla co největší.

Uvedenou úlohu převedeme na normalizovaný tvar kvadratického programování s omezením konvexní množinou, který je

$$\min_x \frac{1}{2}\langle x, H \cdot x\rangle + \langle f, x\rangle, \quad A \cdot x \leq b,$$

kde H je pozitivně definitní matice a f vektor. Dále A je matice a b vektor určující omezující podmínky pro hledané x. Přesně takto formulovaná úloha je řešena v optimalizačním toolboxu Matlabu.

Rozdělující nadrovina je jednoznačně určena normálovým vektorem α, přičemž záleží pouze na jeho směru a ne na jeho absloutní velikosti. Tuto vlastnost využijeme a přepíšeme soustavu nerovnic (4.11) do tvaru

$$c\langle \alpha, x\rangle \geq rc|\alpha|, \tag{4.12}$$

kde c je libovolné kladné číslo. Toto číslo zvolíme tak, aby pravá strana nerovnice (4.12) byla rovna 1, tj. $rc|\alpha| = 1$ a jednoduše vyjádříme číslo r jako

$$r = \frac{1}{c|\alpha|}.$$

Zavedeme nový vektor $\alpha' = c\alpha$ a přepíšeme původní problém (4.10) do tvaru

$$\max_{|\alpha'|} \frac{1}{|\alpha'|} \approx \min_{|\alpha'|} \langle \alpha', E.\alpha'\rangle, \tag{4.13}$$

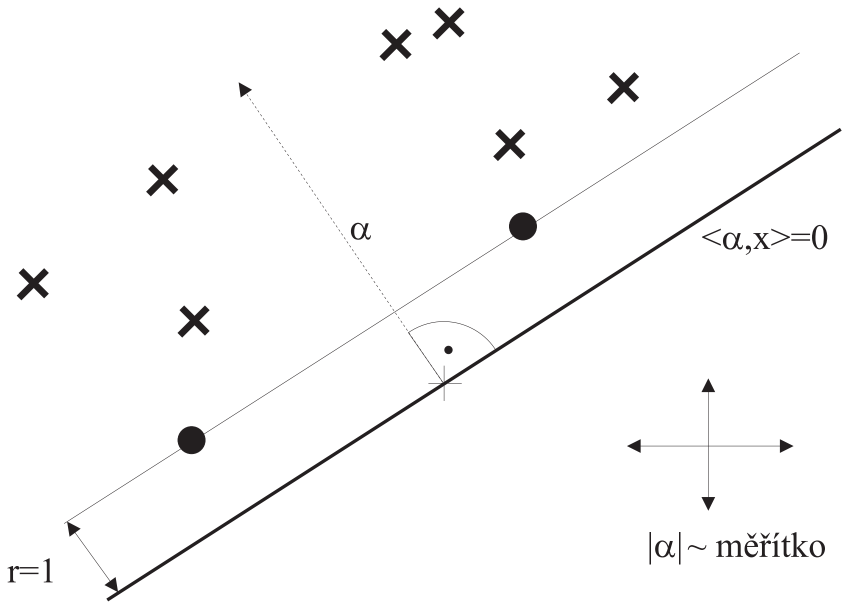
kde E je jednotková matice. Nová soustava nerovnic určujících podmínky řešení je

$$\langle \alpha', x\rangle \geq 1, x \in X,$$

což je ekvivalentní s vyjádřením

$$\langle \alpha', -x \rangle \leq -1, x \in X. \tag{4.14}$$

A tímto způsobem jsme převedli hledání optimálního rozdělení konečných množin bodů na normovanou úlohu kvadratického programování (4.13) s omezením konvexní množinou (4.14).



Obrázek 4.1 Ilustrace metody Linear Support Vector Machine.

Popsaný postup ilustruje obrázek 4.1 a zjednodušeně ho lze popsat takto: Vzdálenost nejbližšího bodu $x \in X$ od nadroviny dané vektorem α, zvolíme $r = 1$. Změnou velikosti vektoru α pak měníme měřítko zobrazení a tím i skutečnou vzdálenost bodů o rozdělující nadroviny.

Poznamenejme ještě, že polohu rozdělující nadroviny určují pouze body $x \in X$, které jsou od ní nejblíže. Tyto body se nazývají Support Vectors a na obrázku 4.1 jsou znázoněny černými kolečky.

4.4 Fisherovy klasifikátory

Fisherův klasifikátor rozděluje konečné množiny bodů stejně jako algoritmy v sekci 4.3. Od těchto algoritmů se ale liší tím, že nerozděluje jen dvě množiny bodů, ale rozděluje libovolný konečný počet tříd, přičemž se hledá jednoduché rozdělení. Fisherův klasifikátor lze chápat jako řešení konečné soustavy lineárních nerovnic, za předpokladu, že takové řešení existuje. Zde uvedeme formulaci, jak je popsána v Knize.

Formulace 4.6 Formulace Fisherova klasifikátoru.

Nechť je X n-rozměrný lineární prostor, K celé číslo a α^k, $k = 1, ..., K$, je K vektorů, které určují rozklad prostoru X na K konvexních kuželů $X(k), k = 1, ..., K$, tak, že bod $x \in X$ leží v množině $X(k)$, je-li splněno

$$\langle \alpha^k, x \rangle > \langle \alpha^j, x \rangle, \; j = 1, 2, ...K, \; j \neq k.$$

Pokud rozklad prostoru X na kužely uvedených vlastností existuje, lze ho nalézt pomocí Fisherova klasifikátoru.

▲

Řešení uvedeného problému spočívá v převedení na problém nalezení vektoru α, který splňuje

$$\langle \alpha, y \rangle > 0, \quad y \in Y, \tag{4.15}$$

kde Y je opět konečná množina bodů, vzniklá transformací množin $X(k), k = 1, ..., K$. Tato transfomace je uvedena v Knize, pátá kapitola, odstavec čtvrtý, verš čtyři. Transformovnou úlohu (4.15), lze řešit libovolným algoritmem z předchozího odstavce 4.3.

My jsme použili perceptronu a Kozincova algoritmu, které umožňují vyjádřit uvedenou transformaci strukturou programu tak, že se nemusí explicitně provádět.

4.4.1 Fisherův klasifikátor pomocí perceptronu

Vstupem algoritmu je množina bodů X, kdy každý bod $x^j \in X$ má svůj index j podle toho, do které třídy patří. Dalším vstupem je číslo K a množiny indexů $J_1, J_2, ..., J_K$, které tyto třídy určují. Maximální počet iterací je dán číslem t_{max}. Pro případ, kdy potřebujeme algoritmus spustit z definovaného stavu, je zde definováno také číslo počáteční iterace t_s a tomu odpovídající řešení $\alpha_s^j, j = 1, ..., K$.

Výstupem algoritmu je množina vektorů $\alpha^j, j = 1, ..., K$, které přísluší jednotlivým třídám (viz formulace 4.6). Tyto vektory určují Fisherův klasifikátor. Jsou-li α^j řešením úlohy, je nastavena návratová hodnota proměnné $solution = 1$, v opačném případě je $solution = 0$. Počet iterací, které algoritmus provedl, je vrácen v proměnné t.

Algoritmus 4.4 Fisherův klasifikátor pomocí perceptronu

Vstup: $X, J = J_1 \cup J_2 \cup ... \cup J_K, K, t_{max}, t_s, \alpha_s^j$ *kde* $j = 1, 2, ...K$

Výstup: α^j *kde* $j = 1, 2, ...K, solution, t$

1. Inicializace algoritmu. Je-li zadán počáteční stav $\{t^s, \alpha_s^j\}$ přiřaď

$$\begin{aligned} t &= t^s, \\ \alpha_t^j &= \alpha_s^j, \quad j = 1, 2, ...K, \end{aligned}$$

   jinak

$$\begin{aligned} t &= 1, \\ \alpha_t^j &= x^j, \quad x^j \in X, qj = 1, 2, ...K, \end{aligned}$$

   kde x^j je j-tý vektor z množiny X.

2. Hledá se taková trojice k, i a x, pro kterou platí $x \in X(k)$, $j \neq k$ a

$$j = \operatorname*{argmax}_i \langle \alpha_t^i, x \rangle, \quad i = 1, 2, ...K.$$

3. Jestliže taková trojice neexistuje jsou vektory $\alpha_t^j, j = 1, 2, ...K$, řešením úlohy. Pak se nastaví $solution = 1$ a algoritmus se ukončí.

4. V opačném případě se vypočte nová hodnota vektorů $\alpha_{t+1}^j, j = 1, 2, ...K$, jako

$$\alpha_{t+1}^i = \begin{cases} \alpha_t^j + x & \text{pro} \quad j = k \\ \alpha_t^j - x & \text{pro} \quad j = i \\ \alpha_t^j & \text{jinak.} \end{cases}$$

5. Přejdi se na krok 2.

4.4.2 Fisherův klasifikátor pomocí Kozincova algoritmu

Řešní Fisherova klasifikátoru pomocí Kozincova algoritmu není v Knize popsáno. Proto jsme algoritmus navrhli sami, analogicky k řešení pomocí perceptronu, i když díky složitějšímu adaptačnímu pravidlu Kozincova algoritmu, není výsledek tak elegantní.

Algoritmus má naprosto stejné vstupní a výstupní argumenty jako předchozí řešení v odstavci 4.4.1.

Algoritmus 4.5 Fisherův klasifikátor pomocí Kozincova algoritmu

Vstup: $X, J = J_1 \cup J_2 \cup ... \cup J_K, K, t_{max}, t_s, \alpha_s^j$ *kde* $j = 1, 2, ...K$

Výstup: α^j *kde* $j = 1, 2, ...K, solution, t$

1. Inicializace algoritmu. Je-li zadán počáteční stav t_s, α_s^j, pak se přiřadí

$$\begin{aligned} t &= t^s, \\ \alpha_t^j &= \alpha_s^j, \quad j = 1, 2, ...K, \end{aligned}$$

   jinak

$$\begin{aligned} t &= 1, \\ \alpha_t^j &= x^j, \quad x^j \in X, j = 1, 2, ...K, \end{aligned}$$

   kde x^j je j-tý vektor z množiny X.

2. Začátek iterace. V případě, že není dosaženo maximálního počtu kroků, tj. $t < t_{max}$, pokračuje se v iteraci přechodem na krok 3. Jinak se nastaví $solution = 0$ a algoritmus se ukončí.

3. Hledá se taková trojice k, i a x, pro kterou platí

$$i = \operatorname*{argmax}_{i=1,...,K} \langle \alpha_t^i, x \rangle, \quad x \in X(k), i \neq k.$$

4. Jestliže taková trojice neexistuje, jsou vektory $\alpha_t^j, j = 1, 2, ...K$, řešením úlohy. Pak se nastaví $solution = 1$ a algoritmus se ukončí.

5. V opačném případě se vypočte nová hodnota vektorů $\alpha_{t+1}^j, j = 1, 2, ...K$, jako

$$\alpha_{t+1}^j = \begin{cases} \alpha_t^j \cdot (1-m) + m \cdot x & \text{pro} \quad j = k, \\ \alpha_t^j \cdot (1-m) - m \cdot x & \text{pro} \quad j = i, \\ \alpha_t^j \cdot (1-m) & \text{jinak.} \end{cases}$$

   kde

$$k = \operatorname*{argmin}_{m} |(1-m) \cdot \alpha' + m \cdot x'|.$$

   Vektory x' a α' se určí z x a α_t^j

$$\begin{aligned} \alpha' &= [\alpha_{t_1}, \alpha_{t_2}, ..., \alpha_{t_K}], \\ x' &= [x'_1, x'_2, ..., x'_K]. \end{aligned}$$

pro vektory x'^j platí

$$x'^j = \begin{cases} x & \text{pro} \quad j = k, \\ -x & \text{pro} \quad j = i, \\ 0 & \text{jinak.} \end{cases}$$

6. Zvětší se počítadlo kroků $t = t + 1$ a přejde se na krok 2.

4.5 Andersonova úloha a její zobecnění

Nejprve uvedeme definici zobecněné Anderosovy úlohy, jak je formulována v Knize. Následně popíšeme různé metody řešení zobecněné Andersonovy úlohy a pro porovnání také původní Andersonovo řešení. V našem programovém balíku jsou implementovány všechny algoritmy, které zde popíšeme.

Nechť je X mnoharozměrný lineární prostor. Výsledkem pozorování objektu je bod x v tomto prostoru. Nechť je k stav objektu nepřístupný pro pozorování a nechť může nabývat jen dvou hodnot $\{1, 2\}$. Nechť je známo, že rozdělení podmíněných pravděpodobností $p(x|k), x \in X, k \in K$, je mnoharozměré gaussovské rozdělení. Matematické očekávání μ_k a kovarianční matice $\sigma_k, k = 1, 2$ těchto rozdělení nejsou známa. Ví se jen, že parametry (μ_1, σ_1) patří jisté známé konečné množině parametrů $\{(\mu^j, \sigma^j)|j \in J_1\}$. Podobně také (μ_2, σ_2) jsou neznámé parametry patřící konečné množině $\{(\mu^j, \sigma^j)|j \in J_2\}$. Použili jsme jak horní, tak dolní indexy. Například μ_1 a σ_1 znamenají skutečné, ale neznámé statistické parametry objektu, který se nachází v prvním stavu. Parametry $\{\mu^j, \sigma^j\}$ při některém horním indexu j jsou jednou z možných dvojic hodnot, kterých parametr může nabýt.

Máme tudíž dvě třídy objektů, z nichž každá je popsána směsí gaussovských distribucí. Komponenty obou směsí jsou známé, pouze jejich váhy jsou neznámé. Když se má při zmíněných neúplných apriorních znalostech statistického modelu a známého pozorování x zjisti stav objektu k, je úlohu potřebné formulovat jako rozhodování s nenáhodnými zásahy (viz. Kniha, druhá kapitola). V našem případě hledáme starategii $q: X \to \{1, 2\}$, která minimalizuje hodnotu

$$\max_{j \in J_1 \cup J_2} \varepsilon(j, \mu^j, \sigma^j, q), \tag{4.16}$$

kde $\varepsilon(j, \mu^j, \sigma^j, q)$ je pravděpodobnost jevu, že gaussovský náhodný vektor x s matematickým očekáváním μ^j a kovarianční maticí σ^j splňuje buď vztah $q(x) = 1$ pro $j \in J_2$ nebo $q(x) = 2$ pro $j \in J_1$. Vztah (4.16) vyjadřuje chybu klasifikátoru se strategií q. My se budeme zabývat případem, kdy je rozhodovací strategie lineární, jak jsme ji popsali v sekci 4.1. Tato strategie rozděluje lineární prostor X na dvě části podle nadroviny $\langle \alpha, x \rangle = \theta$ a lze popsat jako

$$q(x) = \begin{cases} 1, & \text{pro} \quad \langle \alpha, x \rangle \geq \theta, \\ 2, & \text{pro} \quad \langle \alpha, x \rangle < \theta. \end{cases} \tag{4.17}$$

kde $\alpha \in X$ je vektor a θ je práh.

Úloha (4.16) vyhovující požadavku (4.17) se nazývá *zobecněná Andersonova úloha.*

Autorem úlohy, kdy jsou třídy dány pouze jedním gaussovským rozdělením, tedy $|J_1| = |J_2|$, jsou Anderson a Bahadur [1], kteří tuto úlohu poprvé formulovali a navrhli její řešení.

Nakonec ještě zkráceně formulujeme zobecněnou Andersonovu úlohu, respektive její optimální řešení a ε-řešení, abychom se na ně mohli později odvolávat. Místo označení strategie q budeme zapisovat rovnou parametry, které ji určují, tj. α a θ.

Formulace 4.7 Optimální řešení zobecněné Andersonovy úlohy.
Nechť jsou zadány konečné množiny směsí gaussovských rozdělení $\{(\mu^j, \sigma^j)|j \in J_1\}$ *a* $\{(\mu^j, \sigma^j)|j \in J_2\}$*, které popisují první a druhou třídu. Lineární rozhodovací strategie je určena nadrovinou* $\langle\alpha, x\rangle = \theta$*. Výsledek optimalizačního kritéria*

$$\{\alpha, \theta\} = \underset{\alpha,\theta}{\operatorname{argmin}} \max_{j \in J_1 \cup J_2} \varepsilon(\alpha, \theta, \mu^j, \sigma^j) \tag{4.18}$$

je optimální řešení zobecněné Andersonovy úlohy. ▲

V některých případech nepotřebujeme hledat optimální řešení, ale stačí nám zajistit, aby chyba klasifikace byla menší než jistá zadaná hodnota ε_0. Tehdy mluvíme o ε-řešení.

Formulace 4.8 ε-řešení zobecněné Andersonovy úlohy.
Nechť jsou zadány konečné množiny směsí gaussovských rozdělení $\{(\mu^j, \sigma^j)|j \in J_1\}$ *a* $\{(\mu^j, \sigma^j)|j \in J_2\}$*, které popisují první a druhou třídu. Lineární rozhodovací strategie je určena nadplochou* $\langle\alpha, x\rangle = \theta$ *a* ε_0 *je kladné reálné číslo* $\varepsilon_0 < 0,5$*. Vektor* α *a práh* θ*, splňující*

$$\max_{j \in J_1 \cup J_2} \varepsilon(\alpha, \theta, \mu^j, \sigma^j) < \varepsilon_0$$

jsou ε-řešením Andersonovy úlohy.

▲

4.5.1 Pojmy potřebné pro řešení

Analýzou řešení zobecněné Andersonovy úlohy se zabývá větší část páté kapitoly v Knize. My zde uvedeme pouze hlavní výsledky analýzy potřebné k řešení.

Ekvivalentní formulace zobecněné Andersonovy úlohy

V sekci 4.2 o lineární rozhodovací strategii jsme uvedli dvě transformace, které tuto strategii zjednodušují. Stejné trasformace použijeme i při řešení zobecněné Andersonovy úlohy.

V první trasformaci popsané fomulací 4.1 se bodům z trénovací množiny přidává konstantní souřadnice. Podle tohoto postupu se transformují střední hodnoty $\mu^j, j \in J_1 \cup J_2$, popisující směsi gaussovských rozdělení. Protože je přidaná poslední souřadnice konstantní, budou mít transformované kovarianční matice $\sigma^j, j \in J_1 \cup J_2$ poslední řádek a sloupec roven nulám.

Druhá transformace popsaná formulací4.2, která úpravou rozhodovací strategie spojuje obě trénovací množiny do jedné, zůstane nezměněna, ale trasformují se jen střední hodnoty $\mu^j, j \in J_2$.

Celá transfromace pro případ zobecněné Andersonovy úlohy se provede podle následujícího postupu.

Formulace 4.9 Trasformace zobecněné Andersonovy úlohy.

Nechť je $\{(\mu'^j, \sigma'^j)|j \in J\}$ konečnou množinou směsí gaussovských rozdělení, která vznikla transformací z $\{(\mu^j, \sigma^j)|j \in J_1 \cup J_2\}$. Tato trasformace má tvar

$$\mu'^j = \begin{cases} \{\mu_1^j, ...\mu_n^j, 1\} & pro \quad j \in J_1, \\ -\{\mu_1^j, ...\mu_n^j, 1\} & pro \quad j \in J_2, \end{cases}$$

$$\sigma'^j = \begin{bmatrix} \sigma_{11}^j & \dots & \sigma_{1n}^j & 0 \\ \vdots & & \vdots & 0 \\ \sigma_{n1}^j & \dots & \sigma_{nn}^j & 0 \\ 0 & \dots & 0 & 0 \end{bmatrix} \quad pro \ j \in J_1 \cup J_2.$$

Nechť je α' vektor v $n+1$-rozměrném lineárním prostoru. Optimalizační problém

$$\alpha' = \underset{\alpha'}{\mathrm{argmin}} \max_{j \in J} \varepsilon(\alpha', \mu'^j, \sigma'^j) \tag{4.19}$$

je kvivalentní k optimálnímu řešení zobecněné Andersonovy úlohy. ▲

Dále se budeme zabývat pouze tímto kritériem, a proto budeme jeho členy α', μ'^j a σ'^j pro přehlednost zapisovat bez čárek, tedy α, μ^j, σ^j.

Podle uvedené transfromace je zobecněná Andersonova úloha ekvivalentní minimalizaci funkce $\max_{j \in J} \varepsilon(\alpha, \mu^j, \sigma^j)$. Jedno číslo

$$\varepsilon(\alpha, \mu^j, \sigma^j) \tag{4.20}$$

vyjadřuje pravděpodobnost chybné klasifikace za předpokladu, že j-té gaussovské rozdělení dané parametry μ^j a σ^j skutečně popisuje danou třídu. Řešení se opírá o geometrickou představu, kdy každému gaussovskému rozdělení $N(\mu^j, \sigma^j)$ odpovídá množina bodů ohraničená elipsoidem $E(\mu^j, \sigma^j)$. Střed elipsoidu je dán střední hodnotou μ^j, jeho tvar kovarianční maticí σ^j. Množina bodů x ohraničená elipsoidem $E(\mu, \sigma)$ vyhovuje nerovnici $\langle (x - \mu), \sigma^{-1} \cdot (x - \mu) \rangle \leq r^2$, kde r je jeho poloměr.

V Knize je dokázáno, že chyba $\varepsilon(\alpha, \mu^j, \sigma^j)$ ostře klesá se vzdáleností středu elipsoidu od rozdělujcí nadplochy, tedy s rostoucím poloměrem elipsoidu. Proto se snažíme určit rozdělující nadplochu tak, aby poloměr toho nejmenšího elipsoidu, omezeného rozdělující nadrovinou, byl co největší (maxmin optimalizace).

Po transformaci popsané formulací 4.9 prochází rozdělující nadrovina počátkem souřadnic a všechna rozdělení jsou v jedné polorovině. Zvětšováním elipsoidů se snažíme nalézt takovou pozici, kdy bude rozdělujcí nadplocha přitisknuta elipsoidy tak, že jakákoli změna její pozice by zmenšila rozměr alespoň jednoho (toho nejbližšího) elipsoidu.

Při řešení budeme potřebovat bod dotyku elipsoidu $E(\mu^j, \sigma^j)$ s rozdělující nadplochou procházející počátkem, která je dána normálovým vektorem α. Tento bod budeme značit x_0^j a pro jeho výpočet máme vztah

$$x_0^j = \mu^j - \frac{\langle \alpha, \mu^j \rangle}{\langle \alpha, \sigma^j \cdot \alpha \rangle} \sigma^j \cdot \alpha. \tag{4.21}$$

Dalším důležitým pojmem je poloměr elipsoidu $E(\mu^j, \sigma^j)$, který je omezený rozdělující nadplochou. Tento poloměr budeme označovat $r(\alpha, \mu^j, \sigma^j)$ a lze jej vyjádřit po dosazení bodu dotyku x_0^j do rovnice pro elipsoid $E(\mu^j, \sigma^j)$, kdy dostaneme

$$r(\alpha, \mu^j, \sigma^j) = \frac{\langle \alpha, \mu^j \rangle}{\sqrt{\langle \alpha, \sigma^j \cdot \alpha \rangle}}. \tag{4.22}$$

Díky vztahu mezi chybou $\varepsilon(\alpha, \mu^j, \sigma^j)$ a poloměrem $r(\alpha, \mu^j, \sigma^j)$ můžeme přepsat kritérium zobecněné Andersonovy úlohy do tvaru vhodnějšího pro optimalizaci, a sice

$$\alpha = \operatorname*{argmax}_{\alpha} \min_{j \in J} r(\alpha, \mu^j, \sigma^j). \tag{4.23}$$

V Knize jsou analyzovány vlastnosti maximalizované funkce $\min_{j \in J} r(\alpha, \mu^j, \sigma^j)$. Mezi ty nejdůležitější patří fakt, že se jedná o unimodální funkci, která není konkávní a diferencovatelná. Z tohoto důvodu nelze pro její optimalizaci použít klasických gradientních metod.

Nakonec uvedeme větu o nutných a postačujících podmínkách řešení Andersonovy úlohy. Před tím ještě definujeme množinu indexu $J^0 \subset J$, což jsou rozdělení (jim odpovídající elipsoidy) s nejmenší vzdáleností od rozdělující nadplochy. Neboli takové indexy j, pro něž platí

$$\{j \in J^0 | r^j\} = \min_{j \in J} r(\alpha, \mu^j, \sigma^j). \tag{4.24}$$

Věta 4.1 *Nutné a postačující podmínky pro řešení zobecněné Andersonovy úlohy.*

Pokud konvexní obal množiny bodů dotyku $x_0^j, j \in J^0$, obsahuje počátek souřadnic, pak pro jakýkoli vektor α', který není kolineární s vektorem α, platí nerovnost

$$\max_{j \in J} \varepsilon(\alpha', \mu^j, \sigma^j) > \max_{j \in J} \varepsilon(\alpha, \mu^j, \sigma^j).$$

Pokud zmíněný konvexní obal neobsahuje počátek souřadnic, pak existuje vektor $\Delta\alpha$ a kladné číslo T tak, že pro jakékoliv $t, 0 < t \leq T$ je splněna nerovnost

$$\max_{j \in J} \varepsilon(\alpha + t.\Delta\alpha, \mu^j, \sigma^j) > \max_{j \in J} \varepsilon(\alpha, \mu^j, \sigma^j).$$

▲

4.5.2 Popis vstupů a výstupů

NA následujících řádcích popíšeme vstupní a výstupní argumenty společné všem algoritmům řešícím Andersonovy úlohy. Jednotlivé algoritmy mají navíc některé argumenty, které se vztahují ke konkrétnímu řešení. Tyto argumenty uvedeme příslušných odstavcích.

Vstupem algorimů je konečná množina středních hodnot $\mu^j, j \in J_1 \cup J_2$ a kovariančních matic $\sigma^j, j \in J_1 \cup J_2$, které popisují gaussovské rozdělení hustoty pravděpodobnosti. Algoritmy hledají rozdělující nadrovinu pro dvě třídy, přičemž rozdělení hustot pravděpodobností určující první třídu, jsou dána množinou indexů $j \in J_1$ a stejně pro druhou třídu to je množina indexů $j \in J_2$. Maximální počet iterací algoritmu je omezen vstupním argumentem t_max. V případě, kdy chceme algoritmus spustit z předem definovaného stavu, použijeme trojici vstupních argumentů t_s, α_s a θ_s, které jednoznačně určují stav řešení.

Výstupem algorimů je vektor α a práh θ, které určují lineární rozhodovací strategii. Je-li výstup algoritmu opravdu řešením, určuje proměnná *solution*, která nabývá tří různých hodnot.

- $solution = 0$, výstup není řešením úlohy
- $solution = 1$, výstup je řešením úlohy
- $solution = -1$, úloha nemá řešení

Číslo kroku, při kterém byl algoritmus ukončen, je vráceno v proměnné t.

4.6 Původní řešení Andersonovy úlohy

Zde popíšeme původní algoritmus, jak ho navrhli Anderson a Bahadur [1]. Původní řešení uvažuje, že obě třídy jsou popsány pouze jedním normální rozdělením. První třídu popisuje gausovské rozdělení $N(\mu_1, \sigma_1)$ a druhou $N(\mu_2, \sigma_2)$. Řešení je postaveno na faktu, že optimální rozdělující nadplocha musí být tečnou k oběma rozdělením. Takovéto umístění rozdělující nadplochy se dosahuje iterativním výpočtem. Vzdálenost od optimálního řešení je určena vztahem dvou proměnných λ a γ algoritmu (viz popis v Knize). Z tohoto důvodu je speciálním vstupním parametrem číslo Δ_{stop}, které udává postačující přesnost řešení. Přesněji řečeno, algoritmus je ukončen za podmínky, že $|\gamma - \frac{1-\lambda}{\lambda}| \leq \Delta_{stop}$. V případě, že je $\Delta_{stop} = 0$, hledá se optimální řešení.

Chceme-li spustit algoritmus z jiného než počátečního stavu, musíme zadat hodnotu λ_s, která jednoznačně určuje stav řešení.

Výstup algoritmu je stejný jako u všech ostatních (viz popis v odstavci 4.5.2), kromě (stavových) proměnných λ a γ.

Algoritmus 4.6 Původní řešení Andersonovy úlohy

Vstup: $\mu_1, \mu_2, \sigma_1, \sigma_2, t_{max}, \Delta_{stop}, t_s, \lambda_s$

Výstup: $\alpha, \theta, solution, t, \lambda, \gamma$

1. Inicializace algoritmu. Je-li zadán počáteční stav λ_s přiřaď
$$t = t_s, \lambda = \lambda_s,$$
jinak
$$t = 1, \lambda = 0,5.$$
2. Začátek iterace. Zvětši počítadlo kroku $t = t+1$. Je-li splněno $t \leq t_{max}$ pokračuj v iteraci krokem 3, jinak nastav $solution = 0$ a přejdi na poslední krok 5.
3. Urči novou hodnotu α jako
$$\alpha = ((1-\lambda) \cdot \sigma_1 + \lambda\sigma_2)^{-1}(\mu_1 - \mu_2).$$
4. Podmínka ukončení. Nejprve urči
$$\gamma = \sqrt{\frac{\langle \alpha, \sigma_2 \cdot \alpha \rangle}{\langle \alpha, \sigma_1 \cdot \alpha \rangle}}$$
a pak spočti rozdíl
$$\Delta = \left| \gamma - \frac{1-\lambda}{\lambda} \right|.$$
Je-li dosaženo $\Delta < \Delta_{stop}$, nastav $solution = 1$ a přejdi na poslední krok 5. V opačném případě spočti novou hodnotu
$$\lambda = \frac{1}{1-\gamma}$$
a iteruj znovu přechodem na krok 2.
5. Ukončení algoritmu. Dopočítej hodnotu parametru θ, jako
$$\theta = \lambda \langle \alpha, \sigma_2 . \alpha \rangle + \langle \alpha, \mu_2 \rangle.$$

4.7 Řešení zobecněné Andersonovy úlohy

4.7.1 Kostra algoritmu

Řešení zobecněné Andersonovy úlohy vyplývá z analýzy tohoto problému, kterou se zabývá velká část páté kapitoly v Knize. Výsledkem tohoto snažení je kostra algoritmu, kterou popíšeme na následujících řádcích.

Vstupy i výstupy algoritmu jsou shodné s popisem v odstavci 4.5.2, kromě dvou argumentů. Prvním z nich je kladné číslo Δ_r, které určuje minimální změnu kvality řešení $\min_{j\in J} r(\alpha, \mu^j, \sigma^j)$ ve dvou po sobě následujících krocích. Tím je definováno ukončovací kritérium úlohy pro případ, že by se algoritmus „zasekl“ v nějakém zlepšujícím směru $\Delta\alpha$. Algorimus také vrací kvalitu řešení, která byla dosažena při ukončení algoritmu, neboli hodnotu funkce $\min_{j\in J} r(\alpha, \mu^j, \sigma^j)$.

Algoritmus 4.7 Kostra algoritmu pro řešení zobecněné Andersonovy úlohy

Vstup: μ^j, σ^j *kde* $j = 1, 2, ...K, J = J_1 \cup J_2, t_{max}, \Delta_r, t_s, \alpha_s, \theta_s, r_{min}$

Výstup: $\alpha, \theta, solution, r_{min}, t$

1. Provedeme trasformaci (4.9) směsi gaussovských rozdělení $\{(\mu^j, \sigma^j)|j \in J_1 \cup J_2\}$ na $\{(\mu'^j, \sigma'^j)|j \in J\}$,

$$\mu'^j = \begin{cases} \{\mu_1^j, ...\mu_n^j, 1\} & \text{pro} \quad j \in J_1, \\ -\{\mu_1^j, ...\mu_n^j, 1\} & \text{pro} \quad j \in J_2, \end{cases}$$
$$\sigma'^j = \begin{bmatrix} \sigma_{11}^j & ... & \sigma_{1n}^j & 0 \\ \vdots & & \vdots & 0 \\ \sigma_{n1}^j & ... & \sigma_{nn}^j & 0 \\ 0 & ... & 0 & 0 \end{bmatrix} \quad \text{pro } j \in J_1 \cup J_2.$$

   V případě, že je zadán počáteční stav α_s, θ_s, pak přiřadíme

$$\alpha'_s = \{\alpha_{s_1}, ..., \alpha_{s_n}, -\theta_s\},$$

   Pro přehlednost budeme α', μ'^j *a* σ'^j *dále označovat bez čárky jako* α, μ^j *a* σ^j.

2. Inicializace algoritmu. Je-li zadán počáteční stav $\{t_s, \alpha_s\}$ přiřaď

$$t = t_s, \quad \alpha_t = \alpha_s,$$

   jinak

$$t = 1,$$
$$\{\alpha_1 | \langle \alpha_1, \mu^j \rangle > 0, j \in J\}.$$

   V případě, že není nalezeno žádné α_1 splňující podmínku (4.72), pak to znamená, že neexistuje řešení s chybou klasifikace menší než 50%. V tomto případě se nastaví $solution = -1$ a algoritmus se ukončí.

3. Začátek iterace. Je-li dosaženo maximálního kroku, tj. $t = t_{max}$, pak nastav $solution = 0$ a ukonči algoritmus přechodem na krok 7. V opačném případě pokračuj krokem 4.

4. Hledá se zlepšující vektor $\Delta\alpha$, pro který platí

$$\min_{j \in J} r(\alpha_t + k.\Delta\alpha, \mu_j, \sigma_j) > \min_{j \in J} r(\alpha_t, \mu_j, \sigma_j), \quad 0 < k, \quad k \in R. \tag{4.25}$$

Kde $r(\alpha, \mu_j, \sigma$ je nepřímo úměrné velikosti chyby pro j-té rozdělení a spočte se

$$r(\alpha, \mu_j, \sigma) = \frac{\langle \alpha_t, \mu_j \rangle}{\sqrt{\langle \alpha_t, \sigma_j \cdot \alpha_t \rangle}}. \tag{4.26}$$

Není-li nalezeno žádné $\Delta\alpha$ splňující (4.25) znamená, že vektor α_t je řešením úlohy. Pak nastav *solution* $= 1$ a přejdi na krok 7. V opačném případě pokračuj krokem 5.

5. Hledá se kladné číslo k, pro které platí

$$k = \operatorname*{argmin}_{k} \max_{j \in J} \varepsilon(\alpha_t + k \cdot \Delta\alpha, \mu_j, \sigma_j)$$

Nový vektor α_{t+1} se vypočte jako

$$\alpha_{t+1} = \alpha_t + k \cdot \Delta\alpha.$$

6. Kritérium ukončení. Je-li změna kvality řešení menší než zadaná mez Δ_r, což znamená

$$|r^t_{min} - r^{t-1}_{min}| < \Delta_r,$$

pak nastav *solution* $= 1$ a přejdi na krok 7. Jinak zvětši počítadlo kroků $t = t + 1$ a pokračuj v iteraci přechodem na krok 3.

7. Ukončení algoritmu. Proveď se inverzní transformace z kroku 1. Nalezené řešení α_t opět označíme jako α' a určíme z parametry rozdělující nadroviny v původním prostoru, jako

$$\begin{aligned} \alpha &= \{\alpha'_1, ..., \alpha'_n\} \\ \theta &= \alpha'_{n+1}. \end{aligned}$$

Algoritmus také vrací dosaženou kvalitu řešení $r_{min} = r^t_{min}$.

Uvedený algorimus se kládá z podúloh, které je třeba vyřešit. Na použitých metodách a jejich kombinacích samozřejmě závisí výsledné vlastnosti celého algoritmu. V Knize je uvedeno několik možných naplnění této kostry, ale není zde uvedené žádné striktní řešení. Takže jsme měli při implementaci poměrně velkou volnost. Čehož jsme využili a při naplňování kostry algoritmu vyzkoušeli několik variant, včetně několika pokusů o vylepšení metod popsaných v Knize.

První podúloha je v kroku 2 algoritmu, kdy se hledá takový vektor α, aby všechny skalární součiny $\langle \alpha, \mu^j \rangle, j \in J$ byly kladné. Pro tuto úlohu lze použít jakýkoliv algoritmy z odstavce 4.3. Nám se nejvíce osvědčil algoritmus Linear Support Vetor Machine, především proto, že jeho implementace je díky optimalizačnímu toolboxu velmi rychlá. Navíc u něj nejsou problémy s ukončením v konečném čase (viz výše).

Další podúlohou je určování zlepšujícího směru $\Delta\alpha$ v kroku 4 algoritmu. Na určení tohoto vektoru velice závisí konvergence algoritmu k optimálnímu řešení. Při jeho špatném určování se může algoritmus lehce "zaseknout" v nějakém směru.

Posledním problémem je optimalizační úloha v kroku 5 algoritmu. Zde se hledá maximum složité funkce jedné reálné proměnné. Tento problém je třeba vyřešit numerickou optimalizací.

V dalších odstavcích popíšeme metody, které jsme vyzkoušeli a jenž jsou všechny implementovány v našem programovém balíku.

4.7.2 Ukončení algoritmu

Činnost algoritmu se může ukončit třemi možnými způsoby. Zaprvé je to možné hned na začátku, při určování počátečního řešení v kroku 2. V tomto případě nemá cenu hledat další řešení, neboť chyba klasifikace nebude menší než 50%. Což je totožné s klasifikací podle náhodné rozhodovací strategie.

Druhou možností ukončení je případ, kdy se v kroku 4 algoritmu nenalezne žádný vektor $\Delta\alpha$, v jehož směru se snižuje chyba řešení úlohy. Tento případ je žádoucí, neboť to znamená že stávající řešení úlohy je optimální. Respektive optimální s přesností danou metodou, která zlepšující směr $\Delta\alpha$ hledá.

Poslední možností, kdy může algorimus skončit, je krok 6 algoritmu. Zde se zjišťuje, zda-li se kvalita řešení za poslední dva kroky zvýšila o požadovanou minimální mez. Jestliže tomu tak není, je algoritmus ukončen neboť se pravděpodobně „zasekl“ v nějakém zlepšujícím směru $\Delta\alpha$. V ideálním případě by tento stav neměl nastat, ale protože je nutné pro optimalizaci kritéria použít numerickou metodu, je i tento stav možný. Jestli že se tak stane, nemáme žádnou informaci o tom, jak daleko jsme od optimálního řešení úlohy. Uvedený stav je horší než druhá možnost ukončení, a proto se mu budeme snažit vyhnout vhodnou volbou zlepšujícího směru $\Delta\alpha$.

4.7.3 Hledání zlepšujícího směru $\Delta\alpha$

Pro vektor zlepšující vektor $\Delta\alpha$ musí platit, že se chyba řešení ε v tomto směru snižuje, neboli

$$\min_{j\in J} r(\alpha_t + k.\Delta\alpha, \mu_j, \sigma_j) > \min_{j\in J} r(\alpha_t, \mu_j, \sigma_j), \quad 0 < k, \quad k \in R. \tag{4.27}$$

V Knize je dokázáno, že pro směr $\Delta\alpha$, splňující uvedenou pdmínku (4.27), musí platit

$$\langle \Delta\alpha, x_0^j \rangle > 0, j \in J^0. \tag{4.28}$$

Kde J^0 je množina rozdělení s nejmenší vzdáleností (největší chybou) od rozdělující nadroviny

$$\{j \in J^0 | j = \operatorname*{argmin}_{j\in J} r(\alpha, \mu_j, \sigma_j)\}.$$

Metoda A - Určování zlepšujícího směru přímo z definice

Nejprve se vypočte množina bodů dotyku $\{x_0^j, j \in J^0\}$. Zlepšující směr $\Delta\alpha$ se hledá pomocí algoritmu pro jednoduché rozdělení konečných množin bodů tak, aby platilo $\langle \Delta\alpha, x_0^j \rangle > 0, j \in J^0$. Pro tento úkol lze použít jakýkoli algoritmus z odstavce (4.3). I když je tento způsob velmi jednoduchý, dával při našich experimentech nejhorší výsledky.

Metoda B - Snižování chyby nejhoršího rozdělení

Vektor $\Delta\alpha$ se hledá tak, aby se v něm nejrychleji zmenšovala chyba pro rozdělení s největší chybou. To je takový směr, v němž je záporně braná derivace chybové funkce $\varepsilon(\alpha, \mu_j, \sigma_j), j \in J^0$, největší, tj.

$$\Delta\alpha = \operatorname*{argmax}_{\Delta\alpha} \min_{j} \left(- \frac{\partial \varepsilon \left(\alpha + k.\frac{\Delta\alpha}{|\Delta\alpha|}, \mu^j, \sigma^j \right)}{\partial k} \right)$$

nebo, což je totéž,

$$\Delta\alpha = \operatorname*{argmax}_{\{\Delta\alpha||\delta\alpha|=1\}} \min_j \left\langle \Delta\alpha, \frac{x_0^j}{\sqrt{\langle\alpha, \sigma^j.\alpha\rangle}} \right\rangle.$$

Označíme-li $\frac{x_0^j}{\sqrt{\langle\alpha,\sigma^j.\alpha\rangle}}$ symbolem y^j, obdržíme

$$\Delta\alpha = \operatorname*{argmax}_{\Delta\alpha} \min_j \frac{\langle\Delta\alpha, y^j\rangle}{|\Delta\alpha|},$$

což je ekvivalentní úloze optimálního rozdělení množiny bodů y^j nadrovinou. My jsme pro tuto úlohu opět použili algoritmus Linear Support Vector Machine.

Metoda C - Aproximace gaussovským rozdělením s jednotkovou kovarianční matici v bodech dotyku

Vektor $\Delta\alpha$ se hledá tak, aby splňoval

$$\Delta\alpha = \max_{\Delta\alpha} \min_{j\in J} \frac{\langle\Delta\alpha, x_0^j\rangle}{|\Delta\alpha|}, \tag{4.29}$$

kde x_0^j jsou body vypočítané podle

$$x_0^j = \mu^j - \frac{r_{min}}{\sqrt{\alpha, \sigma^j \cdot \alpha}} \sigma^j \cdot \mu^j.$$

Číslo r_{min} je poloměr toho nejmenšího elipsoidu (rozdělení), neboli platí

$$r_{min} = \min_{j\in J} r(\alpha, \mu^j, \sigma^j).$$

Poloměr elipsoidu $r(\alpha, \mu^j, \sigma^j)$ se vypočítá podle vztahu (4.26). Takto nalezený vektor $\Delta\alpha$ jednak splňuje nutnou podmínku (4.27), protože $J^0 \subseteq J$. Navíc ve směru $\Delta\alpha$ nebude chyba žádného rozdělení menší než ta v minulém kroku. Pokud budeme všechny rozdělení brát jako rozrůstající se elipsoidy, jejichž současná velikost je dána poloměrem r_{min}, pak jsou takto určené body x_0^j nejblíže rozdělující nadrovině.

Způsob hledání $\Delta\alpha$ podle (4.29) nejprve aproximuje všechna rozdělení $N(\mu^j, \sigma^j)$ v bodě $x_0^j(\alpha, \mu^j, \sigma^j)$ normálním rozdělením $N(\mu^j, E)$, které mají jednotkovou kovarianční matici. Pro tuto aproximaci se pak hledá optimální směr $\Delta\alpha$. Určení optimální $\Delta\alpha$ je v tomto případě ekvivalentní s úlohou optimálního rozdělení konečných množin bodů. Poznamenejme, že speciální případ Andersonovy úlohy, kdy jsou všechny kovarianční matice jednotkové, je totožný právě s optimálního rozdělení a bude konvergovat k řešení v jednom kroku.

Pro hledání směru $\Delta\alpha$ podle (4.29) lze použít jakýkoli algoritmus pro optimální rozdělení konečných množin bodů. V naší implementaci jsme použili algoritmus Linear Support Vector Machine.

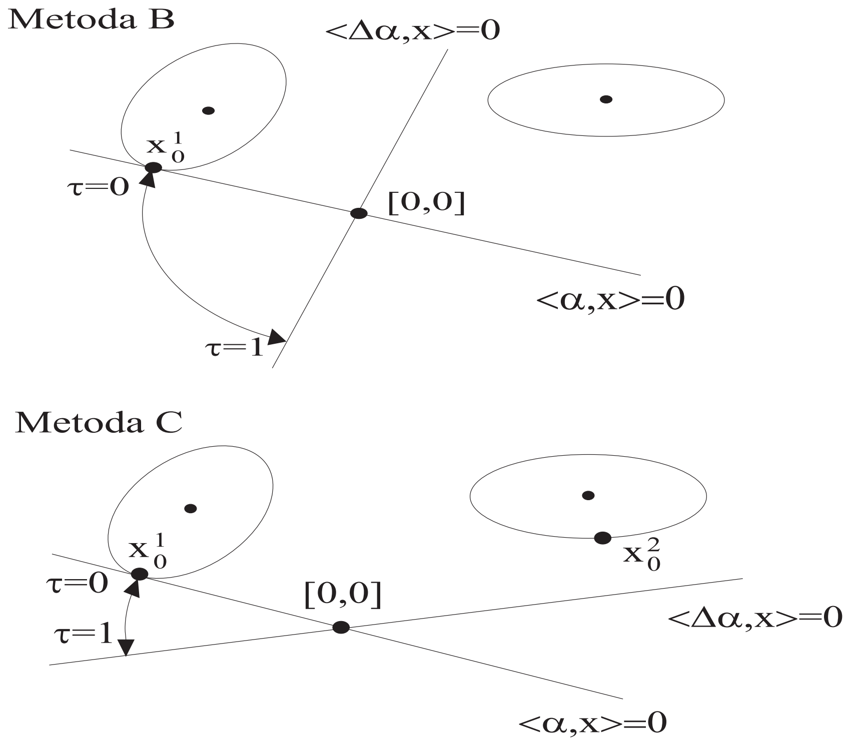
Porovnání způsobů hledání zlepšujícího směru $\Delta\alpha$

Metoda A z odstavce 4.7.3 využívá přímo definice pro zlepšující směr $\Delta\alpha$. Ačkoli je implementace tohoto způsobu velmi jednoduchá, výsledky dosažené jejím použitím byly nejhorší ze všech tří možností.

Druhý způsob, Metoda B 4.7.3, nejprve nalezne rozdělení s největší chybou a to se snaží novým směrem $\Delta\alpha$ maximálně vylepšit. Na ostatní rozdělení se nebere zřetel. Použití tohoto způsobu při některých experimentech vedlo k zaseknutí algoritmu v neoptimálním řešení.

Uvedený nedostatek jsme se nažili odstranit návrhem Metody C 4.7.3, která se snaží vylepšovat všechny rozdělení současně, a tedy používá v každém kroku informaci o všech rozděleních.

Obrázek 4.2 ilustruje rozdíl mezi vyhledáváním směru $\Delta\alpha$ podle Metody B a C. Uvažujeme dvě rozdělení a dvourozměrný prostor příznaků.



Obrázek 4.2 Porovnání metody B a C hledající zlepšující směr $\Delta\alpha$.

4.7.4 Optimalizace kritéria jedné reálné proměnné

Po nalezení zlepšujícího směru $\Delta\alpha$ se prochází úsečka mezi body danými současným řešením α a právě zlepšující směrem $\Delta\alpha$. Nalezení optimálního řešení na této úsečce se provádí numerickou optimalizací, kterou popíšeme v následujícím odstavci.

Optimalizované kritérium má tvar

$$k = \operatorname*{argmin}_{k} \max_{j \in J} \varepsilon(\alpha + k.\Delta\alpha, \mu^j, \sigma^j), \tag{4.30}$$

kde k je kladné reálné číslo. Vztah $\max_{j\in J} \varepsilon(\alpha + k \cdot \Delta\alpha, \mu^j, \sigma^j)$ vyjadřuje kvalitu řešení Andersonovy úlohy, která je dána vektorem $\alpha + k \cdot \Delta\alpha$. Jak jsme již uvedli výše, je kritérium (4.30) ekvivalentní s optimalizační úlohou

$$k = \operatorname*{argmax}_{0<k<\infty} \min_{j \in J} r(\alpha + k \cdot \Delta\alpha, \mu^j, \sigma^j), \tag{4.31}$$

kde vzdálenost

$$r(\alpha + k \cdot \Delta\alpha, \mu^j, \sigma^j) = \frac{\langle(\alpha + k \cdot \Delta\alpha), \mu^j\rangle}{\sqrt{\langle(\alpha + k \cdot \Delta\alpha), \sigma^j \cdot (\alpha + k \cdot \Delta\alpha)\rangle}}.$$

Pro výpočet vzdálenosti r dále platí, že závisí pouze na směru α a ne na jeho velikosti, platí tedy

$$r(\alpha, \mu, \sigma) = r(c \cdot \alpha, \mu, \sigma),$$

kde c je libovolné kladné číslo. Proto můžeme argument vzdálenosti r, kterým je $\alpha + k \cdot \Delta\alpha$, přepsat na

$$\frac{1}{1+k} \cdot \alpha + \frac{1}{1+k} k \cdot \Delta\alpha, \quad 0 < k < \infty.$$

Toto vyjádření lze ekvivalentně zapsat jako

$$\alpha \cdot (1 - \tau) + \tau \cdot \Delta\alpha, \quad 0 < \tau \leq 1.$$

Tímto způsobem jsme převedli minimalizaci na nekonečném intervalu na interval neurčitosti, který je konečný. Dále budeme původní optimalizační problém (4.31) označovat jako

$$\tau = \operatorname*{argmin}_{0<\tau\leq 1} f(\tau).$$

kde funkce $f(\tau)$ je rovna

$$f(\tau) = \min_{j \in J} \frac{\langle(\alpha \cdot (1-\tau) + \tau \cdot \Delta\alpha), \mu^j\rangle}{\sqrt{\langle\alpha \cdot (1-\tau) + \tau \cdot \Delta\alpha, \sigma^j \cdot (\alpha \cdot (1-\tau) + \tau.\Delta\alpha)\rangle}}. \tag{4.32}$$

Připomeňme, že funkce $f(\tau)$ je unimodální, není diferencovatelná a dále není konkávní. Na obrázku (4.3) jsou vidět jednotlivé složky funkce $f(\tau)$, pro případ pěti normálních rozdělení.

Maximalizace funkce diskretizací nezávislé proměnné

Tento algoritmus hledá hodnotu proměnné τ, při které dosahuje funkce $f(\tau)$ svého maxima. Proměnná τ je určena tak, že její odchylka od skutečné hodnoty τ^* není větší než zadaná přesnost, tj. $|\tau^* - \tau| \leq \varepsilon_\tau$.

Při hledání maxima se původní interval dělí v určitém poměru d na dva intervaly $\langle\tau_{beq}, \tau_{mid}\rangle$ a $\langle\tau_{mid}, \tau_{end}\rangle$. V každém kroku algoritmu máme trojici $(\tau^t_{beg}, \tau^t_{mid}, \tau^t_{end})$, pro kterou platí

$$f(\tau^t_{beg}) \leq f(\tau^t_{mid}) \geq f(\tau^t_{end}). \tag{4.33}$$

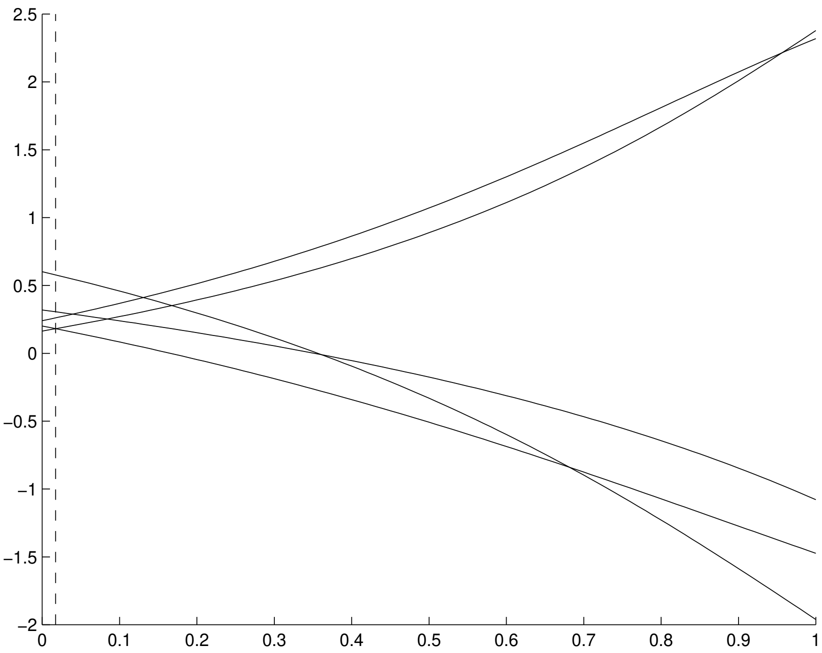
Výpočet nové trojice se provede následujícím způsobem:
Rozdělíme větší z intervalů $(\tau^t_{med} - \tau^t_{beg}), (\tau^t_{end} - \tau^t_{mid})$ na dva v předem daném poměru d. Tím se nám určení nových intervalů rozpadne na dva případy.

1. Je-li $(\tau^t_{mid} - \tau^t_{beg}) > (\tau^t_{end} - \tau^t_{mid})$, spočítáme $\tau = \tau^t_{beg} + d \cdot (\tau^t_{mid} - \tau^t_{beg})$ a ohodnotíme výraz $f(\tau^t_{mid}) > f(\tau)$. V případě, že je splněn, přiřadíme

   $$\tau^{t+1}_{beg} = \tau$$

   jinak

   $$\tau^{t+1}_{end} = \tau^t_{mid} \quad \text{a} \quad \tau^{t+1}_{mid} = \tau.$$



Obrázek 4.3 Příklad funkce $f(\tau) = \min_{j \in J} r$ pro $|J| = 5$. Nalezené maximum je označeno čárkovaně.

2. Je-li $(\tau_{mid}^t - \tau_{beg}^t) \leq (\tau_{end}^t - \tau_{mid}^t)$, spočítáme $\tau = \tau_{mid}^t + d \cdot (\tau_{end}^t - \tau_{mid}^t)$ a ohodnotíme výraz $f(\tau_{mid}^t) > f(\tau)$. V případě, že je splněn, přiřadíme

$$\tau_{edn}^{t+1} = \tau,$$

jinak

$$\tau_{beg}^{t+1} = \tau_{mid}^t \quad \text{a} \quad \tau_{mid}^{t+1} = \tau.$$

Poměr dělení intervalu d v kroku t je dán hodnotou odpovídajícího členu Fibonacciho posloupnosti. Neboli $d(t) = F(t)$, kde F(t) je právě t-tý člen této posloupnosti. Uvedený způsob je rychlejší než metoda bisekce, kdy je poměr dělení roven konstantě $d(t) = \frac{1}{2}$. Fibonacciho posloupnost je definována rekurentním zápisem

$$\begin{aligned} F(1) &= 1, \\ F(2) &= 2, \\ F(t) &= F(t-1) + F(t-2). \end{aligned}$$

Protože je funkce $f(\tau)$ unimodální, bude podmínka (4.33) po aplikaci uvedených kroků stále splněna. Tento postup se opakuje tak dlouho, až je dosaženo požadované přesnosti určení polohy minima, tj. $(\tau_{end} - \tau_{beg}) < \Delta\tau$. Dále tento postup algoritmicky zapíšeme.

Algoritmus 4.8 Maximalizace funkce diskretizací nezávislé proměnné

Vstup: μ^j, σ^j pro $j \in J, \alpha, \Delta\alpha, \varepsilon_\tau$

Výstup: α

1. Výpočet konstant. Pro všechna rozdělení $j \in J$ se vypočítá

$$\begin{aligned}
s^j &= \langle \alpha, \mu^j \rangle, \\
s^{\text{`}j} &= \langle \Delta\alpha, \mu^j \rangle, \\
\Delta s^j &= s^{\text{`}j} - s^j, \\
\sigma^j_\alpha &= \langle \alpha, \sigma^j \cdot \Delta\alpha \rangle, \\
\sigma^j_\Delta &= \langle \Delta\alpha, \sigma^j \cdot \Delta\alpha \rangle, \\
\sigma^j_{\alpha\Delta} &= \langle \Delta\alpha, \sigma^j \cdot \alpha \rangle.
\end{aligned}$$

2. Inicializace algoritmu.

$$\begin{aligned}
F_1 &= 1, F_2 = 1, \\
\tau_{beg} &= 0, \tau_{mid} = 0.5, \tau_{end} = 1, \\
f_{beg} &= \min_{j \in J} f(\tau_{beg}), \\
f_{mid} &= \min_{j \in J} f(\tau_{mid}), \\
f_{end} &= \min_{j \in J} f(\tau_{end}).
\end{aligned}$$

   Pro výpočet funkce $f(\tau)$ se používá optimalizovaný vzorec

$$f(\tau) = \frac{s^j + \tau \cdot \Delta s^j}{\sqrt{(1-\tau)^2 \cdot \sigma^j_\alpha + 2\tau \cdot (1-\tau) \cdot \sigma^j_{\alpha\Delta} + \tau^2 \cdot \sigma^j_\Delta}}.$$

3. Začátek iterace. Opraví se člen Fibonacciho posloupnosti $F = F_1 + F_2$. Dále se určí, který z intervalů $(\tau_{mid} - \tau_{beg})$ a $(\tau_{end} - \tau_{mid})$ je větší.

4. Je-li $(\tau_{mid} - \tau_{beg}) < (\tau_{end} - \tau_{mid})$, pak se spočítá

$$\tau = \tau_{beg} + \frac{F_1}{F}(\tau_{mid} - \tau_{beg})$$

   a určí se hodnota funkce $f_\tau = f(\tau)$. V případě, že $f_{mid} > f_\tau$, přiřadí se

$$\tau_{beg} = \tau, f_{beg} = f_\tau,$$

   jinak

$$\begin{aligned}
\tau_{end} &= \tau_{mid}, \quad f_{end} = f_{mid}, \\
\tau_{mid} &= \tau, \quad\quad f_{mid} = f_\tau.
\end{aligned}$$

5. Je-li $(\tau_{mid} - \tau_{beg}) \geq (\tau_{end} - \tau_{mid})$, pak se spočítá

$$\tau = \tau_{mid} + \frac{F_1}{F}(\tau_{end} - \tau_{mid})$$

   a určí se hodnota funkce $f_\tau = f(\tau)$. V případě, že $f_{mid} > f_\tau$, se přiřadí

$$\tau_{end} = \tau, f_{end} = f_\tau,$$

   jinak

$$\begin{aligned}
\tau_{beg} &= \tau_{mid}, \quad f_{beg} = f_{mid}, \\
\tau_{mid} &= \tau, \quad\quad f_{mid} = f_\tau
\end{aligned}$$

6. Kritéruim ukončení. Není-li ještě dosaženo požadované přesnosti, tj. $(\tau_{end} - \tau_{beg}) > \varepsilon_\tau$, pak se opraví členy Fibonacciho posloupnosti $F_2 = F_1, F_1 = F$ a pokračuje se přechodem na krok 3.

7. Ukončení algoritmu. Vybere se minimální τ, pro které platí

$$\tau = \underset{\tau}{\operatorname{argmin}}(f(\tau_{beg}), f(\tau_{med}), f(\tau_{end}))$$

   a určí se optimální hodnota vektoru α podle vztahu

$$\alpha = \alpha \cdot (1-\tau) + \tau \cdot \Delta\alpha.$$

Maximalizace funkce diskretizací funkční hodnoty

Tento algoritmus hledá hodnotu proměnné τ, při které dosahuje funkce $f(\tau)$ svého maxima. Proměnná τ je určena tak, že odchylka nalezeného maxima $f(\tau)$ od skutečného maxima $f(\tau^*)$ není větší než zadaná přesnost, tj. $|f(\tau) - f(\tau^*)| \leq \varepsilon_f$.

Hodnotu nezávislé proměnné budeme hledat na intervalu $T = \langle 0, 1\rangle$. Na začátku algoritmu potřebujeme horní mez f_{up} a dolní mez f_{down}, které mají následjící vlastnosti: existuje $\tau \in T$, při kterém $f(\tau) \geq f_{down}$, a neexistuje hodnota $\tau \in T$, při kterém $f(\tau) \geq f_{up}$. Algoritmus zjišťuje, zdali existuje hodnota $\tau \in T$, pro kterou platí $f(\tau) \geq f_{mid} = \frac{1}{2}(f_{down} + f_{up})$. Pokud existuje, zvětší se dolní mez na $f_{down} = f_{mid}$, neexistuje-li, sníží se horní mez na $f_{up} = f_{mid}$. Tento postup se opakuje tak dlouho, dokud není dosaženo požadované přesnosti, tedy dokud neplatí $f_{up} - f_{dow} \leq \varepsilon_f$.

Při minimalizaci potřebujeme vyhodnocovat výraz $f(\tau) \geq c$ a je-li splněn, musí se dále určit interval $\tau \in T$ jeho platnosti. V našem případě se jedná o analýzu funkce (4.32), tedy o ohodnocení výrazu

$$min_{j\in J} \frac{\langle (\alpha \cdot (1-\tau) + \tau.\Delta\alpha), \mu^j\rangle}{\sqrt{\langle \alpha \cdot (1-\tau) + \tau.\Delta\alpha, \sigma^j \cdot (\alpha \cdot (1-\tau) + \tau \cdot \Delta\alpha)\rangle}} \geq c. \tag{4.34}$$

Posledním problémem, který zbývá vyřešit, je otázka jak volit počáteční hodnoty mezí f_{down} a f_{up}. V Knize není tento problém explicitně vyřešen. My jsme zvolili jedno řešení vyplývající přímo z vlastností Andersonovy úlohy.

Dolní mez f_{down} se může určit jako $f_{down} = \min_{j\in J} r(\alpha, \mu^j, \sigma^j)$. Kdyby totiž neexistovalo τ splňující $\min_{j\in J} r(\alpha \cdot (1-\tau) + \tau \cdot \Delta\alpha, \mu^j, \sigma^j) \geq \min_{j\in J} r(\alpha, \mu^j, \sigma^j)$, nebyl by vůbec nalezen zlepšující směr $\Delta\alpha$ a optimalizaci by nemělo smysl provádět.

Na druhou stranu musí platit $\min_{j\in J} r(\alpha.(1-\tau)+\tau.\Delta\alpha, \mu^j, \sigma^j) \leq \max_{j\in J} r(\alpha, \mu^j, \sigma^j)$, což například vyplývá z geometrického vyjádření Andersonovy úlohy. Horní mez tedy určíme jako $f_{up} = \max_{j\in J} r(\alpha, \mu^j, \sigma^j)$.

Pro zvýšení přehlednosti jsme výsledek zapsali do dvou algoritmů.

Algoritmus 4.9 Maximalizace funkce diskretizací funkční hodnoty

V algoritmu se hledá interval T, pro který platí $\tau \in T, f(\tau) \geq f_{down}$. Tento výpočet řeší algoritmus popsaný dále, úlohu lze zapsat jako

$$T = fce(c, \{s^j, s'^j, \Delta s^j, \sigma^j_\alpha, \sigma^j_\Delta, \sigma^j_{\alpha\Delta} |\ pro\ j \in J\}).$$

Dále v textu již nebudeme tento zápis opakovat.

Vstup: μ_j, σ_j pro $j \in J, \alpha, \Delta\alpha, \varepsilon_f$

Výstup: α

1. Výpočet konstant. Pro všechna rozdělení $j \in J$ se vypočítá

$$\begin{aligned} s^j &= \langle \alpha, \mu^j\rangle, \\ s'^j &= \langle \Delta\alpha, \mu^j\rangle, \\ \Delta s^j &= s'^j - s^j, \\ \sigma^j_\alpha &= \langle \alpha, \sigma^j \cdot \Delta\alpha\rangle, \\ \sigma^j_\Delta &= \langle \Delta\alpha, \sigma^j \cdot \Delta\alpha\rangle, \\ \sigma^j_{\alpha\Delta} &= \langle \Delta\alpha, \sigma^j \cdot \alpha\rangle. \end{aligned}$$

2. Inicializace algoritmu. Určíme hormí a dolní mez

$$\begin{array}{rcl} f_{down} & = & \min_{j\in J} r(\alpha,\mu^j,\sigma^j), \\ f_{up} & = & \max_{j\in J} r(\alpha,\mu^j,\sigma^j), \end{array}$$

kde

$$r(\alpha,\mu^j,\sigma^j) = \frac{\langle\alpha,\mu^j\rangle}{\sqrt{\langle\alpha,\sigma^j.\alpha\rangle}}.$$

Dále se určí první interval T, pro který platí $\tau \in T, f(\tau) \geq f_{down}$.

3. Začátek iterace. Je-li dosaženo požadované přesnosti, tj. $(f_{up} - f_{down}) < \varepsilon_f$, pak se přejde na poslední krok 5, v opačném případě se pokračuje dalším krokem.

4. Vypočte se hodnota $f_{mid} = \frac{1}{2}(f_{down} + f_{up})$ a dále se určí interval T_{mid}, pro který platí $\tau \in T_{mid}, f(\tau) \geq f_{down}$. Je-li interval T_{mid} prázdný, přířadí se

$$f_{up} = f_{mid},$$

v opačném případě

$$\begin{array}{rcl} f_{down} & = & f_{mid}, \\ T & = & T_{mid} \end{array}$$

a pokračuje se v iteracích přechodem na krok 3.

5. Ukončení algoritmu. Pro $\tau \in T$, platí, že $|f(\tau) - f(\tau^*)| < \varepsilon_f$. Konkrétní hodnotu τ zvolíme např. v polovině intervalu T, tj. $\tau = \frac{1}{2}(T_{lower} + T_{upper})$. Nakonec se vypočte nová hodnota vektoru α, podle vztahu

$$\alpha = \alpha \cdot (1-\tau) + \tau \cdot \Delta\alpha.$$

Popsaná algoritmus potřebuje pro svoji činnost určování intervalu platnosti pro vztah (4.34). Tento problém řeší následující algoritmus.

Algoritmus 4.10 Určení intervalu $\tau \in T$, ve kterém je splněna nerovnost $f(\tau) \geq c$

Vstup: c a konstanty $s^j, s^{\backprime j}, \Delta s^j, \sigma^j_\alpha, \sigma^j_\Delta, \sigma^j_{\alpha\Delta}$, pro $j \in J$

Výstup: interval T

1. Určí se interval $\tau \in T_1$, pro který platí

$$0 \leq \tau \leq \min_{j\in J^0} \frac{-s^j}{\Delta s^j},$$

kde J^0 je množina indexů j, pro které platí $\Delta s^j < 0$.

2. Určí se interval $\tau \in T_2$, ve kterém leží τ, splňující soustavu nerovnic

$$(s^j + \tau\cdot\Delta s^j)^2 \geq c^2\left((1-\tau)^2\cdot\sigma^j_\alpha + 2\tau\cdot(1-\tau)\cdot\sigma^j_{\alpha\Delta} + \tau^2\cdot\sigma^j_\Delta\right), \quad j \in J.$$

Ohodnocení uvedeného výrazu lze přepsat na ekvivalentní ohodnocení soustavy kvadratických nerovnic

$$A^j\cdot\tau^2 + B^j\cdot\tau + C^j \geq 0, \quad j \in J,$$

kde

$$\begin{array}{rcl} A^j & = & (\Delta s^j)^2 - \sigma^j_\alpha\cdot c^2 + 2\sigma^j_{\alpha\Delta}\cdot c^2 - \sigma^j_\Delta\cdot c^2, \\ B^j & = & 2s^j\cdot\Delta s^j + 2\sigma^j_\alpha\cdot c^2 - 2\sigma^j_{\alpha\Delta}\cdot c^2, \\ C^j & = & (s^j)^2 - \sigma^j_\alpha\cdot c^2. \end{array}$$

3. Vypočte se průnik intervalů $T = \langle 0,1\rangle \cup T_1 \cup T_2$.

4.7.5 Výsledný algoritmus

Na základě provedených experimentů jsme určili kombinaci výše uvedených metod, která dávala nejlepší výsledky.

Pro hledání zlepšujícího směru se nejvíce osvědčil způsob popsaný v odstavci 4.7.3, tj. Metoda C - Aproximace gaussovským rozdělením s jednotkovou kovarianční maticí v bodech dotyku. K návrhu tohoto přístupu nás vedla snaha, vyhnout se „zaseknutí“ algoritmu v jednom zlepšujícím směru, jak se nám stávalo při použití Metody B z odstavce 4.7.3, který je popsan v Knize.

Jako nejlepší optimalizační metoda, řešící kritérium (4.31), se ukázal algoritmus 4.8, tj. Maximalizace funkce diskretizací nezávislé proměnné. Dostatečná hloubka dělení diskretizovaného intervalu (počet iterací) odpovídala stovkám iterací. V kombinaci s Metodou C, takto sestavený algoritmus řešil zobeceněnou Andersonovu metodu nejlépe ze všech vyzkoušených.

Samozřejmě jsme také implementovali algoritmus podle doporučení v Knize. V něm je pro hledání zlepšujícího směru použita Metoda B 4.7.3, jinak se od předchozího algoritmu neliší.

4.8 ε-řešení Andersonovy úlohy

Popisovaný algoritmus hledá ε-řešení zobecněné Andersonovy úlohy, jak je definováno v (4.8). Neboli takou strategii rozhodování, při které je maximální chyba klasifikace menší než zadaná mez ε_0. Tato chyba je úměrná poloměru nejmenšího elipsoidu, jenž odpovídá gaussovskému rozdělení s největší chybou. Uvedenou myšlenku přesně definuje věta o ε-řešení Andersonovy úlohy, která je v Knize dokázána.

Věta 4.2 *O ε-řešení zobecněné Andersonovy úlohy. Nechť pro každé kladné $\varepsilon < 0,5$ je číslo r řešením rovnice*

$$\varepsilon = \int_r^\infty \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} dx. \tag{4.35}$$

Vektor α a číslo θ vyhovují požadavku

$$\max_j \varepsilon(\alpha, \theta, \mu^j, \sigma^j) < \varepsilon_0$$

právě tehdy, když α a θ splňují nekonečnou soustavu lineárních nerovnic

$$\left.\begin{array}{ll} \langle \alpha, x \rangle > \theta, & x \in X_1(r) \\ \langle \alpha, x \rangle < \theta, & x \in X_2(r) \end{array}\right\}.$$

▲

Jinými slovy, pravděpodobnost chybné klasifikace pro jedno rozdělení je dána poloměrem elipsoidu r, který je definován jako nekonečná množina bodů x, vyhovující nerovnici

$$\langle (\mu - x), \sigma^{-1} \cdot (\mu - x) \rangle \leq r^2.$$

Nekonečné množiny $X_1(r)$ a $X_2(r)$, které popisují jednotlivé třídy, jsou určeny průnikem elipsoidů $E(r, \mu^j, \sigma^j)$ odpovídající daným rozdělením, neboli

$$X_1(r) = \bigcup_{j \in J_1} E(r, \mu^j, \sigma^j) \quad \text{a} \quad X_2(r) = \bigcup_{j \in J_2} E(r, \mu^j, \sigma^j).$$

Věta 4.2 převádí ε-řešení Andersonovy úlohy na rozdělení nekonečných množin bodů.

Iterační Kozincův algoritmus (stejně i perceptron), který řeší úlohu rozdělení konečných množin bodů (4.3), má velmi dobrou vlastnost. A sice fakt, že může po malé úpravě řešit rozdělení množin, které jsou nějak konstruktivně popsány (např. kružnice, čtverec, elipsa, atd.). Uvedenou podmínku přesně splňují množiny $X_1(r)$ a $X_2(r)$, a proto můžeme použít modifikovaný Kozincův algoritmus pro hledání ε-řešení zobecněné Andersonovy úlohy.

Zmíněná úprava algorimu spočívá ve změně jedné podmínky, při které se kontroluje, zda jsou všechny body správně klasifikovány. Tato podmínka se zamění za krok, kdy se kontroluje, zda rozdělující nadrovina od sebe odděluje množiny $X_1(r)$ a $X_2(r)$, respektive jejich nejvzdálenější body vzhledem k rozdělující nadrovině. Podrobný popis a odvození je v Knize.

Rovnice (4.35) určuje vztah mezi minimální velikostí elipsoidu a tomu odpovídající chybou. Pro určení daného poloměru je třeba vyřešit integrální rovnici. Matlab umožňuje řešení uvedeného problému po spojení s programem Maple V, pomocí Symbolic Toolboxu. Této možnosti jsme ve stávající verzi našeho programového balíku zatím nevyužili, takže se chyba řešení zadává přímo pomocí hodnoty čísla r.

Stav řešení algoritmu je jednoznačně vyjádřen proměnými α_1 a α_2. Z tohoto důvodu jsou tyto proměné přidány jako speciální vstupní a výstupní argumenty. Ostatní argumenty jsou shodné s popisem v odstavci 4.5.2.

Algoritmus 4.11 ε-**řešení Andersonovy úlohy**

Vstup: μ^j, σ^j *kde* $j = 1, 2, ...K, J = J_1 \cup J_2, t_{max}, r, t^s, \alpha_1^s, \alpha_2^s$

Výstup: $\alpha, \theta, solution, t, \alpha_1, \alpha_2$

1. Inicializace algoritmu. Je-li zadán počáteční stav $\{t^s, \alpha_1^s, \alpha_2^s\}$, přiřaď

$$\begin{aligned} t &= t^s, \\ \alpha_1^t &= \alpha_1^s, \\ \alpha_2^t &= \alpha_2^s, \end{aligned}$$

   jinak

$$\begin{aligned} t &= 1, \\ \alpha_1^1 &= \mu_1, \\ \alpha_2^1 &= \mu_2. \end{aligned}$$

   Kde μ_1 je první vektor μ_j z množiny J_1 a μ_2 je první vektor z množiny $j \in J_2$.

2. Začátek iterace. V případě, že není dosaženo maximálního počtu kroků, tj. $t < t_{max}$, pak se pokračuje v iteraci přechodem na krok 3. Jinak se nastaví $solution = 0$ a se algoritmus ukončí.

3. Vypočte se nová hodnota řešení

$$\alpha^t = \alpha_1^t - \alpha_2^t$$

   a

$$\theta^t = \frac{1}{2}(|\alpha_1^t|^2 - |\alpha_2^t|^2).$$

4. Hledá se taková dvojice μ^j a σ^j, pro kterou je splněno

$$\frac{\langle \alpha^t, \mu^j \rangle - \theta^t}{\sqrt{\langle \alpha^j, \sigma^j \cdot \alpha^t \rangle}} \le r, \quad \text{pro } j \in J_1 \tag{4.36}$$

nebo

$$\frac{\theta^t - \langle \alpha^t, \mu^j \rangle}{\sqrt{\langle \alpha^j, \sigma^j \cdot \alpha^t \rangle}} \le r, \quad \text{pro } j \in J_2. \tag{4.37}$$

5. Není-li nalezana ani jedna dvojice μ^j a σ^j, je α^t a θ^t řešením úlohy. Pak se nastaví $solution = 1$ a algoritmus se ukončí.

6. Když pro nějaké $\mu^j, \sigma^j, j \in J_1$ je slpněna nerovnice (4.36), potom se vypočte vektor

$$x^t = \mu^j - \frac{r}{\sqrt{\langle \alpha^t, \sigma^j \cdot \alpha^t \rangle}} \cdot \sigma^j \cdot \alpha^t$$

a nový vektor α_1^{t+1} podle vzorce

$$\alpha_1^{t+1} = \alpha_1^t \cdot (1-k) + x^t \cdot k,$$

kde

$$k = \min_k \left(1, \frac{\langle \alpha_1^t - \alpha_2^t, \alpha_1^t - x^t \rangle}{|\alpha_1^t - x^t|^2} \right).$$

Vektor $\alpha_2^{t+1} = \alpha_2^t$. Zvětší se čítač kroků $t = t + 1$ a přejde se na krok 2.

7. Když pro nějaké $\mu^j, \sigma^j, j \in J_2$ je slpněna nerovnice (4.37), potom se vypočte vektor

$$x^t = \mu^j + \frac{r}{\sqrt{\langle \alpha^t, \sigma^j \cdot \alpha^t \rangle}} \cdot \sigma^j \cdot \alpha^t$$

a nový vektor α_2^{t+1} podle vzorce

$$\alpha_2^{t+1} = \alpha_2^t \cdot (1-k) + x^t \cdot k,$$

kde

$$k = \min_k \left(1, \frac{\langle \alpha_2^t - \alpha_1^t, \alpha_2^t - x^t \rangle}{|\alpha_2^t - x^t|^2} \right).$$

Vektor $\alpha_1^{t+1} = \alpha_1^t$. Zvětší se čítač kroků $t = t + 1$ a přejde se na krok 2.

4.9 Řešení Andersonovy úlohy pomocí nehladké gradientní optimalizace

Zde uvedeme jeden alternativní způsob, který také hledá řešení zobecněné Andersonovy úlohy a jenž je implementačně velice jednoduchý.

Řešení zobecněné Andersonovy úlohy je ekvivalentví optimalizačnímu problému (4.23), kdy se maximalizuje funkce více proměnných $\min_{j \in J} r(\alpha, \mu^j, \sigma^j)$. Jak jsme již uvedli, jedná se o unimodální funkci, která není konkávní a nemá diferenciál. Proto nelze použít klasických gradientních metod optimalizace.

V Knize je popsáno řešení pomocí *zobecněného gradientu*, který je definován i pro nehladké funkce, což je i náš případ.

Definice 4.1 *Zobecněný gradient*

Nechť je X konečněrozměrný lineární prostor, na kterém je definována konkávní funkce $f: X \to R$. Nechť x_0 je zvolený bod a $g(x_0)$ je takový vektor, že funkce

$$f(x_0) + \langle g(x_0), (x - x_0) \rangle$$

závislá na vektoru x není menší než $f(x)$ pro libovolné $x \in X$. Můžeme tedy psát

$$f(x_0) + \langle g(x_0), (x - x_0) \rangle \geq f(x).$$

Vektor $g(x_0)$ se nazývá zobecněný gradient *v bodě x_0.* ▲

Optimalizace se zobecněným gradientem je popsána větou, která je uvedena v diskusi k páté kapitole Knihy. Důkaz této věty lze nalézt v [9], [10].

Věta 4.3 *O gradientní optimalizaci se zobecněným gradientem. Nechť je $f: X \to R$ konkávní funkce; $\gamma_i, i = 1, 2, ..., \infty$, je posloupnost kladných čísel, která konverguje k nule a jejíž součet je nekonečný; f^* je*

$$\max_{x \in X} f(x)$$

a X_0 je množina takových bodů $x \in X_0$, ve kterých se dosahuje maxima; x_0 je jakýkoliv bod v prostoru X a

$$x_i = x_{i-1} + \gamma_i \cdot \frac{g(x_i)}{|g(x_i)|},$$

kde $g(x_i)$ je zobecněný gradient funkce f v bodě x_i. V tomto případě platí

$$\lim_{i \to \infty} f(x_i) = f^*,$$

$$\lim_{i \to \infty} \min_{x \in X_0} |x_i - x| = 0.$$

▲

Věta 4.3 požaduje, aby optimalizovaná funkce byla konkávní. Ale v našem případě tato podmínka není splněna. Přes tuto nesrovnalost lze uvedenou metodu použít pro řešení zobecněné Andersonovy úlohy. K tomuto kroku nás opravňují výsledky praktických pokusů.

V Knize je dokázáno, že zobecněný gradient optimalizované funkce $\min_{j \in J} r(\alpha, \mu^j, \sigma^j)$ je roven bodu libovolnému dotyku $x_0^j, j \in J_0$. Množina J_0 určuje rozdělení s nejmenší vzdáleností od rozdělujíc nadroviny, jak jsme již dříve definovali v (4.24). Jak výpočet těchto bodů, tak samotný postup optimalizace je velice jednoduchý.

Poslední věcí, kterou je třeba určit, je volba řady čísel γ_j. V našich experimentech se nám osvědčila posloupnost definována

$$\gamma_t = \frac{K}{t},$$

kde K je kladná konstanta a t určuje číslo iterace. Konstantu K je možno v algoritmu měnit, její implicitní hodnota je $K = 1$.

Vstupy i výstupy algoritmu jsou shodné s jejich popisem v odstavci 4.5.2, kromě dvou argumentů. Prvním z nich je kladné číslo Δ_r, které určuje minimální změnu kvality řešení $\min_{j \in J} r(\alpha, \mu^j, \sigma^j)$ ve dvou po sobě následujících krocích. Tím je definováno ukončovací kritérium úlohy, jejíž řešení je zaručeno podle věty 4.3 až v nekonečnu. Algoritmus navíc vrací kvalitu řešení, která byla dosažena při ukončení algoritmu, neboli hodnotu funkce $\min_{j \in J} r(\alpha, \mu^j, \sigma^j)$.

Algoritmus 4.12 Řešení Andersonovy úlohy pomocí gradientní optimalizace

Vstup: μ^j, σ^j *kde* $j = 1, 2, ...K, J = J_1 \cup J_2, t_{max}, \Delta_r, t_s, \alpha_s, \theta_s$

Výstup: $\alpha, \theta, solution, r_{min}, t$

1. Provedeme trasformaci (4.9) směsi gaussovských rozdělení $\{(\mu^j, \sigma^j) | j \in J_1 \cup J_2\}$ na $\{(\mu^{\prime j}, \sigma^{\prime j}) | j \in J\}$.

$$\mu^{\prime j} = \begin{cases} \{\mu_1^j, ...\mu_n^j, 1\} & \text{pro} \quad j \in J_1, \\ -\{\mu_1^j, ...\mu_n^j, 1\} & \text{pro} \quad j \in J_2, \end{cases}$$

$$\sigma^{\prime j} = \begin{bmatrix} \sigma_{11}^j & ... & \sigma_{1n}^j & 0 \\ \vdots & & \vdots & 0 \\ \sigma_{n1}^j & ... & \sigma_{nn}^j & 0 \\ 0 & ... & 0 & 0 \end{bmatrix} \quad \text{pro } j \in J_1 \cup J_2.$$

V případě, že je zadán počáteční stav α_s, θ_s, pak přiřadíme

$$\alpha_s' = \{\alpha_{s_1}, ..., \alpha_{s_n}, -\theta_s\}.$$

Pro přehlednost budeme α', μ'^j *a* σ'^j *dále označovat bez čárky jako* α, μ^j *a* σ^j.

2. Inicializace algoritmu. Je-li zadán počáteční stav $\{t_s, \alpha_s\}$ přiřaď

$$t = t_s,$$
$$\alpha_t = \alpha_s,$$

jinak

$$t = 1,$$
$$\{\alpha_1 | \langle \alpha_1, \mu^j \rangle > 0, j \in J\}.$$

Není-li nalezeno žádné α_1, splňující podmínku (4.122), pak se nastaví $solution = -1$ a algoritmus se ukončí.

3. Začátek iterace. Je-li dosaženo maximálního kroku, tj. $t = t_{max}$, nastaví se $solution = 0$ a přejde se na poslední krok 7. V opačném případě se pokračuje v iteraci krokem 4.

4. Hledá se bod dotyku x_0 a číslo r_{min}^t určující kvalitu současného řešení. Kvalita řešení je úměrná nejmenší vzdálenosti mezi střední hodnotou μ_j a bodem dotyku x_0^j, pro všechna rozdělení $j \in J$. Nejprve se určí r_{min}^t jako

$$r_{min}^t = \min_{j \in J} r(\alpha_t, \mu_j, \sigma_j),$$

kde $r(\alpha, \mu_j, \sigma_j)$ se spočítá podle vzorce

$$r(\alpha, \mu_j, \sigma_j) = \frac{\langle \alpha_t, \mu_j \rangle}{\sqrt{\langle \alpha_t, \sigma_j \cdot \alpha_t \rangle}}.$$

A dále se určí jeden bod dotyku x_0, pro který platí

$$x_0 = \mu_j - \frac{r_{min}^t}{\sqrt{\langle \alpha_t, \sigma_j \cdot \alpha \rangle}} \sigma_j \cdot \alpha_t, j \in J^0.$$

Pro množinu indexů J^0 platí, $r_{min}^t = \min_{j \in J^0} r(\alpha_t, \mu_j, \sigma_j)$.

5. Vypočte se nová hodnota vektoru α_{t+1}, jako

$$\alpha_{t+1} = \alpha_t + \gamma_t \cdot \frac{x_0}{|x_0|},$$

kde volba kroku γ_t, je určena přímo z kroku iterace

$$\gamma_t = \frac{1}{t}.$$

6. Kriterium ukončení. Je-li změna kvality řešení menší než zadaná mez Δ_r, což znamená

$$|r^t_{min} - r^{t-1}_{min}| < \Delta_r,$$

pak nastav $solution = 1$ a prejdi na krok 7. Jinak zvětši počítadlo kroků $t = t + 1$ a pokračuj v iteraci přechodem na krok 3.

7. Ukončení algoritmu. Proved se inverzní transformace z kroku 1. Nalezené řešení α_t se opět označí jako α' a parametry rozdělující nadroviny v původním prostoru se určí jako

$$\begin{aligned} \alpha &= \{\alpha'_1, ..., \alpha'_n\} \\ \theta &= \alpha`_{n+1}. \end{aligned}$$

Algorimus také vrací dosaženou kvalitu řešení $r_{min} = r^t_{min}$.

---

5. Kvadratická diskriminační funkce

5.1 Vyrovnání příznakového prostoru

V kapitole 4 jsme popsali lineární rozhodovací strategii a několik algoritmů, které tuto úlohu řeší. Všechny uvedené algoritmy předpokladají, že rozdělované množiny bodů je možno od sebe oddělit nadrovinou, neboli množiny jsou lineárně separovatelné. V obecném případě ale tento předpoklad nelze zaručit a pro rozdělení množin bodů je nutné použít jinou plochu než lineární (nadrovinu).

Vyhledávání nelineární rozdělující nadplochy lze převést na problém vyhledávání nadroviny, jenž jsme vyřešili v kapitole 4. Této metodě se říká *vyrovnávání příznakového prostoru* a můžeme ji použít v případě, když se ví, že se nelineární rozdělujcí nadplocha $f: X \to R$ může vyjádřit ve tvaru

$$f(x) = \sum_{j \in J} \alpha_j f_j(x), \tag{5.1}$$

kde $f_j: X \to R$, jsou známé funkce, které mohou být nelineární. Neznámé jsou jen součinitele α_j. V tomto případě se vyhledávání rozdělující nadplochy, která je ve vstupním prostoru X nelineární, redukuje na vyhledávání lineární nadplochy (nadroviny) ve vyrovnaném prostoru Y tak, že bod $x \in X$ odpovídá bodu $y \in Y$, jehož j-tá souřadnice je $f_j(x)$.

Nalezneme-li řešení ve vyrovnaném prostoru Y, pomocí algoritmů s lineární rozdělující nadplochou, lze výsledek trasformovat na nelineární rozdělujcí nadplochu v původním prostoru X a nebo rozpoznávat trasformované body přímo ve vyrovnaném prostoru pomocí nalezené nadroviny.

5.2 Aplikace vyrovnání na kvadratickou diskriminační funkci

V našem programovém balíku jsme implementovali vyrovnání příznakového prostoru pro kvadratickou rozdělující nadplochu. Tato rozdělující nadplocha je v rozpoznávání významná, například na ni vede optimální bayesovská strategie pro případ normálního rozdělení hustot pravděpodobností.

Původní n-rozměrný příznakový prostor X se trasformuje na vyrovnaný příznakový prostor Y, jehož dimenze je $2n + \frac{1}{2}n(n-1)$. Trasformace bodů z příznakového prostoru

X na vyrovnaný prostor Y má tvar

$$\begin{array}{rcccccc} y = (& x_1, & x_2, & ..., & x_i, & ..., & x_{n-1}, & x_n, \\ & x_1x_1, & x_1x_2, & ..., & x_1x_i, & ..., & x_1x_{n-1}, & x_1x_n, \\ & & x_2x_2, & ..., & x_2x_i, & ..., & x_2x_nn-1, & x_2x_n \\ & & & & & & \vdots & \\ & & & & x_ix_i, & ..., & x_ix_{n-1}, & x_ix_n \\ & & & & & & \vdots & \\ & & & & & & x_{n-1}x_{n-1}, & x_{n-1}x_n, \\ & & & & & & & x_nx_n). \end{array}$$

Uvedená trasformace je v našem programovém balíku implementována pro libovolný konečně rozměrný prostor, samozřejmě s ohledem na paměť.

Dále je v našem balíku nástrojů funkce převádějící parametry z vyrovnaného 5D příznakového prostoru na parametry obecné kvadratické funkce v původním 2D prostoru. Na následujících řádcích je tento převod popsán. Obecná kvadratická rozdělující funkce je ve 2D popsána následujícími rovnicemi

$$\langle x^T, A_{2\times 2}.x\rangle + \langle b_{2\times 1}, x\rangle + c_{1\times 1} = 0, \tag{5.2}$$

$$[x_1, x_2] \begin{bmatrix} a_{11}, & a_{12} \\ a_{21}, & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [b_1, b_2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + c = 0.$$

Dále ji vyjádříme do tvaru vhodného pro vyrovnávání příznakového prostoru (5.1)

$$\sum_{j\in J} \alpha_j f_j(x) = b_1x_1 + b_2x_2 + a_1x_1^2 + (a_2 + a_3)x_1x_2 + a_4x_2^2 + c.$$

Po porovnání s koeficienty nadroviny

$$\alpha_1y_1 + \alpha_2y_2 + \alpha_3y_3 + \alpha_4y_4 + \alpha_5y_5 = \theta$$

dostaneme převodní vztahy z nadroviny ve vyrovnaného prostoru na kvadratickou funkci v původním prostoru

$$\begin{array}{rcl} b_1 & = & \alpha_1, \\ b_2 & = & \alpha_2, \\ a_1 & = & \alpha_3, \\ a_2 & = & \frac{1}{2}\alpha_4, \\ a_3 & = & \frac{1}{2}\alpha_4, \\ a_4 & = & \alpha_5, \\ c & = & -\theta \end{array}$$

6. Algoritmy učení

Pro rozpoznávání objektů je zapotřebí jejich statistický model. Obvykle je popsán podmíněnou hustotou pravděpodobnosti $p(x|k)$, která vyjadřuje pravděpodobnost pozorování x za podmínky, že objekt je ve stavu k (uvedený statistický model například potřebujeme pro konstrukci optimální Bayesovské strategie). V případě, že naše znalost o objektu, respektive o hustotě pravděpodobnosti $p(x|k)$, není dostatečná, můžeme pro její doplnění použít algoritmy učení.

Zde se budeme zabývat případem, kdy se hustota pravděpodobnosti $p(x|k)$ dá brát jako soubor $|K|$ funkcí $p_k(x)$ proměnné x. Funkce $p_k(x)$ popisuje hustotu pravděpodobnosti pozorování x pro k-tou třídu. Dále předpokládáme, že typ funkce $p_k(x)$ je známý, ale neznámé jsou její parametry a_k. Hledaný statistický model pro všechny třídy pak můžeme označit jako $p(x, a_k), k \in K$. Hledané parametry $a_k, k \in K$ se budeme snažit doplnit algoritmy učení, jedná se o parametrický odhad.

6.1 Algoritmy učení bez učitele

V kapitole 4 jsme se zabývali případem učení, kdy byla trénovací data označena indexem třídy. Zde se budeme zabývat opačným případem, kdy nemáme žádnou informaci o tom, do jaké třídy trénovací data patří. Algoritmy řešící tento problém se nazývají algoritmy učení bez učitele.

V knize je popsána velmi obecná třída algoritmů, které se snaží doplnit informaci o hustotě pravděpodobnosti $p(x|k)$ z neoznačených dat. Jejich činnost se skládá ze dvou kroků. V prvním kroku, fáze rozpoznávání, jsou předložená data klasifikována do tříd, pomocí optimální bayesovské strategie. V druhém kroku, fázi učení, se na základě výsledků rozpoznávání provede maximálně věrohodný odhad pro všechny třídy. Tyto kroky se opakují tak dlouho, až se statistické modely $p_k(x), k \in K$ přestanou měnit, neboli algoritmus se dostal do stacionárního bodu řešení. V Knize je uveden důkaz věty, která mluví o monotóní konvergenci ke globálním nebo lokálním statickým bodům.

Mezi speciálními případy, uvedené třídy algoritmů, patří například shluková analýza ISODATA nebo empirický bayesovský přístup podle H. Robbinse. Dále uvedeme algoritmus učení bez učitele, jak je popsán v šesté kapitole Knihy.

Formulace 6.1 Algoritmus učení bez učitele.

Nechť je $m^0 = (p^0(k), a_k^0)|k \in K)$ jistý počáteční soubor neznámých parametrů. Algoritmus učení bez učitele je algoritmus, který krok za krokem buduje posloupnost modelů $m^1, m^2, ..., m^t, m^{t+1}$ podle následujících pravidel.

Nechť $m^t = (p^t(k), a_k^t|k \in K)$ je soubor, který se získá v kroku t algoritmu učení bez učitele. Následující model m^t se vypočítá ve dvou etapách. V první etapě se vypočítávají *čísla*

$$\alpha^t(i,k) = \frac{p^t(k)p(x_i, a_k^t)}{\sum_{k\in K} p^t(k)p(x_i, a_k^t)}$$

pro každé $i = 1, 2, ..., n$ *a každé* $k \in K$, *Jsou to čísla, která by se měla vypočítávat při rozpoznávání, pokud by hodnoty* $p^t(k)$ *a* a_k^t *byly skutečnými hodnotami. Čísla* $\alpha^t(i,k)$ *se podobají aposteriorním pravděpodobnostem stavu* k *za podmínky pozorování* x_i. *Ve druhé etapě se vypočítá nová hodnota modelu* m^{t+1} *podle*

$$p^{t+1}(k) = \frac{\sum_{i=1}^n \alpha^t(i,k)}{\sum_{k\in K}\sum_{i=1}^n \alpha^t(i,k)} = \frac{\sum_{i=1}^n \alpha^t(i,k)}{n},\ k \in K, \tag{6.1}$$

$$a_k^{t+1} = \operatorname*{argmax}_a \sum_{i=1}^n \alpha^t(i,k) \log p(x_i, a),\ k \in K. \tag{6.2}$$

Tímto způsobem vytváříme posloupnost modelů $m^0, m^1, ..., m^t$ *tak dlouho, dokud není splněna podmínka* $m^{t-1} = m^t$. ▲

Popsaný algoritmus jsme implementovali v našem programovém balíku pro případ, kdy přepokládáme, že statistický model je pro danou třídu určen hustotou pravděpodobnosti $p_k(x, a_k)$ s normálním rozdělením a nezávislými i závislými příznaky. Neznámými parametry a_k jsou v tomto případě střední hodnoty μ_k a kovarianční matice σ_k.

Hlavním problémem je určení maximálního odhadu (6.2) pro parametry μ_k a σ_k, který v Knize není uveden. Hledání maximálně věrohodného odhadu se bude řešit i v případě algoritmu minimaxního učení, a proto tento problém vyřešíme pro oba případy najednou v samostatné sekci.

Posledním problémem, který zbývá vyřešit při implementaci tohoto algoritmu, je určení počátečního modelu. Počáteční model m^0 je v našem případě soubor $|K|$ vektorů středních hodnot μ_k a kovariančních matic σ_k. Vektorům středních μ_k hodnot přiřazujeme hodnoty $|K|$ bodů z trénovací množiny X. Tyto body volíme buď náhodně a nebo jednoduše vybereme prvních $|K|$ zadaných bodů. Použití první nebo druhé metody lze zajistit speciálním vstupním argumentem algoritmu.

Počáteční kovarianční matice volíme $\sigma = c \cdot E$, kde hodnota konstanty c závisí na měřítku vstupních dat. Kovarianční matice se v algoritmu učení počítají z hodnot $\alpha(i,k)$, jenž určují aposteriorní pravděpodobnosti i-tého bodu ze vstupní množiny. Pro jejich výpočet se používá bayesovská strategie rozpoznávání, která v našem případě počítá s normálně rozdělenou hustotou pravděpodobnosti. Hodnota této pravděpodonosti je dána exponenciální funkcí, jejímž argumentem je záporně braná Mahalanobisova vzdálenost. V případě nerovnoměrnosti měřítka dat a špatně zvolené kovarianční matice, může být tato vzdálenost natolik vysoká, že při vyhodnocování exponeciální funkce nestačí přesnost reálných čísel používaných v Matlabu. Z uvedeného důvodu provádíme první klasifikaci podle minima vzdálenosti, která je speciálním případem Bayesova klasifikátoru pro $\sigma = c \cdot E$. Kovarianční matice spočítaná z takto klasifikovaných dat již odpovídá jejich skutečnému měřítku a v dalším kroku již lze použít výpočet pro obecnou bayesovskou strategii.

6.2 Minimaxní učení

Výsledek metod učení, které používají maximálně věrohodný odhad, podstatně závisí na statistických vlastnostech objektů x v trénovací posloupnosti. Především je požadováno, aby jejich výběr byl náhodný a nezávislý. V případech, kdy se uvedený požadavek nedá zajistit, můžeme použít algoritmu minimaxního učení, který pracuje s nenáhodnými trénovacími daty. Algoritmus minimaxního učení se snaží vytvořit statistický model tak, aby objekty předložené v trénovovací množině byly co nejpravděpodobnějšími reprezentanty své třídy. Jinými slovy, snaží se určit parametry hustoty pravděpodobnosti $p(x|k)$ tak, aby její hodnota pro data z trénovací množiny $x \in X(k)$ byla co největší. Zde se již nejedná o případ učení bez učitele jako v předchozím odstavci, ale naopak po učiteli požadujeme, aby do trénovací množiny pro danou třídu vybral takové objekty, které ji co nejlépe popisují.

V osmé kapitole Knihy je popsán obecný minimaxní algoritmus a jeho aplikace na markovské posloupnosti. My zde nejprve uvedeme obecný algoritmus a pak ho aplikujeme na model s normálním rozložením hustoty pravděpodobnosti a korelovanými příznaky.

Budeme se zabývat odhadem statistického modelu pro jednu třídu k, proto budeme pracovat s označením trénovací množiny bez indexu X a také hustotu pravděpodobnosti $p(x|k)$ zapisujeme zkráceně jako $p(x)$. Jiná hustota pravděpodobnosti se v minimaxním algoritmu nevyskytuje.

Minimaxní algoritmus vytváří posloupnost modelů, přičemž první je určen maximálně věrohodným odhadem parametrů trénovací množiny X. V dalších krocích se určí objekt x trénovací množiny, který má nejmenší pravděpodobnost, tj. platí pro něj $x = \operatorname{argmin}_{x \in X} p(x)$. Takto nalezené x se přidá do trénovací množiny jako nový objekt, neboli zvýší se jeho četnost $n^{t+1}(x) = n^t(x) + 1$. Uvedeným způsobem se zvýší pravděpodobnost $p(x)$. Poznamenejme, že pro případ normálního rozdělení se vektor střední hodnoty μ posune ve směru nalezeného x. Po přidání nového prvku se opět přejde na krok provádějící maximálně věrohodný odhad. Uvedený postup probíhá tak dlouho, dokud není dosaženo požadované přesnosti, která je daná hodnotou věrohodnostní funkce a jejím dolním odhadem. Pro podrobnější popis odkazujeme na Knihu, kde je uveden i následující obecný algoritmus.

Algoritmus 6.1 Minimaxní odhad modelu $p \in P$

1. Uživatel zadá požadovanou přesnost řešení úlohy $\varepsilon > 0$. Určí čísla $n(x) = 1$ pro všechna x z trénovací množiny X a začne se iterací $t = 1$.

2. Vypočítají se hodnoty
$$\alpha^t = \frac{n^t(x)}{\sum_{x \in X} n^t(x)}.$$

3. Vypočíta se nejvěrohodnější odhad
$$p^t = \operatorname*{argmax}_{p \in P} \sum_{x \in X} \alpha^t(x) \log p(x). \tag{6.3}$$

4. Pokud je splněna nerovnost
$$\sum_{x \in X} \alpha^t(x) \log p^t(x) - \min_{x \in X} \log p^t(x) < \varepsilon, \tag{6.4}$$
potom algoritmus skončí a p^t je řešením úlohy.

5. Pokud není nerovnost (6.4) splněna, označí se
$$x' = \operatorname*{argmin}_{x \in X} p^t(x)$$

a vypočítají se nové hodnoty čísel $n(x), x \in X$, že

$$\begin{aligned} n^{t+1}(x') &= n^t(x') + 1, \\ n^{t+1}(x) &= n^t(x), \quad x \in X, x \neq x'. \end{aligned}$$

6. Přejde se na následující $(t+1)$-ní iteraci, a to přechodem na krok 2.

Jak je vidět z popisu algoritmu, minimaxní odhad lze použít pro statistický model, u kterého známe výpočet maximálně věrohodného odhadu. V Knize je uveden důkaz tohoto tvrzení.

V našem programovém balíku jsme implementovali algoritmus minimaxního odhadu pro model s normálním rozdělením hustoty pravděpodobnosti, za předpokladu závislých (korelovaných) příznaků. Algoritmem budeme odhadovat parametry normálního rozdělení, jimiž jsou střední hodnota μ a kovarianční matice σ. Poznamenejme, že takto definovaná úloha minimaxního učení je ekvivalentní úloze hledání elipsoidu s co možná nejmenším poloměrem, který obejme všechny body z trénovací množiny.

Pro sestavení algoritmu potřebuje vyřešit maximálně věrohodný odhad (6.3), jak je uveden v kroku 3 algoritmu. Tento problém vyřešíme v dalším odstavci, kde dostame vztahy pro výpočet střední hodnoty μ a kovarianční matice σ. Po vložení uvedených vztahu do kroku (6.3) algoritmu, dostaneme výsledný algoritmus minimaxního učení pro normální rozdělení.

6.3 Maximálně věrohodný odhad pro normální rozdělení

Maximálně věrohodný odhad pro normální rozdělení je dobře znám, ale v našem případě je maximalizované kritérium odlišné od klasické úlohy. Přesněji řečeno, hustota pravděpodobnosti je vážena číslem $\alpha(x)$, takže optimalizované kritérium má tvar

$$\{\mu, \sigma\} = \operatorname*{argmax}_{\mu,\sigma} \sum_{x \in X} \alpha(x) \log \left(\frac{1}{(2\pi)^{\frac{l}{2}} \sqrt{\det(\sigma)}} e^{-\frac{1}{2}(x-\mu)^T \sigma (x-\mu)} \right),$$

kde číslo l-je udává dimenzi a $det(\sigma)$ je determinant matice. Za předpokladu $\alpha(x) = 1, x \in X$ se jedná o klasický odhad parametrů normálního rozdělení, kdy se μ spočítá jako střední hodnota a σ jako kovarianční matice. V našem případě je však $\alpha(x)$ závislé na parametru x. Pro případ učení bez učitele je číslo $\alpha(x)$ rovno podmíněné pravděpodobnosti $p(k|x)$ a v případě minimaxního učení je to relativní výskyt prvku x v množině X.

V druhém případě, algoritmu minimaxního učení, lze popsanou úlohu celkem jednoduše vyřešit, a to díky vlastnostem čísla $\alpha(x)$, které umožnuje převést uvedený problém na klasickou úlohu pro $\alpha(x) = 1, x \in X$.

Při optimalizaci kritéria v případě algoritmu učení bez učitele jsme již museli optimalizovat kritérium pro obecnou hodnotu $\alpha(x)$. Podařilo se nám odvodit vztah pro výpočet parametrů μ a σ za předpokladu, že jsou příznaky měření $x \in X$ nezávislé. Pro případ závislých (korelovaných) příznaků jsme navrhli postup vycházející pouze z intuice, který kombinuje oba dva vyřešené případy. Vzhledem k tomu, že maximálně věrohodný odhad normálního rozdělení je dosti známý, dá se předpokládat i existence řešení našeho problému, nicméně se nám ho nepodařilo v literatuře nalézt. V knize [2] je odvozen maximálně věrohodný odhad pro případ, kdy je kovarianční matice diagonální a prvky na diagonále jsou stejné, což je málo obecné.

6.3.1 Maximálně věrohodný odhad pro algoritmus minimaxního učení

Zde vyřešíme kritérium (6.3) pro minimaxní algoritmus učení, které má tvar

$$p^* = \operatorname*{argmax}_p \sum_{x \in X} \alpha(x) \log p(x), \tag{6.5}$$

kde $p(x)$ je hustota pravděpodonosti (v našem případě normálně rozdělená) a $\alpha(x)$ je relativní výskyt prvku x, který se spočte

$$\alpha(x) = \frac{n(x)}{\sum_{x \in X} n(x)}. \tag{6.6}$$

Číslo $n(x)$ počet prvků x v trénovací množině X. Vztah pro relativní četnost (6.6) můžeme dosadit do kritéria (6.5), čímž dostaneme

$$p^* = \operatorname*{argmax}_p \sum_{x \in X} \frac{n(x)}{\sum_{x \in X} n(x)} \log p(x). \tag{6.7}$$

Díky tomu, že $\sum_{x \in X} n(x)$ je pro danou množinu X konstantní, můžeme kritérium (6.7) přepsat na tvar

$$p^* = \operatorname*{argmax}_p \sum_{x \in X} n(x) \log p(x). \tag{6.8}$$

Nyní zavedeme množinu X', která obsahuje všechny body z X tolikrát, kolik udává číslo $n(x)$. Díky tomu bude pro množinu X' číslo $n'(x) = 1, x \in X$, a proto můžeme kritérium (6.8) přepsat do tvaru

$$p^* = \operatorname*{argmax}_p \sum_{x \in X'} \log p(x). \tag{6.9}$$

Za předpokladu, že $p(x)$ je normálně rozdělená distribuční funkce, je maximálně věrohodný odhad pro kritérium ve tvaru (6.9) roven střední hodnotě μ a kovarianční matici σ získané z bodů množiny X'. Tyto statistické momenty můžeme zapsat jako

$$\begin{aligned} \mu = \frac{\sum_{x \in X'} x}{n} &= \frac{\sum_{x \in X} n(x) x}{\sum_{x \in X} n(x)} = \\ = \sum_{x \in X} \frac{n(x)}{\sum_{x \in X} n(x)} x &= \sum_{x \in X} \alpha(x) x. \end{aligned}$$

a podobně kovarianční matice

$$\begin{aligned} \sigma_{ij} &= \frac{\sum_{x \in X'} (x_i - \mu_i)(x_j - \mu_j)}{n} = \\ &= \frac{\sum_{x \in X} n(x)(x_i - \mu_i)(x_j - \mu_j)}{\sum_{x \in X} n(x)} = \\ &= \sum_{x \in X} \alpha(x)(x_i - \mu_i)(x_j - \mu_j), \end{aligned}$$

kde σ_{ij} je člen na i-tém řádku a v j-tém sloupci kovarianční matice, která je symetrická, takže stačí počítat koeficienty například jen pro její horní trojúhelníkovou část a ostatní doplnit podle vztahu $\sigma_{ij} = \sigma_{ji}$.

6.3.2 Maximálně věrohodný odhad pro algoritmus učení bez učitele

Zde se budeme zabývat kritériem (6.2) pro algoritmus učení bez učitele. Za předpokladu, že je hustota pravděpodobnosti normálně rozdělena a příznaky jsou nezávislé, může kritérium pro maximálně věrohodný odhad přepsat do tvaru

$$
\begin{aligned}
\{\mu_k, \sigma_k\} &= \underset{\mu_k,\sigma_k}{\operatorname{argmax}} \sum_{i=1}^{n} \alpha(i,k) \log \prod_{j=1}^{l} \frac{1}{\sqrt{2\pi\sigma_k}} e^{-\frac{(x_i^j-\mu_k^j)^2}{2\sigma_k^j}} = \\
&= \underset{\mu_k,\sigma_k}{\operatorname{argmax}} \sum_{i=1}^{n} \alpha(i,k) \sum_{j=1}^{n} \log \frac{1}{\sqrt{2\pi\sigma_k^j}} e^{-\frac{(x_i^j-\mu_k^j)^2}{2\sigma_k^j}},
\end{aligned}
$$

kde x_i^j, μ_k^j je j-tá složka vektoru $x_i = \{x_i^1, ..., x_i^l\}$, resp. μ_k a σ_k^j je j-tý diagonální prvek kovarianční matice σ_k. Jak je vidět, rozpadla se nám maximalizace (6.2) pro jednu třídu k, na maximalizaci $j = 1, ..., l$ kritérii tvaru

$$
\{\mu_k^j, \sigma_k^j\} = \underset{\mu_k,\sigma_k^j}{\operatorname{argmax}} \sum_{i=1}^{n} \alpha(i,k) \log \frac{1}{\sqrt{2\pi\sigma_k^j}} e^{-\frac{(x_i^j-\mu_k^j)^2}{2\sigma_k^j}}.
$$

Abychom určili bod maxima, položíme derivaci kritéria podle μ_k a σ_k rovnu nule,

$$
\begin{aligned}
\frac{\partial}{\partial \mu_k^j} \sum_{i=1}^{n} \alpha(i,k) \log \frac{1}{\sqrt{2\pi\sigma_k^j}} e^{-\frac{(x_i^j-\mu_k^j)^2}{2\sigma_k^j}} &= 0 \\
\frac{\partial}{\partial \sigma_k^j} \sum_{i=1}^{n} \alpha(i,k) \log \frac{1}{\sqrt{2\pi\sigma_k^j}} e^{-\frac{(x_i^j-\mu_k^j)^2}{2\sigma_k^j}} &= 0.
\end{aligned}
$$

Po úpravě dostaneme výsledný předpis pro výpočet μ_k a σ_k ve tvaru

$$
\begin{aligned}
\mu_k^j &= \frac{\sum_{i=1}^{n} \alpha(i,k) x_i^j}{\sum_{i=1}^{n} \alpha(i,k)}, \ j = 1, ..., l, \\
\sigma_k^j &= \frac{\sum_{i=1}^{n} \alpha(i,k)(x_i^j - \mu_k^j)^2}{\sum_{i=1}^{n} \alpha(i,k)}, \ j = 1, ..., l,
\end{aligned}
$$

a nový model po maximálně věrohodném odhadu je dán

$$
\begin{aligned}
\mu_k &= \{\mu_k^1, \mu_k^2 ..., \mu_k^l\} \\
\sigma_k &= \begin{bmatrix} \sigma_k^1 & 0 & ... & 0 \\ 0 & \sigma_k^2 & ... & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & ... & \sigma_k^l \end{bmatrix}.
\end{aligned}
$$

Pro obecný případ, kdy jsou příznaky nezávislé, jsme navrhli postup výpočtu intuitivně. Vycházíme z toho, že obecný případ musí zahrnovat obě specifické úlohy, které jsme již vyřešili. Zaprvé musí platit pro maximálně věrohodný odhad algoritmu minimaxního učení, kdy je hledán odhad pro jednu třídu. To znamená, že $p(k) = 1$ a platí

$$\alpha(i,k) = p(k|x) = \frac{p(x|k) \cdot p(k)}{p(x)} = \frac{p(x)}{p(x)} = 1.$$

Dále tento odhad musí platit pro druhý vyřešení případ, kdy jsou příznaky navzájem závislé. Obě uvedené podmínky splňuje výpočet

$$\begin{aligned} \mu_k^j &= \frac{\sum_{i=1}^n \alpha(i,k) x_i^j}{\sum_{i=1}^n \alpha(i,k)},\ j = 1,2,...l, \\ \sigma_k^{hj} &= \frac{\sum_{i=1}^n \alpha(i,k)(x_i^h - \mu_k^h)(x_i^j - \mu_k^j)}{\sum_{i=1}^n \alpha(i,k)},\ h = 1,...,l,\ j = 1,...l. \end{aligned}$$

Výsledný vektor střední hodnoty pro třídu k je určen vypočtenými složkami $\mu_k = \{\mu_k^1, \mu_k^2, ..., \mu_k^l\}$. Stejně tak kovarianční matice σ_k je určena členy σ_k^{hj}, kde h udává řádek a j sloupec.

7. Experimenty

V této kapitole popíšeme dva experimenty, které se snaží dokázat použitelnost našeho programového balíku i pro praktické problémy. První experiment hledá klasifikátor pro seismická data (měření otřesů). V druhém experimentu jsme se snažili otestovat funkčnost algoritmů řešících zobecněnou Andersonovu úlohu, které jsme v našem programovém balíku věnovali velký prostor.

Programy implementující první experiment jsou uloženy v samostatném adresáři s názvem *\experime\seismo* a druhý experiment v adresáři *\experime\anderson*. Tyto adresáře obsahují všechny programy a datové soubory potřebné pro jejich spuštění.

V dalším textu budeme zapisovat číselné údaje s desetinou tečkou místo čárky a to z důvodu snadnějšího převodu z Matlabu, který tento číselný zápis používá.

7.1 Seismická data

Data se skládájí ze záznamu parametrů odstřelů z lomu v Tušimicích a z měření účinků těchto odstřelů dvěmi seismickými stanicemi. Seismické stanice mají různou polohu, která je neměnná. Výsledkem jejich měření je maximální zaznamenaná amplituda seismického otřesu v daném místě. Velikost maximální amplitudy odpovídá ničivému účinku otřesu. Cílem naší analýzy je určit závislost mezi parametry odstřelů a amplitudou zaznamenanou seismickou stanicí. Experimenty jsme rozdělili na dva nezávislé případy, podle toho jestli je použit údaj o amplitudě ze stanice ve Spořicích a nebo v Černicích.

Amplitudu otřesu lze diskretizovat podle zadané prahové hodnoty na dva intervaly, čímž dostaneme problém klasifikace do dvou tříd, neboli dichotomii. Závislost amplitudy A_{max}, zadaného prahu p a příslušnosti měření k třídě $k = \{1, 2\}$ (respektive intervalu 1 nebo 2) je

$$k = \begin{cases} 1, & \text{pro} \quad A_{max} < p, \\ 2, & \text{pro} \quad A_{max} \geq p. \end{cases} \tag{7.1}$$

První interval, třída $k = 1$, odpovídá slabým otřesům a druhý interval, třída $k = 2$, odpovídá silným otřesům. Určení co je slabý a co silný otřes, zavisí na velikosti prahu p. Pro praxi má význam především hodnota $p = 1$, která je určena normou.

Vstupní data jsou zapsána v textovém souboru. Čísla jsou v řádku oddělena čárkami. Na každém řádku je stejný počet čísel, které odpovídají jednomu měření. Každý sloupec odpovídá jednomu příznaku. Popis příznaků v jednotlivých sloupcích je uveden v tabulce 7.1.

V dalším popisu budeme označovat příznaky čísly jak jsou uvedeny v tabulce 7.1. Ve vstupním souboru je celkem 108 položek měření. Údaj amplitudy ve sloupci 7 a 8 není pro všechny odstřely naměřen. Pro stanici ve Spořicích (sloupec 7) je 92 měření a pro stanici v Černicích (sloupec 8) je 88 měření. Nedefinované hodnoty jsou označeny číslem -1.

Tabulka 7.1 Popis vstupních textových dat se záznamy odpalů.

Číslo sloupce	Popis příznaku
1	Číslo odstřelu.
2	Odstřelená metráž [bm].
3	Množství použité trhaviny [kg].
4	Počet použitých rozbušek [ks].
5	Odstřelená kubatura [m^3]. Množství rozpojené horniny.
6	Měrná spotřeba [$kg.m^{-3}$].
7	Maximální amplituda A_{max} změřená ve Spořicích.
8	Maximální amplituda A_{max} změřená v Černicích .
9	Souřadnice S-JTSK, X.
10	Souřadnice S-JTSK, Y.

Tabulka 7.2 Soubor s měřením odpalů seismickou stanicí ve Spořicích.

Matice dat X	příznaky 2, 3, 5, 9, 10
Identifikátor třídy I	navzorkovaná A_{max} (7) s prahem p=1
Počet měření	92
Počet příznaků	5
Vektor Amax	původní vektor hodnota A_{max}

7.1.1 Provedené experimenty

Vytvoření vstupních souborů

Nejprve jsme vytvořili dva soubory v datovém formátu našeho programového balíku, pro případ měření ve Spořicích a v Černicích, ze vstupního textového souboru *tusimice.txt*. Při tom jsme provedli selekci příznaků, podle doporučení seismologa. Pro stanici ve Spořicích jsou data vygenerována programem *experim1.m* a výsledek je uložen v souboru *spo.mat*, jehož popis je uveden v tabulce 7.2. Podobně pro Černice jsou data generována programem *experim2.m*, výsledek je v souboru *cer.mat* a popis v tabulce 7.3.

Hledání rozdělující strategie pro slabé a silné otřesy.

Naším cílem bylo nalézt rozhodovací strategii, která by rozdělila vstupní data na slabé a silné otřesy bez znalosti měření A_{max} seismických stanic. Jak jsme již posali výše, jsou slabé a silné otřesy určeny prahovou hodnotou amplitudy A_{max}. V tomto experimentu jsme se snažili měnit prahovou hodnotu p a zjišťovat, zdali pro takto určené třídy jsou data rozdělitelná (separabilní) nebo ne. Jako rozdělující nadplochu jsme zvolili kvadratickou funkci.

Na vstupní data jsme nejprve použili algoritmus vyrovnání příznakového prostoru (funkce *qtransf.m*), a pro takto získaná data jsme hledali lineární rozdělující nadlochu algoritmem Linear Support Vectors Machine (funkce *svm.m*), která se v původním prostoru zobrazuje jako kvadratická funkce.

Experiment pro stanici ve Spořících je implementován v programu *experim3.m* a pro stanici v Černicích v programu *experim4.m*. Výsledky těchto experimentů jsou zobrazeny v tabulce 7.4. Kromě prahové hodnoty a informace o rozdělitelnosti kvadratickou funkcí je v tabulce uveden počet měření patřících do třídy slabých a do silných otřesů.

Z výsledků je vidět, pro které prahové hodnoty p jsou data rozdělitelná kvadratickou

Tabulka 7.3 Soubor s měřením odpalů seismickou stanicí v Černicích.

Matice dat X	příznaky 2, 3, 5, 9, 10
Identifikátor třídy I	navzorkovaná A_{max} (8) s prahem p=1
Identifikátor třídy I	amplituda A_{max}, příznak 7
Počet měření	88
Počet příznaků	5
Vektor Amax	původní vektor hodnota A_{max}

Tabulka 7.4 Výsledky rozdělení měřených odpalů ve Spořicích a Černicích na slabé a silné otřesy pomocí kvadratické funkce.

	Spořice			Černice		
Práh	Kvadraticky rozdělitelné	Slabých otřesů	Silných otřesů	Kvadraticky rozdělitelné	Slabých otřesů	Silných otřesů
0.10	ANO	2	90	ANO	1	1
0.15	ANO	3	89	ANO	3	85
0.20	ANO	7	85	ANO	3	85
0.25	ANO	10	82	ANO	4	84
0.30	ANO	15	77	NE	9	79
0.35	NE	28	64	NE	17	71
0.40	NE	37	55	NE	25	63
0.45	NE	48	44	NE	39	49
0.50	NE	54	38	NE	44	44
0.55	NE	62	30	NE	49	39
0.60	NE	69	23	NE	56	32
0.65	NE	75	17	NE	60	28
0.70	NE	77	15	NE	65	23
0.75	NE	80	15	NE	67	21
0.80	NE	85	7	NE	69	19
0.85	NE	87	5	NE	72	16
0.90	ANO	90	2	ANO	73	15
0.95	ANO	90	2	NE	74	14
1.00	ANO	90	2	ANO	75	13
1.05	ANO	91	1	ANO	77	11
1.10	ANO	91	1	ANO	77	11

funkcí. Hodnota prahu p, pro nějž můžou být data rozdělena, tvoří dva souvislé intervaly, jak je vidět v tabulce. Počet trénovacích dat sice není příliš velký, ale skutečnost, že jsou plně kvadraticky rozdělitelná pro prahové hodnoty ze souvislých intervalů dává naději, že tato rozdělitelnost vychází z fyzikálních vlastností problému.

Použití algoritmu učení bez učitele

Na oba vstupní soubory jsme použili algoritmus učení bez učitele, který hledá statistický model vstupních dat. O datech předpokládáme:

- Dají se klasifikovat do 2. tříd.
- Mají normálně rozdělené hustoty pravděpodobností $p(x|k)$.
- Příznaky jsou nezávislé.

Takto formulovanou úlohu řeší program *unsuni.m* z našeho programového balíku, jehož výstupem jsou:

- Vektory středních hodnot: $\mu^k, k \in \{1, 2\}$.

- Kovarianční matice: $\sigma^k, k \in \{1, 2\}$.
- Apriorní pravděpodobnosti: $p(k), k \in \{1, 2\}$.
- Infromace o tom, jestli je získaný model stacionární bod učení a nebo není.
- Vektor výsledků klasifikace $C = \{c_1, c_2, ...c_n\}$ do třídy $k \in \{1, 2\}$ podle bayesovské rozhodovací strategie sestavené na základě získaného modelu.

Každé spuštění algoritmu učení bez učitele dává, respektive může dát jiné výsledky (jiný model), protože algoritmus začíná každé učení z náhodně vygenerovaného stavu. Pro stanici v Černicích (*cer.mat*) je popsaný experiment zapsán v programu *experim5.m*, výsledek učení se ukládá do souboru *cerlern.mat*. Podobně pro stanici ve Spořicích je to program *experim6.m* a soubor s výsledky *spoler.mat*.

Model dat získaných z algoritmu učení bez učitele je nutné dále analyzovat, abychom zjistili, jestli má nějaký fyzikální význam. My jsme s výsledným modelem provedli jeden experiment, který zde popíšeme.

Pro obržený statistický model jsme chtěli ověřit, jestli nějak koresponduje s očekávaným předpokladem, že lze klasifikovat do dvou tříd slabých a silných otřesů. Proto jsme se snažili nalézt závislost mezi prahem p, určujícím tyto intervaly podle (7.1), a chybou klasifikátoru sestaveného z vypočítaných (naučených) modelů. Jinými slovy, hledali jsme takový práh p, při kterém by bayesovská strategie rozhodování sestavená z výsledného modelu dávala co nejmenší chybu. Tento experiment je implementován v programu *experim7.m*. Jeho vstupními parametry je vstupní soubor (*spo.mat* nebo *cer.mat*) a soubor s výsledkem učení (*spolern.mat* nebo *cerlern.mat*). Výsledkem popisovaného experimentu je závislost hodnoty prahu p a odpovídající chyby klasifikace.

Popsaných experimentů s učením jsme provedli poměrně málo, neboť samotný algoritmus učení je časově náročný, stejně jako podrobná analýza získaného statistického modelu.

7.1.2 Výsledky experimentů se seismickými daty

Zjistili jsme, že podle parametrů odstřelů lze klasifikovat do třídy slabých a silných otřesů podle kvadratické rozhodovací strategie. Nalezli jsme intervaly prahové hodnoty amplitudy, určující třídy slabých a silných otřesů, které lze takto kvadraticky rozdělit. Do těchto intervalů patří i prakticky významná hodnota prahu $p = 1$, která je určena normou.

Dále jsme experimentovali s algoritmy učení bez učitele, jenž byly použity pro hledání statistických modelů vstupních dat. Z obdržených modelů se nám nepodařilo odvodit jejich fyzikální význam.

7.2 Test algoritmů zobecněné Andersonovy úlohy

Účelem tohoto testu je vyzkoušet účinnost algoritmů pro řešení zobecněné Andersonovy úlohy, které jsou v našem programovém balíku obsaženy. Budeme mezi sebou porovnávat algoritmy, které hledají optimální řešení a nakonec pro srovnání použijeme algoritmus ε-řešení Andersonovy úlohy. Popis testovaných algoritmů je uveden v tabulce 7.5.

Tabulka 7.5 Testované algoritmy hledající řešení zobecněné Andersonovy úlohy

	Jméno programu	Popis algoritmu
A	ganders.m	Naplnění obecné kostry, algoritmus 4.7: Hledání zlepšujícího směru snižovaním chyby nejhoršího rozdělení 4.7.3. Maximalizace kritéria diskretizací nezávislé proměnné 4.7.4.
B	ganders2.m	Naplnění obecné kostry, algoritmus 4.7: Hledání zlepšujícího směru aproximací gaussovským rozdělením s jednotkovou kovarianční maticí 4.7.3. Maximalizace kritéria diskretizací nezávislé proměnné 4.7.4.
C	gganders.m	Nehladká gradientní optimalizace, algoritmus 4.12
D	eanders.m	ε-řešení, algoritmus 4.11

Tabulka 7.6 Vstupní datové soubory pro testování algoritmů řešících zobecněnou Andersonovu úlohu.

	wineqn1.mat	wineqn2.mat
Dimenze příznakového prostoru N	104	104
Počet rozdělení v první třídě	1	20
Počet rozdělení v druhé třídě	2	20

7.2.1 Testovací data

Jako testovací soubor jsme použili data obsahující chemická měření třech skupin italských vín. Zdrojový soubor pochází ze združení Delve, http://www.cs.utoronto.ca/ delve/, které shromažďuje data pro rozpoznávání a strojové učení. Data jsou uložena v textovém souboru. Čísla jsou oddělena čárkami. Každý řádek obsahuje 14 číselných údajů. První řádek je identifikátor třídy, který může nabývat hodnot 1,2 a 3 podle toho o jakou odrůdu vína se jedná. Ostatních 13 řádků jsou různá měření vyjádřena reálnými čísly. Pro podrobný popis odkazujeme na soubor *wine.nam*, kde jsou původní informace. Důležitým poznatkem je to, že data jsou kvadraticky rozdělitelná, proto budeme hledat řešení Andersonovy úlohy ve vyrovnaném prostoru příznaků (nadrovina se v původním prostoru jeví jako kvadratická funkce).

Vstupní soubor jsme nejprve převedli do našeho interního formátu. Takto vytvořená data byla transfromována algoritmem vyrovnávajícím příznakový prostor. Pro řešení Andersonovy úlohy je třeba mít normální modely tříd. Vytvořili jsme dva různé modely.

V prvním modelu je třída 1 popsána jedním normálním rozdělením, jehož parametry jsme získali z transformovaných bodů, původně popisující první odrůdu vín. Třída 2 obsahuje dvě normální rozdělení získané z transformovaných bodů pro druhou a třetí odrůdu. Všechny parametry jsme určili maximálně věrohodným odhadem. Tato vstupní data jsou uložena v souboru *wineqn1.mat* a vygenerována byla programem *winexp1.m.*

V druhém modelu opět třída 1 popisuje první odrůdu a třída 2 druhou a třetí odrůdu vín. Rozdíl je v tom, že obě třídy jsou popsány 20 různými normálními rozděleními. Pro jejich vytvoření jsme rovnoměrně rozdělili vstupní množiny bodů na dvacet skupin a z nich udělali maximálně věrohodný odhad. Tato vstupní data jsou uložena v souboru *wineqn2.mat* a vygenerována byla programem *winexp2.m.* V tabulce 7.6 je pro přehlednost uveden jejich popis.

Tabulka 7.7 Test algoritmu *gganders.m* a *ganders.m* na souboru *wineqn1.mat*: dimenze N=104, počet normálních rozdělení $K_1 = 1$ a $K_2 = 2$.

gganders.m				ganders.m			
min r	chyba [%]	iterace	čas [s]	min r	chyba [%]	iterace	čas [s]
0.5957	27.57	1	4.29	0.5957	27.56	1	2.9
1.1281	12.96	200	19.1	1.1235	13.06	2	8.46
1.1294	12.94	400	33.3				
1.1297	12.93	600	47.8				
1.1299	12.93	800	62.4				
1.1300	12.92	1000	76.8				
1.1301	12.92	1200	91.1				
1.1302	12.92	1400	105.8				
1.1302	12.92	1600	120.4				
1.1303	12.92	1800	134.57				
1.1303	12.92	2000	149.57				

Tabulka 7.8 Test algoritmu *ganders2.m* a *eanders.m* na souboru *wineqn1.mat*: dimenze N=104, počet normálních rozdělení $K_1 = 1$ a $K_2 = 2$.

ganders2.m				eanders.m			
min r	chyba [%]	iterace	čas [s]	min r	chyba [%]	iterace	čas [s]
0.5957	27.56	1	2.9	0.9000	18.41	1	0.1
1.5682	5.86	10	49.9	0.9850	16.23	3	0.2
1.6843	4.61	20	94.5	1.0700	14.23	3	0.2
1.6967	4.49	30	144.4	1.1550	12.40	3	0.2
1.7040	4.42	40	194.3	1.2400	10.75	4	0.2
1.7094	4.37	50	242.0	1.3250	9.26	4	0.2
1.7169	4.30	60	291.1	1.4100	7.93	4	0.3
1.7243	4.23	70	347.4	1.4950	6.75	248	12.1
1.7289	4.19	80	395	1.5800	5.71	755	38.5
1.7316	4.17	90	446.4	1.6650	4.80	1573	81.1
1.7590	3.93	100	500.3	1.7500	4.01	2497	134.8

7.2.2 Výsledky testů

U testovaných algoritmů jsme zaznamenávali počet iterací, čas v sekundách, dosažené řešení vyjádřené poloměrem nejmenšího elipsoidu (viz popis řešení Andersonovy úlohy) a tomu odpovídající maximální chybu klasifikace v procentech. Vyjádření rychlosti výpočtu v sekundách je samozřejmě vztaženo na použitý počítač. Všechny popsané testy jsme provedly na stroji s proceserem PENTIUM P75MHz a pamětí 24MB RAM.

První tři algoritmy hledající optimální řešení jsme nechali běžet přibližně stejný čas, abychom mohli porovnat jak se blíží k řešení. Algoritmus ε-řešení, hledající řešení s předem zadanou chybou, jsme testovali v rozsahu hodnot získaných od první skupiny algoritmů, opět abychom mohli porovnat výsledky. Jinak řečeno, u první skupiny je nezávislá proměnná délka běhu algoritmu a u ε-řešení je to požadovaná chyba.

Testovací programy jsou implementovány v souborech *atest.m, btest.m, ctest.m* a *dtest.m*, kde první písmenko označuje testovaný algoritmus, jak jsme jej označili v tabulce 7.5.

Tabulka 7.9 Test algoritmu *gganders.m* a *ganders.m* na souboru *wineqn2.mat*: dimenze N=104, počet normálních rozdělení $K_1 = 20$ a $K_2 = 20$.

gganders.m				ganders.m			
min r	chyba [%]	iterace	čas [s]	min r	chyba [%]	iterace	čas [s]
0.2381	40.59	1	56.5	0.2381	40.59	1	58.4
0.2060	41.84	500	255.9	0.3291	37.11	4	94.3
0.2073	41.79	1000	455.3				
0.2080	41.76	1500	654.4				
0.2085	41.74	2000	852.4				
0.2090	41.72	2500	1051.8				
0.2093	41.71	3000	1250.5				
0.2096	41.70	3500	1448.4				
0.2098	41.69	4000	1648.4				
0.2101	41.68	4500	1847.2				
0.2102	41.67	5000	2044.8				

Tabulka 7.10 Test algoritmu *ganders2.m* a *eanders.m* na souboru *wineqn2.mat*: dimenze N=104, počet normálních rozdělení $K_1 = 20$ a $K_2 = 20$.

ganders2.m				eanders.m			
min r	chyba [%]	iterace	čas [s]	min r	chyba [%]	iterace	čas [s]
0.2381	40.59	1	55.3	0.2300	40.90	3	1.48
1.4139	7.87	10	677.7	0.6320	26.37	1350	316.4
1.9347	2.65	20	1688.9	1.0340	15.06	8818	975.1
2.1130	1.73	30	2729.5	1.4360	7.55	-	$>$ 9 hod
2.1837	1.45	40	3646.4				
2.2419	1.25	50	4538.1				

7.2.3 Zhodnocení výsledků

Algoritmus *ganders.m*, naplňující obecnou kostru řešení podle doporučení v Knize, se bohužel na obou testovaných příkladech zastavil v druhém kroku, což znamená, že uvázl v nějakém špatně zvoleném zlepšujícím směru. Jeho vlastnosti by se dali do jisté míry zlepšit zvýšením počtu iterací při numerické optimalizaci kritéria (viz. odstavec 4.7.1odstavec).

Algoritmus *gganders.m* využívající nehladkou gradientní optimalizaci nedosáhl v obou případech nejlepších výsledků. Zlepšení by se pravděpodobně dosáhlo, pokud bychom experimentovali s koeficienty řady, která váží přírůstky zobecněných gradientů v každé iteraci (viz popis řešení).

Z trojice algoritmů, hledajících optimální řešení, se v obou případech nejvíce osvědčil algorimus *ganders2.m*, který naplňuje obecnou kostru řešení s vylepšeným hledáním zlepšujícího směru. Jak je vidět z výsledků, chyba řešení se monotónně snižovala s rostoucím počtem iterací. Rychlost řešení se v druhém příkladě (20 a 20 rozdělení) snížila zhruba 18 krát oproti prvnímu příkladu (1 a 2 rozdělení), nicméně v obou případech bylo dosaženo velice rozumného výsledku a dá se předpokládat, že pokud bychom algorimus neukončili, snižoval by dále chybu řešení.

Nakonec jsme testovali algoritmus *eanders.m*, hledající rozdělující nadrovinu pro předem zadanou chybu klasifikace, neboli ε-řešení. Chybu klasifikace jsme volili v rozsahu výsledků dosažených předchozím algoritmem *ganders2.m*, abychom porovnali jejich rychlost. V prvním případě, kdy byl vstupem malý počet normálních rozdělení, se ukázal algoritmus ε-řešení zhruba 4 krát rychlejší. V druhém příkladě, když jsme zvýšili počet rozdělení na 20 pro každou třídu, byl výsledek opačný. Zde byl algoritmus ε-řešení podstatně pomalejší než algoritmus *ganders2.m* a dokonce nebylo časově únosné dopočítat všechny hodnoty v tabulce.

Podle provedených testů se jeví jako nejlepší algoritmus *ganders2.m*, naplňující obecnou kostru pomocí vypšené metody pro hledání zlepšujícího směru. Podrobnější popis je v tabulce 7.5, kde je označen jako algoritmus B.

8. Závěr

Vytvořili jsme programový balík obsahující vybrané metody statistického rozpoznávání. Součástí balíku jsou kromě samotných algoritmů rozpoznávání také demonstrační programy, nástroje pro interaktivní vytváření dat a pomocné funkce pro práci s nimi.

Programový balík obsahuje všechny algoritmy popsané ve specifikaci zadání této diplomové práce (viz specifikace v kapitole 1). Kromě nich jsme implementovali algoritmy:

- Naplnění kostry algoritmu pro obecné řešení Andersonovy úlohy metodami popsanými v Knize a námi navrženým vylepšením jedné podúlohy.
- ε-optimální rozdělení konečných množin bodů pomocí modifikovaného Kozincova algoritmu.
- Optimální rozdělení konečných množin bodů pomocí Linear Supports Vector Machine.

Metody rozpoznávání obsažené v balíku tvoří tématické celky podle toho, jaký typ úlohy řeší. Pro každou tématickou skupinu jsme vytvořili demonstrační program, ve kterém lze porovnávat chování různých algoritmů při řešení konkrétní úlohy. Prostředí demonstračního programu umožňuje vizualizaci procesu řešení, jeho interaktivní ovládání a také vytváření syntetických vstupních dat. Programový balík dále obsahuje nástroje pro převod externích dat do jeho interního formátu.

Mezi metodami rozpoznávání jsme se nejvíce zaměřili na řešení zobecněné Andersonovy úlohy, zvláště pak na naplnění obecné kostry algoritmu popsané v odstavci 4.7.1. Přitom se nám podařilo navrhnout a implementovat vylepšení její podúlohy, ve které se hledá zlepšující směr řešení (viz odstavec 4.7.3).

Všechny implementované funkce jsme opatřili nápovědou podle konvencí zavedených v Matlabu, díky tomu lze používat matlabovské nástroje ulehčující uživateli práci s naším programovým balíkem. Text nápovědy je napsán v angličtině stejně jako komentáře ve zdrojovém kódu. Také pracujeme na samostatné dokumentaci v jazyku HTML, která ještě není dokončena.

Použitelnost programového balíku pro řešení praktických úloh jsme ověřovali ve dvou experimentech. V prvním experimentu se nám podařilo klasifikovat seismická data pomocí kvadratické rozhodovací strategie. Praktický význam objevené vlastnosti se v současné době zkoumá. Cílem druhého experimentu bylo otestovat implementované algoritmy řešící zobecněnou Andersonovu úlohu. Jako nejlepší se ukázala metoda hledající ε-řešení a především algoritmus naplňující obecnou kostru, který obsahuje námi vylepšenou metodu pro určování zlepšujícího směru.

Literatura

[1] T.W. Anderson and R.R. Bahadur. *Classification into two multivariate normal distributions with differrentia covariance matrices.* Annals Math. Stat., 33:420-431, june 1962.

[2] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, Great Britain, 3th edition, 1997.

[3] Christopher J.C. Burges. *A Tutorial on Support Vector Machines for Pattern Recognition.* Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

[4] Zdeněk Kotek, Vladimír Mařík, Václav Hlaváč, Josef Psutka, and Zdeněk Zdráhal. *Metody rozpoznávání a jejich aplikace.* ACADEMIA Praha, 1993.

[5] B.N. Kozinec. *Rekurentnyj algoritm razdelenia vypuklych obolocek dvuch mnozestev, in Russian (Recurent algorithm separating convex hulls of two sets).* Sovetskoje radio, Moskva, 1973.

[6] Vladimír Mařík, Štěpánková, Jiří Lažanský, and kolektiv. *Umělá inteligence I.* ACADEMIA Praha, 1993.

[7] F. Rosenblatt. *Principles of Neurodynamics: Perceptron and theory of Brain Mechanisms.* Spartan Books, Washington, D.C., 1962.

[8] Michail I. Schlesinger and Václav Hlaváč. *Deset přednášek z teorie statistického a strukturního rozpoznávání.* Vydavatelství ČVUT, Praha, Czech Republic, 1999.

[9] N.Z. Shor. *Metody minimizacii nedifferenciruemych funkcij i ich prilozenia (Methods of non-differentiable functions minimizations and their applications), in Russian.* Naukova Dumka, Kiev, Ukrajine, 1979.

[10] N.Z. Shor. *Nondifferentiable optimization and polynomial problems.* Kluwer Academic Publisher, Dordrecht, The Netherlands, 1998.

[11] V.N. Vapnik. *The nature of the statistical learning theory.* Springer-Verlag, 1995.