Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Odhad složitosti algoritmů pomocí strojového učení

2017

Tomáš Chlup

Vedoucí práce: Mgr. Petr Osička, Ph.D.

Studijní obor: Informatika, prezenční forma

Bibliografické údaje

Autor:	Tomáš Chlup
Název práce:	Odhad složitosti algoritmů pomocí strojového učení
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2017
Studijní obor:	Informatika, prezenční forma
Vedoucí práce:	Mgr. Petr Osička, Ph.D.
Počet stran:	52
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Tomáš Chlup
Title:	Estimation of algorithm complexity with the help of machine learning.
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2017
Study field:	Computer Science, full-time form
Supervisor:	Mgr. Petr Osička, Ph.D.
Page count:	52
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Cílem práce bylo určit zda, nebo jak přesně lze odhadovat složitost algoritmů pro konkrétní instance bez toho, abychom algoritmus spustili. Práce pojednává o úspěšnosti odhadování u určených algoritmů. Velká část práce je věnována také definováním vhodných vlastností vstupních instancí, pro co nejlepší odhadování složitosti.

Synopsis

The mail goal of thesis was to determine whether or how to accurately estimate the complexity of algorithms for a particular instance without running the algorithm itself. Thesis deals with the success of the estimation algorithm complexity of certain algorithms. Much of the work is also devoted to defining the suitable properties of input instances, for the best possible estimation of complexity.

Klíčová slova: časová složitost; algoritmus; binární matice; vlastnosti grafu

Keywords: time complexity; algorithm; binary matrix; graph properties

Děkuji vedoucímu práce Mgr. Petru Osičkovi, Ph. D, za cenné rady a odbornou pomoc při zpracovávání bakalářské práce

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam vět

1 Úvod

V informatice často nastává situace, kdy hledáme vhodný algoritmus pro řešení zadaného problému. Pro instanci problému nemusí existovat algoritmus, který by ji řešil v rozumném čase. Tato práce se zabývá problematikou zda, nebo jak přesně, lze určovat složitost algoritmu pro konkrétní vstup bez toho, abychom algoritmus spustili. Určit složitost pro konkrétní instanci nemusí být u některých algoritmů náročné. Například pokud by jsme chtěli určit složitost řadícího algoritmu Insertion Sort a znali počet inverzí vstupního pole, můžeme prohlásit, že výsledná složitost bude odpovídat počtu inverzí. V této práci se budeme zabývat problémy, u kterých přesně nevíme, co ovlivňuje jejich složitost. Pomocí strojového učení se pokusíme u konkrétních algoritmů sestavit regresivní model pro předpovídání složitosti.
Odhad složitosti algoritmu bez jeho spuštění má i praktické uplatnění, například výběr vhodného algoritmu pro konkrétní vstup. Příkladem může být problém setřídění velkého pole, u kterého budeme schopni v krátkém čase spočítat jeho klíčové vlastnosti. Na jejich základě potom můžeme z množiny řadících algoritmů vybrat algoritmus, u kterého budeme vědět (předpokládat), že dané pole setřídí nejrychleji. Další možností využití může být například upravení parametrů algoritmu pro konkrétní vstup.
V první kapitole této práce se seznámíme s testovanými algoritmy. Ve druhé kapitole si definujeme vhodné vlastnosti vstupů a v posledních kapitolách se zaměříme na generování vstupních dat a samotné testování.

2 Testované algoritmy

Všechny testované algoritmy se zabývají výpočtem podmnožiny formálních konceptů. Formální koncept popisuje množina objektů, které můžou mít atributy z množiny atributů. Vstupní data pro testované algoritmy obsahují tedy množinu objektů a jejich atributy. Jako množinu objektů si můžeme představit různá auta a jako množinu atributů jejich vlastnosti. Můžeme pak například zachytit zdali dané auto má automatickou klimatizaci nebo jiné vlastnosti. Definice v této kapitole byli převzaty z [2]

Vstupní množinu můžeme reprezentovat trojicí

$$\langle X, Y, I \rangle$$

kde X je množina objektů, Y množina atributů a I je binární relace o velikosti $|X| \times |Y|$ reprezentují vztah mezi objekty $x \in X$ a atributy $y \in Y$. Využijeme toho, že binární relace na množině X a Y lze reprezentovat maticí I tak, že $I_{xy} = 1$ právě když, $\langle x, y \rangle$ patří do relace I (množiny X a Y uvažujeme jako uspořádané). Dále budeme tedy hovořit o maticích, ale když to bude vhodné (např. stručnější), budeme pro I používat relační notaci, tedy $\langle x, y \rangle \in I$ místo $I_{x,y} = 1$. Tedy pokud $I_{x_1,y_1} = 1$ ($\langle x_1, y_1 \rangle \in I$) znamená to že, objet x_1 má atribut y_1. Trojici $\langle X, Y, I \rangle$ budeme nazývat formální kontext.

Příklad 1 (Příklad matice I)

$$I = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \end{array}$$

kde $X = \{x_1, x_2, x_3\}$, $Y = \{y_1, y_2, y_3\}$ a $I = \{\langle x_1, y_1 \rangle$,$\langle x_2, y_2 \rangle$,$\langle x_2, y_3 \rangle$,$\langle x_3, y_2 \rangle$,$\langle x_3, y_3 \rangle\}$.
Matici $I \subseteq X \times Y$ budeme nazývat formální kontext.

Výstupem testovaných algoritmů je množina formálních konceptů (formální definici si uvedeme později). Formální koncepty si můžeme představit jako maximální obdélníky v matici I obsahující samé jedničky. V matici z příkladu 1 by to byl jeden obdélník tvořený objektem x_1 a atributem y_1 a druhý obdélník tvořený objekty x_1, x_2 a atributy y_2, y_3.

Abychom se mohli začít věnovat samotným algoritmům, zavedeme si nejdříve dvě základní operace $\uparrow$ a $\downarrow$. Tyto operace použijeme pro definici formálního konceptu a také je budeme používat pro samotný výpočet.

Definice 2 (↑ a ↓)
Pro množiny $A \subseteq X$ a $B \subseteq Y$ definujeme ↑,↓ jako

$$A^{\uparrow} = \{y \in Y | \forall x \in A : \langle x, y \rangle \in I\},$$
$$B^{\downarrow} = \{x \in X | \forall y \in B : \langle x, y \rangle \in I\}.$$

Množinu A budeme nazývat jako extent, množinu B budeme nazývat jako intent.

Příklad 3 (Matice I a operace ↑,↓)

$$I = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{array} \begin{array}{c} \begin{array}{ccccc} y_1 & y_2 & y_3 & y_4 & y_5 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

$A = \{x_2, x_5\}$ a $A^{\uparrow} = \{y_2, y_3\}$ nebo $A = \{x_1, x_3, x_4, x_6\}$ a $A^{\uparrow} = \{y_2\}$.
$B = \{y_1, y_2, y_5\}$ a $B^{\downarrow} = \{x_3, x_4\}$ nebo $B = \{y_1, y_2\}$ a $B^{\downarrow} = \{x_3, x_4, x_6\}$.

Definice 4 (Formální koncept)
Dvojice $\langle A, B \rangle$ v kontextu $\langle X, Y, I \rangle$ kde $A \subseteq X$, $B \subseteq Y$ a platí $A^{\uparrow} = B$, $B^{\downarrow} = A$ se nazývá formální koncept v I.

2.1 Výpočet všech formálních konceptů

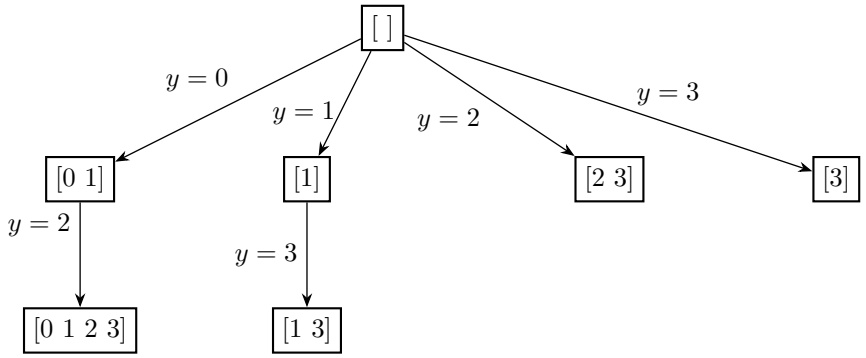
2.1.1 Algoritmus Close-by-one (CBO)

Rekurzivní algoritmus který vrací všechny formální koncepty pro vstupní formální kontext. Algoritmus byl převzat z [3].

CBO využívá faktu, že formální koncepty jsou jednoznačně definovány jejich intenty. Pokud B je intent, potom $\langle B^{\downarrow}, B \rangle$ je formální koncept. Dále algoritmus využívá toho, že B je intent nějakého formálního konceptu pokud je uzavřený. To znamená že B je intent právě tehdy, když $B = B^{\downarrow\uparrow}$. CBO počítá množinu všech intentů X jako $X = \{B \subseteq Y | B = B^{\downarrow\uparrow}\}$. Pro výpočet uzávěru využívá CBO subrutinu *Compute-closure(B, y)*, která přijímá množinu atributů B, atribut $y \notin B$ a vrací $(B \cup \{y\})^{\downarrow\uparrow}$.
CBO využívá podmínku $D \cap \{0, 1, ..., y-1\} = B \cap \{0, 1, ..., y-1\}$ (ř.11-16) která zajišťuje, aby se jeden uzávěr nepočítal vícekrát (pokud podmínka není splněna, nebudeme rekurzivně volat CBO na množinu D).
CBO začíná s $B = \emptyset$ a $y = y_1 \in Y$. Výstupem je $Int(X, Y, I)$ (množina intentů) z které můžeme sestavit množinu formálních konceptů jako $C(X, Y, I) = \{\langle B^{\downarrow}, B \rangle | B \in Int(X, Y, I)\}$.

Algorithm 1 Close-by-one

```
1: procedure CloseByOne(I, B, y)
2:     Ulož B
3:     if B = Y or y > n then
4:         return
5:     for j from y to n do
6:         if B[j] = 0 then
7:             set B[j] = 1
8:             set D = ComputeClosure(I, B, j)
9:             set skip = false
10:            for k from 0 upto j − 1 do
11:                if D[k] ≠ B[k] then
12:                    set skip = true
13:                    break for
14:            if skip = false then
15:                CloseByOne(I, D, j + 1)
16:            set B[j] = 0
```



Obrázek 1: Strom výpočtu k příkladu CBO

Příklad 5 (Příklad výpočtu algoritmu Close-by-one)
Vstupní matice bude I

$$I = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

Algoritmus voláme s $y = 0$ a $B = \emptyset$. Prvním uzávěrem je $\emptyset$. Algoritmus začíná cyklus na ř. 6-22 s $j = y = 0$, uzávěr atributu $\{0\}$ je vypočítán jako $\{0, 1\}$, atribut 0 do uzávěru patří - podmínka na ř.11-16 není splněna takže pokračuje rekurzivní volání Close-by-one s $y = 1$ a $B = \{0, 1\}$. V cyklu na ř. 6-22 začínáme s $j = y = 1$, ale náš uzávěr B již obsahuje atribut 1, takže pokračujeme s $j = 2$, uzávěr atributů $\{0, 1, 2\}$ je vypočítán jako celá Y ($\{0, 1, 2, 3\}$), podmínka na ř.11-16 není splněna, takže pokračujeme rekurzivním voláním, které ale hned končí protože $B = Y$. Pokračujeme s $j = 3$, uzávěr $\{0, 1, 3\}$ je vypočítán opět

jako Y, ale podmínka na ř.11-16 není splněna protože $2 \notin \{0,1,3\}$ a zároveň $2 \in [0,1,2,3]$, proto již rekurzivně nevoláme a tato větev algoritmu zároveň končí. Pokračujeme v původním volání s $j = 1$ a $B = \emptyset$, uzávěr $\{1\}$ je $\{1\}$, pokračujeme rekurzivním voláním s $y = 2$ a $B = \{1\}$. Cyklus na ř. 6-22 začíná s $j = 2$, uzávěr $\{1,2\}$ je Y, podmínka na ř.11-16 je splněna a proto pokračujeme dalším $j = 3$, uzávěr $\{1,3\}$ je $\{1,3\}$, podmínka je splněna, pokračujeme rekurzivním voláním s $y = 4$ a $B = \{1,3\}$ které hned končí protože $j = y = 4$, cyklus se přeskočí. Vracíme se opět do původního volání kde $j = 2$ a $B = \emptyset$, uzávěr $\{2\}$ je vypočítán jako $\{2,3\}$, pokračujeme rekurzivním voláním kde $y = 3$ a $B = \{2,3\}$, ale do množiny B už nemůžeme nic přidat takže větev končí. Pokračujeme opět v původním volání s $j = 3$ a $B = \emptyset$, uzávěr $\{3\}$ je $\{3\}$, rekurzivní volání hned končí, protože $j = y = 4$, ze stejného důvodu končí i původní volání a s ním celý algoritmus. Výsledné uzávěry jsou $\emptyset, \{0,1\}, \{0,1,2,3\}, \{1\}, \{1,3\}, \{2,3\}, \{3\}$.

2.1.2 Algoritmus Next-Closure (NC)

Algoritmus který stejně jako *CloseByOne* vrací všechny formální koncepty (intenty), ale v jiném uspořádání (lexikografickém). Nejprve definujeme relaci uspořádání $<$ pro dvě množiny atributů a operaci $\oplus$, kterou algoritmus využívá pro výpočet dalšího uzávěru. Algoritmus a definice byli převzaty z [4].

Definice 6 ($<_m$)
Pro $A, B \subseteq Y$ definujeme $<_m$ jako $A <_m B$ *když* $m \in B$ a $m \notin A$ a $(\forall n < m)\ n \in A$ právě když $n \in B$.

Definice 7 ($<$)
Pro $A, B \subseteq Y$ platí $A < B$ právě když $A <_m B$ pro nějaké $m \in Y$.

Definice 8 ($\oplus$)
Pro $A \subseteq Y$ a m_i, definujeme $\oplus$ jako

$$A \oplus m_i = ((A \cap \{m_1, m_2, ..., m_{i-1}\}) \cup \{m_i\})^{\downarrow\uparrow}$$

Algoritmus využívá faktu, že nejmenší větší uzávěr množiny $A \subset Y$ z lexikografického pohledu je $A \oplus m_i$ kde m_i je největší element M pro který je $A <_{m_i} A \oplus m_i$.
Vstup algoritmu *NextClosure* je formální kontext I výstupem je $Int(X, Y, I)$ (množina *intentů*) v lexikografickém uspořádání.

Příklad 9 (Příklad výpočtu algoritmu Next-closure)
Vstupní matice bude I

$$I = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

Algorithm 2 Next-closure

```
1: procedure NextClosure(I, B)
2:     A = ∅^{↓↑}
3:     while A ≠ H do
4:         ulož A
5:         A =NextClosureSub(I, A)
```

Algorithm 3 Next-closure-subrutine

```
1: procedure NextClosureSub(I, A)
2:     for all y ∈ Y v opačném pořadí do
3:         if y ∈ Y then
4:             A = A \ {y}
5:         else
6:             B = (A ∪ {y})^{↓↑}
7:             if B \ A neobsahuje žádný atribut < y then
8:                 return B
9:     return H
```

Algoritmus začíná voláním s $A = \emptyset$, cyklus začíná od posledním y, tedy $y = 3$, první uzávěr je vypočítán jako $\{3\}$, v dalším volání máme množinu $A = \{3\}$. Cyklus začíná opět s $y = 3$, ale první průchod jen odebere prvek 3 z uzávěru A a algoritmus pokračuje s $y = 2$ a $A = \emptyset$, další uzávěr je vypočítán jako $\{2, 3\}$. Pokračujeme voláním s $A = \{2, 3\}$, první dva průchody cyklu na ř.2-11(subrutine) odeberou prvky $2, 3$ z množiny A a pokračujeme dalším průchodem s $y = 1$, uzávěr je [1]. Další volání s $A = \{1\}$ vypočítá uzávěr $\{1, 3\}$ a dále algoritmus pokračuje voláním s $A = \{1, 3\}$, v prvním průchodu cyklu je odebrán prvek 3, v dalším je uzávěr vypočítán jako $\{0, 1, 2, 3\}$, není splněna podmínka na ř.7-9, pokračuje se s $y = 1$, odebere se atribut 1 z množiny A a pro $\emptyset$ je vypočítán uzávěr $\{0, 1\}$, pokračujeme voláním a $A = \{0, 1\}$. V prvním průchodu je vypočítán uzávěr $\{0, 1, 2, 3\}$ a není splněna podmínka ř.7-9, pokračujeme s $y = 2$, uzávěr je opět $\{0, 1, 2, 3\}$, ale podmínka je splněna, protože $y = 2$. Další volání s $A = \{0, 1, 2, 3\}$ končí vrácením H (halt) a algoritmus končí. Výsledné uzávěry jsou stejné jako u algoritmu *Close-by-one*, ale jsou lexikograficky uspořádány : $\{3\}, \{2, 3\}, \{1\}, \{1, 3\}, \{0, 1\}, \{0, 1, 2, 3\}$.

2.2 Rozklad booleovských matic

Další algoritmy se nezabývají výpočtem všech formálních konceptů, ale rozkladem matice I $(m \times n)$ na matice A o velikosti $n \times k$ a B o velikosti $k \times m$ tak aby jejich $A \circ B$ (produkt) byl původní matice I a zároveň bylo číslo k co nejmenší (tzn. matice A, B co nejmenší).

Definice 10 ($A \circ B$)
Produkt dvou booleovských matic A a B je definován jako

$$(A \circ B)_{i,j} = \bigvee_{l=1}^{k} A_{i,l} \wedge B_{l,j}$$

kde $\bigvee$ je maximum (také logická disjunkce) a $\wedge$ je klasické násobení (konjunkce)[2].

Definice 11 (Pokrytí)
Máme formální koncept $\mathcal{F}_1$ a kontext $\mathcal{I} = \langle X, Y, I \rangle$. Formální koncept $\mathcal{F}_1 = \langle \{x_n\}, \{y_n\} \rangle$ kde $x_n \in X$ a $y_n \in Y$ pokrývá jedničku ve formálním kontextu $\mathcal{I}$, pokud $\langle x_n, y_n \rangle \in I$ ($I_{x_n,y_n} = 1$).

Věta 12 (Existence pokrytí)
Pro formální kontext $\mathcal{I} = \langle X, Y, I \rangle$ existuje množina $\mathcal{F}$ která pokrývá všechny jedničky.

Důkaz (Existence pokrytí)
Uvedeme jak sestavit množinu formálních konceptů tak aby pokrývala všechny jedničky ve formálním kontextu. Máme $\mathcal{I} = \langle X, Y, I \rangle$, množinu formálních konceptů budeme sestavovat tak, že pro každou $\langle x, y \rangle \in I$ sestavíme formální koncept F_i tak, že $F_i = \langle \{x\}, \{y\} \rangle$, dále F_i přidáme do $\mathcal{F}$. $\mathcal{F}$ zřejmě pokrývá formální kontext $\mathcal{I}$. □

Definice 13 (Sestavení rozkladu matic z formálních konceptů)
Máme formální kontext $\mathcal{I} = \langle X, Y, I \rangle$ a množinu formálních konceptů $\mathcal{F}$ takovou, že formální koncepty z $\mathcal{F}$ pokrývají všechny jedničky ve formálním kontextu $\mathcal{I}$. Sestavíme matice A a B z formálních konceptů $F_i = \langle C_i, D_i \rangle \in \mathcal{F}$ tak, že $A_{ji} = 1$ pokud $j \in C_i$ jinak $A_{ji} = 0$ a $B_{ij} = 1$ pokud $j \in D_i$ jinak $B_{ij} = 0$. Potom $(A \circ B) = I$.

Příklad 14 (Formální koncepty I a produkt $A \circ B$)

$$I_1 = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{array} \begin{array}{c} \begin{array}{ccccc} y_1 & y_2 & y_3 & y_4 & y_5 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Formální koncepty budou $\langle A_1, B_1 \rangle = \langle \{x_3, x_4\}, \{y_1, y_2, y_5\} \rangle$, $\langle A_2, B_2 \rangle = \langle \{x_2, x_6\}, \{y_1, y_4\} \rangle$, $\langle A_3, B_3 \rangle = \langle \{x_2, x_5\}, \{y_3, y_4\} \rangle$, $\langle A_4, B_4 \rangle = \langle \{x_1, x_6\}, \{y_2\} \rangle$,
A množina formálních konceptů $\mathcal{F} = \{\langle A_1, B_1 \rangle, \langle A_2, B_2 \rangle, \langle A_3, B_3 \rangle, \langle A_4, B_4 \rangle\}$.

Vidíme že jednotlivé formální koncepty se můžou překrývat. Dále $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \circ \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

2.2.1 Algoritmus Grecond

Problém nalezení rozkladu matice je tedy problémem nalezení množiny jejích formálních konceptů. *Grecond* hledá co nejmenší počet formálních konceptů, začíná od velkých obdelníků, které pokrývají co nejvíce nepokrytých jedniček, až po obdelníky, které můžou obsahovat (pokrývat) jen jednu jedničku. Algoritmus ovšem koncepty nepočítá tak, že by nejdříve generoval všechny koncepty a pak hledal ten nejlepší. Nejvhodnější koncepty generuje postupně. Používá k tomu greedy techniku přidávání takového atributu, který aktuálně generovaný koncept nejvíce zvětší (maximalizuje počet nepokrytých jedniček které koncept pokrývá). Výsledek algoritmu nemusí být vždy optimální. Výsledný počet formálních konceptů může být větší než minimální možný, proto také výsledné matice A, B, sestavené z formálních konceptů, nemusí být nejmenší možné matice jejichž produkt dává výchozí matici I. Algoritmus byl převzat z [2].

Algorithm 4 Grecond

procedure GRECOND(I) ▷ (Booleovská matice)
  set $U = \{\langle i, j\rangle | I_{i,j} = 1\}$
  set $\mathcal{F} = \emptyset$
  while $U \neq \emptyset$ **do**
    set $D = \emptyset$
    set $V = 0$
    while existuje $j \notin D$ takové, že $^*|D \odot j| > V$ **do**
      vyber $j \notin D$ které maximalizuje $D \odot j$:
        set $D = (D \cup \{j\})^{\downarrow\uparrow}$
        set $V = |(D^{\downarrow} \times D) \cap U|$
    set $C = D^{\downarrow}$
    add $\langle C, D\rangle$ **to** $\mathcal{F}$
    for each $\langle i, j\rangle \in C \times D$ **do**
      odeber $\langle i, j\rangle$ **from** U
  return $\mathcal{F}$
    ▷ $^*D \odot j = ((D \cup \{j\})^{\downarrow} \times (D \cup \{j\})^{\downarrow\uparrow}) \cap U$

Příklad 15 (Příklad výpočtu algoritmu Grecond)
Vstupní matice bude I

$$I = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

Cyklus while na ř. 7-14 hledá atribut, který rozšíří aktuální formální koncept $\langle D, D^{\downarrow} \rangle$ tak, aby pokrývat co nejvíce jedniček z pomocné množiny U. Ta nám značí, které jedničky v matici ještě nejsou pokryté množinou formálních konceptů, kterou jsme doposud spočítali. V prvním průchodu cyklu na ř.7-14 kdy $D = \emptyset$ vybere atribut s největším V tj. atribut 2 kde $V = 4$ a přidá ho do D. Pokrytí množiny D již nelze zvětšit přidáním dalšího atributu, vytvoříme formální koncept $F_1 = \langle \{1, 2\}, \{2, 3\} \rangle$ a upravíme množinu U tak, aby neobsahovala jedničky, které tento formální koncept pokrývá. V dalším cyklu algoritmus vybere atribut 0 s $V = 2$, pokrytí množiny D opět nelze zvětšit, vytvoříme formální koncept $F_2 = \langle [0], [0, 1] \rangle$. V dalším průchodu přidáme do D atribut 1 s $V = 2$, D nelze zvětšit, vytvoříme $F_3 = \langle [0, 2], [1] \rangle$. Máme pokryté všechny jedničky v matici I ($U = \emptyset$) a vracíme množinu formálních konceptů $\mathcal{F} = F_1, F_2, F_3$.

2.2.2 Algoritmus Asso

Další algoritmus, který se používá pro výpočet rozkladu matice. Oproti algoritmu Grecond, který má vstupní parametr jen formální kontext I, má algoritmus Asso navíc další parametry. Parametry jsou čísla τ, w_1, w_2 a k. Jejich funkci si upřesníme v popisu samotného algoritmu.

V popisu algoritmu budeme využívat označení pro i-tý sloupec matice I označení $col < I, i >$ a označení pro i-tý řádek matice I $row < I, i >$, obojí budeme brát jako vektor. Dále budeme označovat $< \vec{a}, \vec{b} >$ jako skalární součin vektoru $\vec{a}$ a $\vec{b}$. Algoritmus si v prvním kroku vytváří asociační matici A tak, že $A_{i,j} = 1$ pokud $< col(I, i), col(I, j) > / < col(I, i), col(I, i) >$ je větší než τ a pokud $< col(I, i) >$ obsahuje alespoň jednu jedničku, jinak $A_{i,j} = 0$.
V dalším kroku počítá faktory, dokud to jde, nebo už jich nemá k. Udržuje si dvě pomocné matice. Matici U, která se na začátku inicializuje na matici I a v každém průchodu cyklu se v ní aktualizují již pokryté jedničky na nuly. Matici E o stejné velikosti jako I, která se inicializuje na $E = \{\forall i, j | E_{i,j} = 0\}$ a v každém průchodu cyklu se v ní aktualizují nuly na jedničky na místech, kde nový faktor dává jedničku, ale v matici I je nula. Matice U nám zajišťuje abychom jednu chybu nepočítali vícekrát.
Výběr faktoru :
1. pro každý řádek $row < A, i >$ (množinu atributů) hledáme množinu objektů X. Nastavíme X na prázdnou množinu a pro každý objekt x spočítáme hodnotu $w = w_1 * (< row(A, i), row(U, x) >) - w_2 * N$ kde N je počet míst na kterých má $row < A, i >$ jedničku a oba vektory $row < E, x >$, $row < I, x >$ nulu (tj. kolik míst, kde je v matici I nula, dáme nově jedničku). Pokud je hodnota w nenulová,

přidáme objekt x do množiny X, součet hodnot w všech přidaných objektů do X označíme jako V.
2. nový faktor je tvořen množinou objektů X a řádkem matice A takovými, že hodnota V bude největší ze všech řádků matice A. Pokud je hodnota V nulová pro všechny řádky matice A, znamená to, že nelze nalézt nový faktor a algoritmus končí.
3. upravíme matici U a E podle nového faktoru ($[X, row(A, i)]$) tak, že do U přidáme nuly na všech řádcích $x \in X$, na kterých má $row < A, i >$ jedničku (zaznamenáme pokryté jedničky v I) a do matice E přidáme jedničky na řádcích $x \in X$, na kterých má $row < A, i >$ jedničku a $row < I, x >$ nulu (zaznamenáme nové chyby, místa kde má matice I nulu, ale náš faktor jedničku).
Parametr τ je koeficient, který zvolíme tak, aby asociační matice měla pro nás optimální počet jedniček. Parametr k udává maximální počet průchodů cyklem pro výběr nového faktoru (tzn. maximální počet faktorů) a parametry w_1, w_2 (hlavně jejich poměrem) udáváme v případě, že preferujeme více faktorů nebo chybu. Jak vidíme oproti algoritmu Grecond může vracet chybné faktory tak, že produkt matic A a B, sestavený z vrácených faktorů, nebude stejná matice jako I, ale bude mít některé jedničky navíc, nebo budou některé chybět.

Příklad 16 (Příklad výpočtu algoritmu Asso)
Vstupem algoritmu bude $I, \tau = 0.4, w_1 = 2, w_2 = 1, k = 10$.

$$I = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

Algoritmus nejprve vypočítá asociativní matici A.

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Poté sestaví matici $U = I$ a nulovou matici E. V průchodu cyklem *for* na ř.6-23 vypočítá hodnotu *value* pro každý formální koncept tvořený objekty z matice I a řádkem z matice A. Hodnoty v prvním průchodu budou pro jednotlivé řádky A_i matice A postupně, $\langle [0, 2], [0, 1, 3] \rangle$ ($[x \in X, A_0]$, $X =$ množina čísel řádků matice I) kde $value = 9$, $\langle [0, 2], [0, 1, 3] \rangle$ ($[x \in X, A_1]$) kde $value = 9$, $\langle [0, 1, 2], [2, 3] \rangle$ ($[x \in X, A_2]$) kde $value = 6$ a $\langle [0, 1, 2], [1, 3] \rangle$ ($[x \in X, A_3]$) kde $value = 6$. Jako první formální koncept je vybrán první s hodnotou 9, tj. $\langle [0, 2], [0, 1, 3] \rangle$. Algoritmus upraví pomocné matice U a E.

$$U = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} E = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

V dalším průchodu cyklu na ř.6-23 budou hodnoty pro jednotlivé řádky A_i postupně $\langle [], [0, 1, 3] \rangle$ kde $value = 0$, $\langle [], [0, 1, 3] \rangle$ kde $value = 0$, $\langle [1], [2, 3] \rangle$ kde

Algorithm 5 Asso

```
1: procedure Asso(I, τ, w_1, w_2, k)                                  ▷ I[m][n]
2:     set F
3:     set A = computeA
4:     set U = I
5:     set E[m][n] = {∀i, j|E_{i,j} = 0}
6:     for i from 0 to k do
7:         set values = ∅
8:         set potentionalFactors = ∅
9:         for all row(A, i) ∈ A do
10:            X = ∅
11:            set value = 0
12:            for all x ∈ objects do
13:                set N = computeN
14:                set w = w_1 * (< row(A, i), row(U, x) >) − w_2 * N
15:                if w > 0 then
16:                    add x to X
17:                    set value = value+w
18:            add value to values
19:            add [X, row(A, i)] to potentionalFactors
20:        set newFactor to [X, row(A, i)] with biggest value
21:        refresh U
22:        refresh E
23:        add newFactor to F
24:    return F
```

$value = 4$ a $\langle [1], [1,3] \rangle$ kde $value = 1$. Je vybrán formální koncept $\langle [1], [2,3] \rangle$ a opět se upraví U a E.

$$U = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} E = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

V dalším průchodu již nemáme žádnou nenulovou hodnotu *value* a celý algoritmus končí vrácením $\mathcal{F} = \{\langle [0,2], [0,1,3] \rangle, \langle [1], [2,3] \rangle\}$. Jak můžeme vidět, množina $\mathcal{F}$ není optimální množina formálních konceptů pro vstupní matici I, protože pokrývá navíc jednu jedničku, která v původní matici I není tj. $A_{\mathcal{F}} \circ B_{\mathcal{F}} \neq I$. Pokud bychom upravili vstupní parametr w_2 na $w_2 = 3$, výstupem algoritmu by byla $\mathcal{F} = \{\langle [0,2], [1,3] \rangle, \langle [1], [2,3] \rangle, \langle [0], [0,1,3] \rangle\}$ a $A_{\mathcal{F}} \circ B_{\mathcal{F}} = I$.

3 Vlastnosti binárních matic

Abychom mohli vytvořit smysluplná data pro strojové učení, je potřeba definovat vhodné vlastnosti na vstupních maticích. Vlastnosti matic můžeme rozdělit na vlastnosti samotné matice a vlastnosti grafu, který matice může reprezentuje.

3.1 Základní vlastnosti

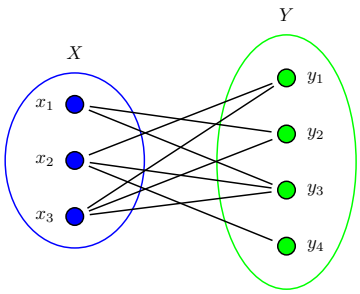
Na vstupní matici $I \subseteq X \times Y$ můžeme definovat základní vlastnosti, které závisí na velikosti samotné matice, jako je délka= $|X|$ a výška= $|Y|$. Další základní vlastností samotné matice je počet jedniček, který definujeme jako $\sum_{i=0}^{|X|}\sum_{j=0}^{|Y|} I_{i,j}$ a hustota jedniček v matici definovaná jako $\frac{počet jedniček}{|X|\times|Y|}$.

3.2 Matice jako bipartitní graf

Matice $I \subseteq X \times Y$ může reprezentovat také bipartitní, hranově neohodnocený a neorientovananý graf. Matice I reprezentuje graf $G_I = (V, E)$ tak, že $V = X + Y$ je množina vrcholů a $E = \{\{x, y\}|I_{x,y} = 1\}$ množina hran. Množina vrcholů tvořená objekty a množina vrcholů tvořená atributy může být vzájemně propojená. Žádné objekty ani atributy mezi sebou nemají hrany, a proto je graf vždy bipartitní.

Příklad 17 (Matice I a bipartitní graf G_I)

$$I = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$



Obrázek 2: Bipartitní graf G_I

Matici sousednosti bipartitního grafu $G_I = (V, E)$ definujeme jako $M_{G_I} \subseteq \mathcal{X} \times \mathcal{X}$ (kde $\mathcal{X} = \{x_1, ..., x_n, y_1, ...y_n\}$), kde $(M_{G_I})_{ij} = \{1 \ |\{i, j\} \in E\}$.

Příklad 18 (Matice sousednosti)
Matice sousednosti k grafu z předchozího příkladu

$$I = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{array} \begin{array}{c} \begin{array}{ccccccc} x_1 & x_2 & x_3 & y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

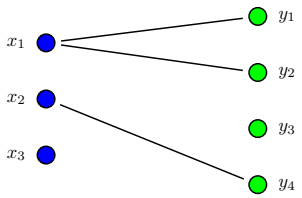
3.3 Vlasnosti bipartitního grafu

3.3.1 Počet komponent grafu

Definice 19 (Komponenta grafu)
Komponenta grafu G_{K_i} je maximální podgraf grafu G takový, že mezi všemi vrcholy obsaženými v podgrafu existuje cesta.

Příklad 20 (Komponenty grafu)

$$I = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$



Bipartitní graf G_I se skládá ze čtyř komponenty. $G_{K_1} = \{x_1, y_1, y_2\}$, $G_{K_2} = \{x_1, y_4\}$, $G_{K_3} = \{x_3\}$, $G_{K_4} = \{y_4\}$.

Jako algoritmus, který počítá počet komponent grafu G_I, budeme používat algoritmus, který prochází graf do šířky. Nejdříve si všechny uzly označí jako nedosažitelné. Následně prochází postupně všechny uzly a hledá uzly dosažitelné z aktuálního uzlu. Všechny uzly, do kterých vede cesta z aktuálního uzlu, poté označí jako dosažitelné a pokračuje dál.

3.3.2 Excentricita grafu

Definice 21 (Excentricita uzlu a grafu)
Excentricita vrcholu $v \in G$ je maximální vzdálenost z v do jiného vrcholu

Algorithm 6 Počet komponent grafu

```
1: procedure Počet komponent grafu(I)                    ▷ I[m][n]
2:     set D = ∅
3:     set G = graph(I)                                  ▷ G = (V, E)
4:     set početKomponent= 0
5:     for v in V do
6:         if D does contains v then
7:             add v to D
8:             početKomponent = početKomponent +1
9:             for each t in V do
10:                if t is reachable from v then
11:                    add t to D
12:    return početKomponent;
```

grafu $t \in G$. Značíme jako exc(v).
Excentricita grafu je průměrná excentricita jeho vrcholů.

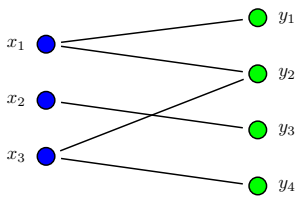
3.3.3 Diametr a rádius grafu

Definice 22 (Diametr grafu (průměr))
diam(G) je největší excentricita vrcholů grafu G

Definice 23 (Rádius grafu (poloměr))
rad(G) je nejmenší excentricita vrcholů grafu G

Příklad 24 (Excentricita, diametr a rádius grafu G_I)

$$I = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$



Excentricity jednotlivých uzlů grafu G_I : $exc(x_1) = 3$ (nejvzdálenější je y_4), $exc(x_2) = 2$ (nejvzdálenější je samotný x_2, protože matice sousednosti není reflexivní), $exc(x_3) = 3$, $exc(y_1) = 4$, $exc(y_2) = 2$,$exc(y_3) = 2$, $exc(y_4) = 4$.
Průměrná excentricita grafu G_I je 2.857, $diam(G_I) = 4$ a $rad(G_I) = 2$.

Algorithm 7 Excentricita uzlu

```
1: procedure EXCENTRICITA UZLU(G, x)                    ▷ G = (V, E), x ∈ V
2:     set D = ∅
3:     set N = ∅
4:     set excentricita = 0
5:     for each v ∈ V do
6:         if v je dosažitelné z x then
7:             D add v
8:         else
9:             N add v
10:    if D = ∅ then
11:        return excentricita
12:    excentricita = 1
13:    set temp = 1
14:    for k = 0 to |V| − 1 do
15:        if N ≠ ∅ then
16:            set newD = dosazitelne pres jednu hranu z jiz dosazitelnych
17:            if newD ≠ ∅ then
18:                e = e + temp
19:                temp = 1
20:                N remove all from newD
21:                D add all from newD
22:            else
23:                temp = temp+1
24:        else
25:            break
26:    return excentricita
```

3.3.4 Průměrný počet různých cest mezi uzly

Definice 25 (Cesta v grafu)
Cesta v grafu $G = (V, E)$ mezi dvěma vrcholy $x, y \in V$ je posloupnost hran $C = \{v_1, v_2\}, \{v_2, v_3\}..., \{v_{n-1}, v_n\}$ taková, že $v_1 = x$, $v_n = y$ a platí, že $\forall\{v_i, v_j\} \in C\ \exists\{v_i, v_j\} \in E_G$ a $v_i \neq v_j$ pro $i \neq j$.

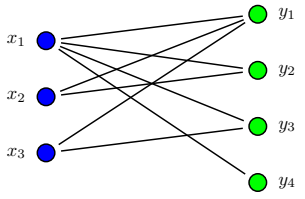
Počet různých cest mezi dvěma uzly $x, y \in V_{G_I}$ je počet cest C mezi x a y, které neobsahují stejné hrany.
Algoritmus, který počítá počet různých cest mezi dvěma vrcholy x, y, hledá cestu mezi x a y. Pokud nějakou najde, pokusí se najít další bez použití hran, které byli použity v předchozích nalezených cestách. Postup opakuje dokud existuje další cesta. Počet různých cest pro graf G_I je průměrný počet různých cest mezi všemi jeho vrcholy.

Algorithm 8 Počet cest

```
1: procedure Počet cest(G, x, y)                    ▷ G = (V, E), x, y ∈ V
2:     set G' = G
3:     set početCest = 0
4:     while true do
5:         C = najdiCestu(G',x,y)
6:         if C = ∅ then
7:             break
8:         početCest = početCest+1
9:         remove all e ∈ C from G'
10:    return početCest
```

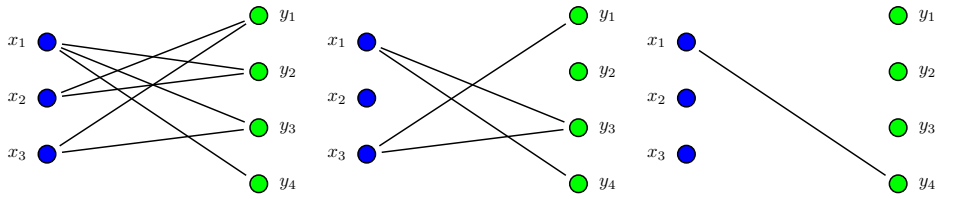
Příklad 26 (Počet cest mezi uzly)

$$I = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$



Algoritmus hledající cesty mezi uzly x_1, y_1 v grafu G_I jako první nalezne cestu C= $\{x_1, y_1\}$. Hrany které obsahuje cesta C smaže z grafu G'_1 a hledá další cestu v grafu G'. Další cestu vypočítá jako C= $\{x_1, y_2\}, \{y_2, x_2\}, \{x_2, y_1\}$ a hrany smaže. Další nalezená cesta bude C= $\{x_1, y_3\}, \{y_3, x_3\}, \{x_3, y_1\}$ a hrany opět smaže, při dalším průchodu ale cestu již nenajde a proto počet cest mezi x_1, y_1

v grafu G_I je roven 3. Graficky si postup algoritmu můžeme znázornit jako



3.3.5 Počet hran v kostře

Definice 27 (Kostra grafu)
Kostra grafu $G_I = (V, E)$ je podgraf $K = (V, E_K)$, který neobsahuje kružnice (neexistuje více než jedna cesta mezi dvěma vrcholy).

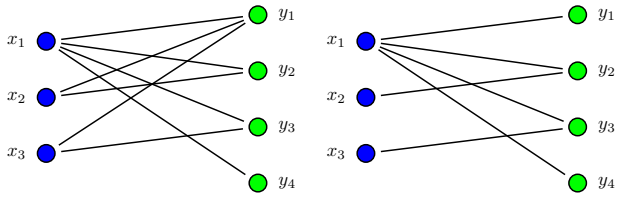
Počet hran v kostře je tedy počet hran v podgrafu K. Jelikož kostra bipartitních grafů reprezentovaných maticemi obecně nemusí obsahovat pouze jednu komponentu, budeme uvažovat vlastnost počet hran v kostře jako počet hran ve všech komponentách kostry grafu G_I. Algoritmus počítající počet hran v kostře si nejprve pro každý uzel vytvoří pomocnou strukturu, která bude reprezentovat komponentu do které uzel patří. Na začátku v této struktuře bude samotný uzel. Potom postupně prochází celý graf a spojuje komponenty podle toho, jestli mezi nimi existuje hrana.

Algorithm 9 Počet hran v kostře

```
1:  procedure Počet hran v kostře(G)          ▷ G = (V, E), V = {0, 1, ...}
2:      set početHran = 0
3:      set komponenta[|V|]
4:      for i=0 to |V| do
5:          set komponenta[i]= i
6:      for all v ∈ V do
7:          for all k ∈ V do
8:              if {v, k} ∈ E then
9:                  if komponenta[v] ≠ komponenta[k] then
10:                     aktualizuj komponenty které obsahují v nebo k
11:                     na komponenta[v] ∪ komponenta[k]
12:                     početHran = početHran+1
13:     return početHran
```

Příklad 28 (Počet hran v kostře)

$$I = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$



Obrázek 3: Graf G_I a jeho kostra K_{G_I}

I kdyby jsme vybrali jinou kostru grafu G_I, tak bude mít stejný počet hran. Jakákoli kostra grafu G_I má 6 hran.

3.3.6 GraphLinkEfficiency

Tato vlastnost je funkce přiřazující každému grafu G jeho efektivitu v intervalu $[0, 1]$. Čím více obsahuje graf hran, tím větší bude mít efektivitu. Naše bipartitní grafy nikdy nebudou mít efektivitu rovnu 1, protože nemůžou obsahovat všechny hrany mezi uzly.

Definice 29 (GraphLinkEfficiency)

$$E(G) = \frac{2}{n(n-1)} \sum_{i<j \in G} \frac{1}{dist(i,j)}$$

kde n je počet uzlů grafu G a $dist(i, j)$ je nejkratší možná vzdálenost mezi uzly $i, j \in G$ [5].

3.4 Další vlastnosti

Další vlastnosti binární matice I jsou vlastnosti grafu doplňkového k bipartitnímu grafu G_I který reprezentuje matice I.

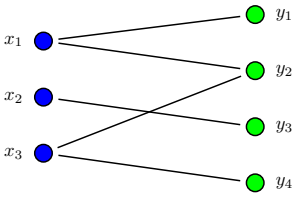
Definice 30 (Doplňkový graf)

Doplňkový graf pro bipartitní graf $G = (V, E)$, je graf $G_d = (V, E_d)$ takový, že $\{u, v\} \in E_d$ právě když $\{u, v\} \notin E$ a $u \neq v$.

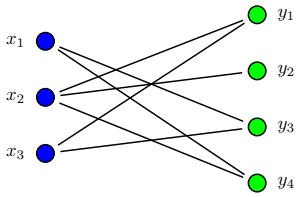
Příklad 31 (Doplňkový graf)

Matice I a bipartitní graf G_I který reprezentuje, jeho dopňkový graf G_{I_d} a matice I_d která ho reprezentuje.

$$I = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \end{array}, \; G_I \sim$$



$$I_d = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{array}, G_{I_d} \sim$$



Doplňkový graf k matici I budeme označovat jako D_I. Pro matici I máme tedy definovány její základní vlastnosti. Vlastnosti grafu, který tato matice reprezentuje, vlastností doplňkového grafu D_I (všechny vlastnosti které jsme měřili na grafu G_I můžeme aplikovat i na graf D_I).

4 Generování vstupních dat

V této kapitole si uvedeme možnosti generování a také jejich využití ve vytvořené testovací aplikaci. Jelikož vstupní data pro naše algoritmy jsou binární matice, budeme se zabývat vhodným generováním binárních matic. Musíme zajistit, aby vstupních instancí bylo dostatečné množství a také aby instance byli co nejvíce odlišné. V aplikaci je pro testování možné zadat počet vstupních instancí, které se budou generovat. Všechny generované instance (matice) mají náhodnou výšku a délku z intervalu který můžeme určit, oba parametry jsou náhodné a nezávislé, takže některé matice můžou být čtvercové, většina bude obdélníková. Dále jsou v aplikaci 3 možnosti plnění matic jedničkami, které ovlivňují jak bude výsledná matice vypadat z hlediska rozmístění jedniček v matici.

4.1 Základní generování

Základní způsob jak generovat binární matice (bipartitní grafy) je náhodný výběr délky a šířky matice v určitém intervalu. Následně na každé místo v matici zvolit náhodně jedničku nebo nulu, čímž bude šance na všech místech matice stejná. V našich datech bude délka i šířka v zadaném intervalu, šance, že bude v matici na daném místě jednička bude všude stejná, tedy například 0.5. V aplikaci je možné nastavit interval pro tuto šanci. Tento bude fungovat tak, že v první matici bude šance první hodnota z intervalu a šance se bude postupně zvyšovat podle počtu již vygenerovaných instancí tak, že u poslední instance bude šance poslední hodnota ze zadaného intervalu.

4.1.1 Markovův řetěz

V aplikaci jde také zapnout funkce Markovova řetězu, která výslednou matici ještě několikrát "zamíchá". Tato funkce tvoří ze vstupní matice $M_0 \subseteq X \times Y$ matice $M_1, M_2, ..., M_n$ kde $n = (|X| \times |Y|) \cdot log(|X| \times |Y|)$. Z matice M_i ($i \in \{0, 1, 2, ..., n-1\}$) vytvoří matici M_{i+1} tak, že náhodně vybere dvě políčka $M_{i_{q,j}}$ a $M_{i_{k,l}}$ kde $q \leq k$ a $j \leq l$, pokud $M_{i_{q,j}} = M_{i_{k,l}} = 1$ a zároveň $M_{i_{q,l}} = M_{i_{k,j}} = 0$ tak hodnoty políček prohodí.

4.2 Zvětšení šance pomocí lineární funkce

Další způsob generování má za cíl, aby uprostřed matice byla větší šance než na okraji matice. Šance se zvětšuje pomocí lineární funkce, která přiřazuje hodnoty z intervalu $[0, 0.5]$. Uprostřed matice je hodnota rovna 0.5, na okraji 0. V aplikaci je tato možnost realizována dvěma lineárními funkcemi f_1 a f_2. f_1, řídí se podle počtu řádků matice a f_2 podle počtu sloupců. Při plnění dat do matice je pro místo $I[i][j]$ vypočítána hodnota f_1 lineární funkce v závislosti na i a hodnota f_2 v závislosti na j, šance na jedničku na políčku $I[i][j]$ je rovna minimu z hodnot

$f_1(i), f_2(j)$. Funkce :

$$f_1(i) = |\frac{i - \frac{h}{2}}{h}| + 0.5$$

$$f_2(j) = |\frac{j - \frac{w}{2}}{w}| + 0.5$$

kde h je výška matice -1 a w je šířka matice -1. Funkce f přiřazující lineárně zvětšenou šanci pro políčko $I[i][j]$ je tedy $f(i,j) = min(f_1(i), f_2(j))$.

Příklad 32 (Lineárně zvětšená šance v matici)
Matice $I \subseteq X \times Y$ kde $|X| = 7$ a $|Y| = 9$. Čísla v následující matici reprezentují šanci na jedničku.

$$\begin{pmatrix}
0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\
0.000 & 0.125 & 0.166 & 0.166 & 0.166 & 0.166 & 0.166 & 0.125 & 0.000 \\
0.000 & 0.125 & 0.250 & 0.333 & 0.333 & 0.333 & 0.250 & 0.125 & 0.000 \\
0.000 & 0.125 & 0.250 & 0.375 & 0.500 & 0.375 & 0.250 & 0.125 & 0.000 \\
0.000 & 0.125 & 0.250 & 0.333 & 0.333 & 0.333 & 0.250 & 0.125 & 0.000 \\
0.000 & 0.125 & 0.166 & 0.166 & 0.166 & 0.166 & 0.166 & 0.125 & 0.000 \\
0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000
\end{pmatrix}$$

Dále tyto šance v aplikaci můžeme zvětšit zadáním intervalu, stejně jako u základního generování se stejnou šancí na všech místech. Interval se tentokrát nebude chovat jako samotná šance, ale hodnota se bude přičítat k vypočítané šanci pro konkrétní políčka. Pokud tedy zadáme interval $[0.1, 0.4]$, bude u první generované instance funkce přiřazovat hodnotu $f(i,j) + 0.1$, pro krajní políčko bude šance 0.1 a pro prostřední (pokud existuje) 0.6. U poslední instance bude přiřazovat hodnoty $f(i,j)+0.4$, u krajního políčka bude šance 0.4, u prostředního 0.9. Kdyby jsme v aplikaci zadali interval $[0.0, 0.5]$ a počet instancí by byl 100, první generovaná matice by měla pro políčka stejné šance jako matice z příkladu 25, poslední (100) by měla šance následující

$$\begin{pmatrix}
0.500 & 0.500 & 0.500 & 0.500 & 0.500 & 0.500 & 0.500 & 0.500 & 0.500 \\
0.500 & 0.625 & 0.666 & 0.666 & 0.666 & 0.666 & 0.666 & 0.625 & 0.500 \\
0.500 & 0.625 & 0.750 & 0.833 & 0.833 & 0.833 & 0.750 & 0.625 & 0.500 \\
0.500 & 0.625 & 0.750 & 0.875 & 1.000 & 0.875 & 0.750 & 0.625 & 0.500 \\
0.500 & 0.625 & 0.750 & 0.833 & 0.833 & 0.833 & 0.750 & 0.625 & 0.500 \\
0.500 & 0.625 & 0.666 & 0.666 & 0.666 & 0.666 & 0.666 & 0.625 & 0.500 \\
0.500 & 0.500 & 0.500 & 0.500 & 0.500 & 0.500 & 0.500 & 0.500 & 0.500
\end{pmatrix}$$

4.3 Zvětšení šance pomocí bivariantního normálního rozdělení

Stejně jako u zvětšení šance pomocí lineární funkce, bude tato možnost generovaní zvětšovat šanci na jedničku uprostřed matice. Šanci bude distribuovat v matici za pomoci dvourozměrného normálního rozdělení. Dvourozměrné normální

rozdělení přiřazuje hodnoty do intervalu $[0, 1]$, součet všech šancí je vždy roven jedné, tzn. při větší matici je šance pro většinu políček nulová. Pokud potřebujeme aby šance v matici nebyli zanedbatelné, zvolíme si maximální šanci max, kterou v matici chceme (bude uprostřed matice). Výslednou hodnotu funkce normálního rozdělení pro políčko v matici $I[i, j]$ vynásobíme $\frac{max}{a}$, kde a je maximální šance, kterou přiřazuje normální rozdělení pro konkrétní matici. Funkce f_d pro $I \subseteq X \times Y$ a políčko $I[i][j]$ přiřazuje hodnotu

$$f_d(i, j) = \frac{max}{a} \cdot \frac{1}{2\pi\sigma_1\sigma_2} \exp(-\frac{1}{2} \cdot ((\frac{i - u_1}{\sigma_1})^2 + (\frac{j - u_2}{\sigma_2})^2))$$

kde $u_1 = |X|-1$, $u_2 = |Y|-1$, $a = \frac{1}{2\pi\sigma_1\sigma_2}$ a parametry max, σ_1, σ_2 můžeme zadat. max udává maximální možnou šanci v matici, σ_1 a σ_2 udávají rozptyl šancí v matici. V aplikaci se σ_1 a σ_2 zadává v poměru k výšce a šířce matice. max se dá opět určit intervalem tak, že u první generované matice bude max první hodnota z intervalu. Podle počtu instancí se bude hodnota max inkrementovat a u poslední instance bude max poslední hodnota ze zadaného intervalu.

Příklad 33 (Matice s bivariatním normálním rozdělením šance)

Matice $I \subseteq X \times Y$ kde $|X| = 7$ a $|Y| = 7$. Parametry pro bivariantí rozdělelení jsou v prvním případě $max = 0.5$, $\frac{\sigma_1}{výšce} = 0.2$, $\frac{\sigma_2}{šířce} = 0.2$, u druhé matice $max = 0.5$, $\frac{\sigma_1}{výšce} = 0.45$, $\frac{\sigma_2}{šířce} = 0.45$. (Čísla v následující maticích reprezentují šanci na jedničku)

$$\begin{pmatrix} 0.005 & 0.018 & 0.039 & 0.050 & 0.039 & 0.018 & 0.005 \\ 0.018 & 0.064 & 0.139 & 0.180 & 0.139 & 0.064 & 0.018 \\ 0.039 & 0.139 & 0.300 & 0.387 & 0.300 & 0.139 & 0.039 \\ 0.050 & 0.180 & 0.387 & 0.5 & 0.387 & 0.180 & 0.050 \\ 0.039 & 0.139 & 0.300 & 0.387 & 0.300 & 0.139 & 0.039 \\ 0.018 & 0.064 & 0.139 & 0.180 & 0.139 & 0.064 & 0.018 \\ 0.005 & 0.018 & 0.039 & 0.050 & 0.039 & 0.018 & 0.005 \end{pmatrix}$$

$$\begin{pmatrix} 0.201 & 0.259 & 0.302 & 0.317 & 0.302 & 0.259 & 0.201 \\ 0.259 & 0.334 & 0.388 & 0.408 & 0.388 & 0.334 & 0.259 \\ 0.302 & 0.388 & 0.452 & 0.475 & 0.452 & 0.388 & 0.302 \\ 0.317 & 0.408 & 0.475 & 0.5 & 0.475 & 0.408 & 0.317 \\ 0.302 & 0.388 & 0.452 & 0.475 & 0.452 & 0.388 & 0.302 \\ 0.259 & 0.334 & 0.388 & 0.408 & 0.388 & 0.334 & 0.259 \\ 0.201 & 0.259 & 0.302 & 0.317 & 0.302 & 0.259 & 0.201 \end{pmatrix}$$

5 Klasifikace

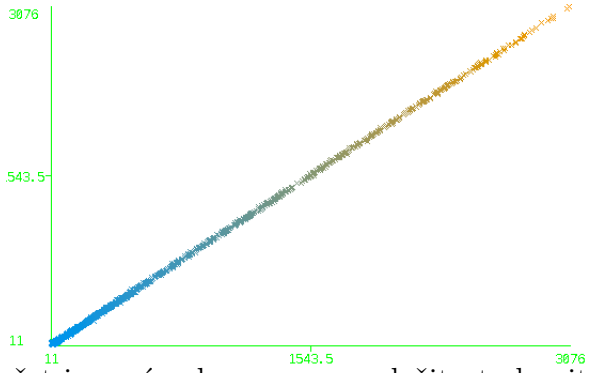
Chceme vytvořit model z vstupních dat a poté zjistit zda, nebo jak přesně, jsme schopni určit hodnotu požadovaného atributu matice na základě ostatních vlastností. Po vytvoření modelu z dat budeme sledovat jak přesně je vytvořený model schopný predikovat složitost na základě ostatních atributů. Množina atributů (vlastností) náhodně vygenerované matice je instance. Pro vytvoření modelu je potřeba vybrat vhodné klasifikátory tj. v našem případě vhodné algoritmy, které implementují strojové učení s učitelem (tj. strojové učení při kterém je známa trénovací množina se správně určenými kategoriemi pro všechny atributy). Jelikož všechny uvedené vlastnosti matic jsou číselné hodnoty, musíme zvolit klasifikátory které pracují s diskrétními hodnotami.

5.1 Křížová validace

Máme vstupní množinu instancí. Křížová validace pracuje na principu rozdělení vstupní množiny na dvě podmnožiny. První je trénovací množina, na které vytvoříme model a druhá je testovací množina, na které se vytvořený model otestuje a zjistíme odchylku od správných hodnot. Tento postup se několikrát opakuje s jiným rozdělením vstupní množiny a výsledná odchylka bude průměrná odchylka všech modelů. Například budeme vstupní množinu obsahující 1000 instancí rozdělovat na dvě podmnožiny tak, že z každých deseti instancí dáme první instanci do testovací množiny a zbytek instancí do trénovací množiny. Výsledkem bude trénovací množina obsahující 900 instancí a testovací obsahující 100. Tento postup budeme opakovat desetkrát, pokaždé s jiným rozdělením prvků.

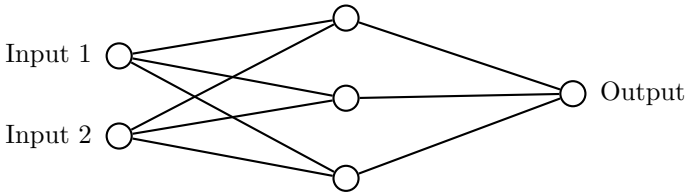
5.2 Lineární regrese

Je proložení bodů v grafu přímkou. V našem případě budeme mít pro instance s n atributy n-rozměrný prostor, ve kterém budou jednotlivé instance body v tomto prostoru. Lineární regrese má potom za cíl aproximovat funkci definovanou těmito body v grafu přímkou v $n$rozměrném prostoru.
Cíl je tedy aproximovat funkci $f(X)$ kde X je vektor $\langle x_1, x_2, ..., x_n \rangle$ a x_i je atribut, pomocí lineární funkce $f_X(X) = (\sum_{i=1}^{n} x_i c_i) + c_{n+1}$. Kde $c_1, ...c_{n+1}$ jsou koeficienty. V grafu závislosti počtu inverzí a složitosti pro algoritmus Insertion Sort by funkci $F(X)$, která přiřazuje složitost algoritmu pro pole definované vektorem, který obsahuje jeden atribut - počet inverzí, odpovídala funkce $F_X(X) = x$ kde x je počet inverzí.



Na ose x máme počet inverzí pole a na ose y složitost algoritmu Insertion Sort. Čísla v grafu udávají minimální, střední a maximální hodnotu daného atributu. Je zřejmé, že u algoritmu Insertion Sort je počet inverzí pole a počet porovnání přímo úměrný. V grafu je vyobrazeno 1000 instancí.

Obrázek 4: Graf Insertion Sort



Obrázek 5: MLP

5.3 Vícevrstvá neuronová síť

Anglicky Multi Layer Perceptron (MLP). Perceptron (neuron) je binární klasifikátor, který mapuje vstup x na hodnotu $f(x)$. Vstupem neuronu může být vstupní vektor (hodnota) a také výstup z jiného neuronu. MLP je neuronová síť složená z více vrstev neuronů. Každý neuron je spojen se všemi neurony v sousedních vrstvách. Mezi neurony v jedné vrstvě nejsou žádné vazby.

5.4 SMO

Sequential minimal optimization (SMO) je optimalizační metoda pro metodu strojového učení s učitelem Support vector machines (SVM). Základem metody SVM je jádrová transformace prostoru příznaků dat do prostoru transformovaných příznaků vyšší dimenze. SVM se poté snaží rozdělit tento prostor tak, aby množiny trénovacích dat různých tříd (různé hodnoty vyšetřovaného atributu) leželi v opačných poloprostorech tzn. prostor rozdělí rovinami podle různých tříd [6].

5.5 Weka

Weka [1] je software, který obsahuje sbírku algoritmů pro strojové učení, včetně všech uvedených. Tento software byl využit pro analýzu vytvořených dat. Ve vytvořené aplikaci jsou pro klasifikaci použity knihovny tohoto softwaru. Weka využívá pro datové soubory Attribute Relationship File Format (.arff), tj. textový soubor, který obsahuje na začátku informace o datech a poté samotná data. Informace o datech sestávají z popisu jednotlivých atributů, jejich jména a formátu. Data jsou jednotlivé instance. Na každém řádku je n-tice představující jednu instanci (n je počet atributů).

6 Experimenty

V každém testu bylo vygenerováno 10000 instancí (matic). Délka a výška matic je náhodně vybrána z intervalu $[5, 30]$. Využité jsou všechny tři metody generování matic, tzn. máme tři testy. V testu č.1 je při generování matic na všech místech stejná šance na jedničku (0.5). U testu č.2 je lineárně zvětšená šance na jedničku s parametry 0.2 a 0.3. U testu č.3 je šance v matici rozdělena s pomocí normální distribuce s parametry $\frac{x}{\sigma_1} = 0.5$, $\frac{y}{\sigma_2}$ a maximální šancí na jedničku 0.5, x je výška matice a y je šířka matice.
Délka a výška matice se vybírá nezávisle na sobě, matice tedy můžou mít jak čtvercový tvar, tak obdélníkový. Výsledky testů jsou znázorněny tabulkou, která má čtyři sloupce. V prvním sloupci máme použité klasifikátory, ve druhém sloupci korelační koeficient, který vyjadřuje závislost vyšetřovaného atributu a ostatních atributů použitých k jeho predikci. Ve třetím sloupci je průměrná absolutní odchylka (Mean absolute error) a ve čtvrtém sloupci je průměrná relativní odchylka (Relative absolute error), která udává průměrnou odchylku v procentech.

V druhém experimentu se budeme zabývat schopností predikce složitosti v závislosti na délce vstupních matic a počtem jedniček v matici. Bude využita pouze první metoda generování matic, tedy na všech místech stejná šance na jedničku. Velikost matic je rozdělena do intervalů $[7, 15]$, $[15, 23]$, $[23, 30]$. Šance na jedničku bude pro všechny velikosti matic postupně $0.25, 0.5, 0.75$. To také znamená, že hustota jedniček v matici bude (pro jednotlivé šance) okolo hodnota $0.25, 0.5$ a 0.75. Pro každý klasifikátor budeme mít jednu tabulku. V prvním sloupci tabulky bude šance na jedničku (hustota), která bude společná pro tři řádky tabulky. Tabulka bude rozdělená na sloupce po uvedených intervalech velikosti vstupních matic. Pro každou dvojici hustota a interval velikosti budou v tabulce tři hodnoty. První bude korelační koeficient, druhá bude průměrná odchylka a třetí průměrná relativní odchylka. Každá tabulka bude tedy obsahovat 27 hodnot.

6.1 Grecond

U algoritmu Grecond se budeme zabýval predikcí jeho složitost na základě vlastností matice, pro kterou ho chceme použít. Složitostí nemyslíme asymptotickou složitost, ale přesný počet průchodů nejhlubšího cyklu tohoto algoritmu. U tohoto algoritmu je to počet průchodů cyklem pro výběr atributu j maximalizujícího pokrytí D, protože algoritmus musí zkusit rozšířit dosavadní D všemi $j \notin D$, aby zjistil který atribut pokrytí množiny D maximalizuje.

6.1.1 První experiment

Tabulka 1: Test č.1 pro Grecond

	Corr. coef.	Mean a.e.	Relative a.e. (%)
Linear Regression	0.9843	117.1754	16.6158
Multilayer Perceptron	0.996	57.782	8.1937
SMOreg	0.9839	115.6557	16.4003

Tabulka 2: Test č.2 pro Grecond

	Corr. coef.	Mean a.e.	Relative a.e. (%)
Linear Regression	0.9805	112.9224	18.3098
Multilayer Perceptron	0.9936	65.2347	10.5775
SMOreg	0.9801	111.434	18.0685

Tabulka 3: Test č.3 pro Grecond

	Corr. coef.	Mean a.e.	Relative a.e (%)
Linear Regression	0.9766	123.7632	19.8715
Multilayer Perceptron	0.9922	72.986	11.7187
SMOreg	0.976	121.906	19.5733

Z testů je zřejmé, že nejlepších výsledků bylo dosaženo při generování matic se stejnou šancí na všech místech. Nejlepších výsledků dosahovala neuronová síť (MLP) a to u všech testů. Uspokojivého výsledku dosáhly všechny klasifikátory, ve všech testech měli korelační koeficient blížící se k 1.
Pro představu bylo rozmezí pro výslednou složitost cca 15-4000 s průměrnou hodnotou cca 850. Můžeme si také uvést jak vypadala funkce $f_X(X)$, která přiřazuje složitost, vytvořená Lineární regresí u testu č.1.

$f_X(X) = 19.6723 \cdot$ Výška matice +
$55.781 \cdot$ Šířka matice +
$11.3431 \cdot$ Počet jedniček v matici+
$-35.8664 \cdot$ Diametr grafu +
$-93.4161 \cdot$ Průměrný počet různých cest v grafu +
$-30.9993 \cdot$ Počet hran v kostře grafu +
$-1498.646 \cdot$ GraphLinkEfficency pro graf +
$102.1905 \cdot$ Excentricita doplňkového grafu +
$-33.4201 \cdot$ Diametr doplňkového grafu +
$152.7261 \cdot$ Průměrný počet různých cest doplňkového grafu +

-45.3253 · Počet hran v kostře doplňkového grafu +
-2329.9134 · GraphLinkEfficency pro doplňkový graf +
2246.8541

Vidíme, že některé vlastnosti, jako počet komponent nebo rádius grafu, nebyli zohledněny, to znamená, že nebyli pro predikci vůbec potřeba. Naopak koeficienty u použitých vlastností napovídají, jakou mají tyto vlastnosti váhu. Pokud chceme posuzovat, která vlastnost se nejvíce podílí na růstu složitosti, musíme zohlednit nejen koeficient, ale i průměrnou hodnotu dané vlastnosti. Uvedeme střední hodnoty (E(X)) atributů použitých ve funkci $f_X(X)$

X	E(X)
Výška matice	17.504
Šířka matice	17.566
Počet jedniček v matici	153.544
Diametr grafu	3.857
Průměrný počet různých cest v grafu	6.311
Počet hran v kostře grafu	33.987
GraphLinkEfficency pro graf	0.571
Excentricita doplňkového grafu	3.275
Diametr doplňkového grafu	3.844
Průměrný počet různých cest doplňkového grafu	6.319
Počet hran v kostře doplňkového grafu	33.989
GraphLinkEfficency pro doplňkový graf	0.57

Vidíme tedy, že pokud se bude například zvětšovat výška matice, neovlivní to složitost algoritmu tolik, jako zvětšování šířky matice. Vidíme také, že ke složitosti přispívá počet jedniček v matici. Složitost se bude nejspíše zmenšovat při růstu hran v kostrách grafů (normálního a doplňkového). Můžeme zkusit opakovat experiment č.1 s odebráním některých atributů, které měli vliv na výslednou funkci. Odebereme tedy atributy Šířka matice, Počet jedniček v matici, Počet hran v kostře a Počet hran v kostře doplňkového grafu a experiment opakujeme (pouze pro lineární regresi).

Výsledek testu :
Korelační koeficient 0.9682
Průměrná absolutní odchylka 172.4724
Relativní absolutní odchylka 24.5249 %

Oproti původnímu (test č.1) je výsledek o něco horší. Uvedeme výslednou funkci $f_X(X)$ = -32.621 · Výška matice +
429.5726 · Excentricita grafu +
-143.0123 · Rádius grafu +
-26.4254 · Diametr grafu +
224.8951 · Průměrný počet různých cest v grafu +
-3963.9354 · GraphLinkEfficency pro graf +
431.492 · Excentricita doplňkového grafu +
-161.2333 · Rádius doplňkového grafu +
-46.349 · Diametr doplňkového grafu +
210.479 · Průměrný počet různých cest v doplňkovém grafu +
-4248.9701 · GraphLinkEfficency pro doplňkový graf +
1830.0421

Koeficienty u atributů se změnily a nově se do funkce zahrnuly vlastnosti excentricita grafu, rádius grafu a doplňkového grafu. Nebylo však dosaženo stejné přesnosti predikce jako u původního testu, u kterého byli k dispozici všechny vlastnosti.

6.1.2 Druhý experiment

Lineární regrese

Tabulka 4: Výsledek experimentu č.2 pro LR (Grecond)

Density	Attribute	Size interval [7,15]	[15,23]	[23,30]
0.25	Corr. coef.	0.9675	0.9528	0.9067
	Mean a.e.	22.0052	65.1228	150.4267
	Relative a.e.	23.7135	28.1539	39.6285
0.50	Corr. coef.	0.9746	0.9791	0.9725
	Mean a.e.	23.8916	53.6809	94.6918
	Relative a.e.	20.9025	19.1749	21.7716
0.75	Corr. coef.	0.9694	0.9698	0.9639
	Mean a.e.	21.7036	54.3768	94.7895
	Relative a.e.	23.4997	23.3346	25.5996

Z testů vyplývá, že největší přesnost je dosažena u středně velkých matic s šancí na jedničku 0.5. Naopak nejhorší přesnost byla pozorována u největších matic s nejmenší šancí na jedničku, jinak řečeno u velkých matic s řídkým pokrytím jedničkami (hustotou). Můžeme také pozorovat, že nejlepší přesnosti je dosaženo u všech velikostí při střední hustotě jedniček.

Multilayer perceptron

Tabulka 5: Výsledek experimentu č.2 pro MP (Grecond)

Density	Attribute	Size interval		
		[7,15]	[15,23]	[23,30]
0.25	Corr. coef.	0.9677	0.9525	0.8791
	Mean a.e.	22.9261	66.4372	173.12
	Relative a.e.	24.7059	28.7221	45.6069
0.50	Corr. coef.	0.9746	0.9791	0.9725
	Mean a.e.	23.5441	59.4924	102.0438
	Relative a.e.	20.5984	21.2508	23.4619
0.75	Corr. coef.	0.9695	0.9625	0.9579
	Mean a.e.	22.0423	61.4651	101.2937
	Relative a.e.	23.8664	26.3764	27.3562

Nejhorší přesnost byla pozorována opět u největších matic s nejmenší hustotou. Nejlepší přesnost byla u nejmenších matic. Stejně jako u lineární regrese je nejlepší přesnost pozorována u matic se střední hustotou.

SMOreg

Tabulka 6: Výsledek experimentu č.2 pro SMOreg (Grecond)

Density	Attribute	Size interval		
		[7,15]	[15,23]	[23,30]
0.25	Corr. coef.	0.9673	0.9527	0.9065
	Mean a.e.	21.7802	65.0418	150.3166
	Relative a.e.	23.4711	28.1189	39.5996
0.50	Corr. coef.	0.9744	0.9791	0.9725
	Mean a.e.	23.7507	53.6771	94.702
	Relative a.e.	20.7792	19.1735	21.7739
0.75	Corr. coef.	0.9691	0.9697	0.9638
	Mean a.e.	21.6019	54.3678	94.8253
	Relative a.e.	23.3895	23.3308	25.6093

Výsledky testů pro SMOreg jsou podobné jako u testů lineární regrese.

6.1.3 Odhad počtu formálních konceptů

U algoritmu Grecond se můžeme zabývat také předpovídáním počtu formálních konceptů, které se vypočítají pro vstupní matici. Následující tabulky mají stejný význam jako u prvního experimentu, pouze s tím rozdílem, že se nesnažíme předpovídat složitost, ale finální počet formálních konceptů. Průměrný počet formálních konceptů pro matice s náhodnou délkou a výškou v intervalu $[5, 25]$ je cca 14.7.

Tabulka 7: Test č.1 pro Grecond a počet formálních konceptů

	Corr. coef.	Mean a.e.	Relative a.e (%)
Linear Regression	0.9870	0.8491	15.3279
Multilayer Perceptron	0.9831	1.0235	18.4758

Tabulka 8: Test č.2 pro Grecond a počet formálních konceptů

	Corr. coef.	Mean a.e.	Relative a.e (%)
Linear Regression	0.9865	0.8852	15.3889
Multilayer Perceptron	0.9827	1.0224	17.7746

Tabulka 9: Test č.3 pro Grecond a počet formálních konceptů

	Corr. coef.	Mean a.e.	Relative a.e (%)
Linear Regression	0.9859	0.935	15.7581
Multilayer Perceptron	0.9826	1.0486	17.6731

Můžeme pozorovat, že při odhadu počtu formálních konceptů, které algoritmus Grecond vypočítá, je mírně lepší lineární regrese. Uvedeme si funkci $f_X(X)$, která přiřazuje počet formálních konceptů z testu č.1 :

$f_X(X)$ = -0.005 · Výška matice +
0.0112 · Šířka matice +
8.8674 · Hustota jedniček +
0.0314 · Počet jedniček v matici +
-0.1914 · Počet komponent grafu +
-0.3317 · Excentricita grafu +
0.0612 · Diametr grafu +
-0.0579 · Průměrný počet různých cest v grafu +
0.027 · Počet hran v kostře grafu +
3.4014 · GraphLinkEfficency pro graf+
0.0955 · Počet komponent doplňkového grafu +

1.9325 · Průměrný počet různých cest v doplňkovém grafu +
-0.0032 · Počet hran v kostře doplňkového grafu +
-1.9111 · GraphLinkEfficency pro doplňkový graf +
-4.0718

Hustota jedniček a GraphLinkEfficency jsou čísla v intervalu $[0, 1]$. Podle tohoto modelu má tedy největší váhu při určování výsledného počtu formálních konceptů, které Grecond vypočítá vlastnost, průměrný počet různých cest v doplňkovém grafu.

6.2 Asso

U algoritmu Asso se budeme zabývat složitostí ze stejného hlediska jako u algoritmu Grecond. Složitost bude počet průchodů nejhlubšího cyklu tj. cyklus na ř. 12-19 12.

6.2.1 První experiment

Tabulka 10: Test č.1 pro Asso

	Corr. coef.	Mean a.e.	Relative a.e (%)
Linear Regression	0.9611	1328.7574	27.6202
Multilayer Perceptron	0.9987	238.5086	4.9578
SMOreg	0.9730	710.1887	14.0629

Tabulka 11: Test č.2 pro Asso

	Corr. coef.	Mean a.e.	Relative a.e (%)
Linear Regression	0.9661	1221.6591	25.4601
Multilayer Perceptron	0.9993	149.4001	3.1136
SMOreg	0.9885	686.4549	14.3061

Tabulka 12: Test č.3 pro Asso

	Corr. coef.	Mean a.e.	Relative a.e (%)
Linear Regression	0.9631	1304.4995	26.8984
Multilayer Perceptron	0.9996	110.4662	2.2778
SMOreg	0.9874	722.3086	14.8938

Průměrná hodnota pro složitost je cca 6300. V experimentu se ukázalo, že nejlepší model opět vytvořila ve všech případech neuronová síť. Nejlepší model byl

vytvořen pro test č.3 na rozdíl od testu s algoritmem Grecond, kde byl nejlepší model vytvořen u testu č.1.

6.2.2 Druhý experiment

Lineární regrese

Tabulka 13: Výsledek experimentu č.2 pro LR (Asso)

Density	Attribute	Size interval		
		[7,15]	[15,23]	[23,30]
0.25	Corr. coef.	0.9835	0.9948	0.9985
	Mean a.e.	107.0424	174.3868	154.5936
	Relative a.e.	17.2921	9.5483	5.169
0.50	Corr. coef.	0.9884	0.9965	0.9984
	Mean a.e.	88.7732	131.6731	152.6379
	Relative a.e.	14.3943	7.2978	5.1146
0.75	Corr. coef.	0.9679	0.9582	0.9520
	Mean a.e.	117.5859	433.6846	779.0732
	Relative a.e.	23.4875	27.613	28.3054

Nejlepších výsledků je dosaženo při velkých maticích a malé hustotě jedniček. Při velké hustotě a velké matici bylo dosaženo naopak nejhoršího výsledku. Můžeme také pozorovat, že přesnost odhadu se snižuje se zvětšující se hustotou a to i u malých matic.

Multilayer perceptron

Tabulka 14: Výsledek experimentu č.2 pro MP (Asso)

Density	Attribute	Size interval		
		[7,15]	[15,23]	[23,30]
0.25	Corr. coef.	0.9902	0.9979	0.9997
	Mean a.e.	80.6291	94.0543	43.4944
	Relative a.e.	13.0252	5.1489	1.4543
0.50	Corr. coef.	0.9939	0.9971	0.9981
	Mean a.e.	60.2207	89.9486	139.4318
	Relative a.e.	9.7646	4.9853	4.6721
0.75	Corr. coef.	0.9732	0.9544	0.9423
	Mean a.e.	117.2022	443.1495	864.3666
	Relative a.e.	23.4108	28.2156	31.4043

Stejně jako v případě lineární regrese se přesnost snižuje se zvyšující se hustotou jedniček v matici. Nejlepší a nejhorší výsledek je také stejný jako u lineární regrese.

6.3 Close-by-one (CBO)

Algoritmus CBO vrací všechny formální koncepty. Budeme se zabývat tím, kolik vypočítá potenciálních formálních konceptů. To znamená, že budeme počítat kolikrát vypočítá uzávěr na ř.8. V prvním a druhém experimentu nebudeme tedy predikovat složitost jako v případě předchozích algoritmů, ale počet potenciálních formálních konceptů.

6.3.1 První experiment

Tabulka 15: Test č.1 pro Close-by-one

	Corr. coef.	Mean a.e.	Relative a.e (%)
Linear Regression	0.8681	1580.0443	49.5179
Multilayer Perceptron	0.9923	310.2745	9.7239
SMOreg	0.9471	872.5719	27.346

Tabulka 16: Test č.2 pro Close-by-one

	Corr. coef.	Mean a.e.	Relative a.e (%)
Linear Regression	0.8336	950.7676	54.9843
Multilayer Perceptron	0.9872	279.1317	16.1426
SMOreg	0.9499	446.9211	25.846

Tabulka 17: Test č.3 pro Close-by-one

	Corr. coef.	Mean a.e.	Relative a.e (%)
Linear Regression	0.8990	526.9225	43.7781
Multilayer Perceptron	0.9929	125.0903	10.3928
SMOreg	0.9656	279.3244	23.207

Nejlepší model vytvořila ve všech testech opět neuronová síť. Průměrný počet potenciálních konceptů se u testů lišil. V případě testu č.1 (na všech místech stejná šance) byla průměrná hodnota 2941, u testu č.2 1662 (lineárně zvětšená) a u testu č.3 (normální distribuce) 1242. Proto se odchylky tolik liší u jednotlivých testů. Lineární regrese měla ve všech testech zřetelně horší výsledky než ostatní klasifikátory.

6.3.2 Druhý experiment

Lineární regrese

Tabulka 18: Výsledek experimentu č.2 pro LR (CBO)

Density	Attribute	Size interval [7,15]	[15,23]	[23,30]
0.25	Corr. coef.	0.9608	0.9525	0.9475
	Mean a.e.	11.6257	53.0021	140.3916
	Relative a.e.	25.3714	28.5705	30.6789
0.50	Corr. coef.	0.9246	0.9446	0.9496
	Mean a.e.	45.1342	337.72	1430.216
	Relative a.e.	35.9113	31.074	29.5282
0.75	Corr. coef.	0.8478	0.8848	0.8983
	Mean a.e.	133.0925	2971.1402	32112.83
	Relative a.e.	50.5795	44.2237	41.5074

Můžeme pozorovat, že úspěšnost se snižuje se zvětšující se hustotou. Nejlepšího výsledku bylo dosaženo u malých matic s malou hustotou a nejhoršího u malých matic s velkou hustotou. Nejlepší výsledky byly dosaženy v testu č.3 tedy při normálním rozdělením šancí v matici.

Multilayer perceptron

Tabulka 19: Výsledek experimentu č.2 pro MP (CBO)

Density	Attribute	Size interval [7,15]	[15,23]	[23,30]
0.25	Corr. coef.	0.9533	0.9520	0.9467
	Mean a.e.	13.638	54.9328	144.8596
	Relative a.e.	29.7628	29.6112	31.6552
0.50	Corr. coef.	0.9558	0.9624	0.9587
	Mean a.e.	34.3612	278.8762	1336.3834
	Relative a.e.	27.3397	25.6597	27.5909
0.75	Corr. coef.	0.9160	0.9352	0.9280
	Mean a.e.	110.7144	2059.3262	26827.01
	Relative a.e.	42.075	30.6519	34.6753

Nejlepších výsledků dosáhla neuronová síť při střední velikosti matic. Lepších

výsledků než u lineární regrese bylo dosaženo hlavně u velké hustoty, jinak jsou výsledky obou klasifikátorů srovnatelné.

6.4 Next-closure

U algoritmu Next-closure se stejně jako u CBO zaměříme na počet vypočítaných potenciálních konceptů. U algoritmu Next-closure budeme tedy predikovat, kolikrát se provede výpočet uzávěru na ř.6.

6.4.1 První experiment

Tabulka 20: Test č.1 pro Next-closure

	Corr. coef.	Mean a.e.	Relative a.e (%)
Linear Regression	0.8736	1564.7739	49.1898
Multilayer Perceptron	0.9902	376.3834	11.8319
SMOreg	0.9504	860.955	27.0648

Tabulka 21: Test č.2 pro Next-closure

	Corr. coef.	Mean a.e.	Relative a.e (%)
Linear Regression	0.8334	952.1904	54.3398
Multilayer Perceptron	0.9733	391.1165	22.3203
SMOreg	0.95	446.6429	25.4891

Tabulka 22: Test č.3 pro Next-closure

	Corr. coef.	Mean a.e.	Relative a.e (%)
Linear Regression	0.9007	518.7201	43.3724
Multilayer Perceptron	0.9851	208.052	17.3961
SMOreg	0.9664	275.7846	23.0595

Ve všech testech vytvořila nejlepší model opět neuronová síť. Průměrné hodnoty potenciálních uzávěrů je pro test č.1 2900, pro test č.2 1683 a pro test č.3 1227. Nejlepší výsledky byly dosaženy opět v testu č.3.

6.4.2 Druhý experiment

Lineární regrese

Tabulka 23: Výsledek experimentu č.2 pro LR (Next-closure)

Density	Attribute	Size interval [7,15]	[15,23]	[23,30]
0.25	Corr. coef.	0.9602	0.9498	0.9465
	Mean a.e.	11.6915	54.236	143.2127
	Relative a.e.	27.9222	29.535	31.0399
0.50	Corr. coef.	0.926	0.902	0.9472
	Mean a.e.	46.5961	410.2244	1497.5294
	Relative a.e.	35.9822	43.172	30.6178
0.75	Corr. coef.	0.8443	0.8747	0.8941
	Mean a.e.	134.0988	0.2987	31947.3857
	Relative a.e.	50.9765	45.2741	41.5728

Stejně jako u CBO, můžeme pozorovat, že se úspěšnost snižuje se zvětšující se hustotou. Pro střední a velkou hustotu se úspěšnost zvětšuje při zvětšování instancí. Pro malou hustotu se přesnost při zvětšování matic naopak zmenšuje. Nejlepší výsledek je dosažen na malých maticích při malé hustotě stejně jako u CBO.

Multilayer perceptron

Tabulka 24: Výsledek experimentu č.2 pro MP (Next-closure)

Density	Attribute	Size interval [7,15]	[15,23]	[23,30]
0.25	Corr. coef.	0.9545	0.9567	0.9461
	Mean a.e.	13.5371	50.6777	142.4818
	Relative a.e.	29.7344	27.5973	30.8815
0.50	Corr. coef.	0.9601	0.9816	0.9591
	Mean a.e.	35.6284	174.4784	1324.572
	Relative a.e.	27.5128	17.4397	27.0816
0.75	Corr. coef.	0.9396	0.9401	0.9353
	Mean a.e.	79.9559	1932.525	24358.378
	Relative a.e.	30.3945	29.2847	31.6973

U všech velikostí matic je největší přesnost dosažena při střední hustotě. Nejlepšího výsledku bylo dosaženo při střední hustotě na středně velkých maticích. Výsledky jsou podobné jako pro CBO.

7 Popis rozhraní testovací aplikace

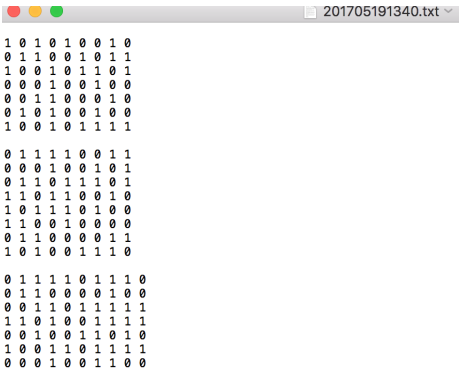
Pro účely testování byl vytvořen jednoduchý formulářový program v JavaFX. Program umožňuje generování dat (matic) a jejich analýzu. Úvodní okno programu obsahuje postupně tlačítka *Setting*, *I have matrices*, *I have arff file* a *Make everything*. Tlačítko *Setting* otevře okno s nastavením, ve kterém můžeme nastavit složku, do které se budou ukládat výsledky testů a také počet vláken, které aplikace bude využívat při výpočtech. Pokud v nastavení nevybereme vlastní složku, bude vytvořena a využívána složka data ve stejné lokaci, ze které je aplikace spuštěna.

7.1 Kompletní vygenerování dat

Pokud na úvodní obrazovce aplikace vybereme možnost Make everything, otevře se okno pro kompletní vygenerování dat a jejich klasifikaci. Jako první musíme vybrat algoritmus ze seznamu. Po výběru algoritmu se nám zpřístupní možnosti generování a klasifikace. V první části můžeme určit počet generovaných instancí, který musí být větší než 10 kvůli křížové validaci při klasifikaci. Také můžeme definovat minimální a maximálních velikost generovaných matic. Výšku a šířku matic můžeme vybrat nezávisle.
Další část nastavení je věnována metodám generování matic. První možností je generování matic se stejnou šancí na jedničku na každém místě matice. Jsou zde dva parametry, první udává počáteční šanci, druhý konečnou šanci (4.1). Další možností generování je lineárně zvětšená šance směrem do středu matice. Opět zde máme dva parametry, jejich význam je vysvětlen v kapitole 4.2. Třetí možností je distribuce šance v matici pomocí normálního rozdělení (4.3).
Poslední část nastavení je zaměřená na atributy instancí a výběr klasifikátoru. V prvním sloupci můžeme vybrat které vlastnosti se mají využít při vytváření modelu, v druhém sloupci můžeme vybrat jeden atribut, který chceme klasifikovat. Dále můžeme vybrat klasifikátory a generování spustit. Po spuštění testu vidíme průběh generování dat, průběh vytváření modelu není monitorován. U většiny testů trvá nejdéle vytvoření modelu pomocí SMOreg, hodně času také zabere generování dat, pokud se generují velké matice. Po skončení testu je výsledek vypsán do textového pole. Vygenerované matice a výsledek testu se ukládají do textového souboru, instance do souboru s koncovkou arff. Soubory jsou pojmenovány podle data vytvoření.

7.2 Vlastní matice

Pokud máme textový soubor s maticemi, které chceme vyhodnotit, vybereme na úvodní obrazovce možnost *I have matrices*. Otevře se okno kde musíme nejprve vybrat náš textový soubor s maticemi ze souborového systému a vybrat algoritmus který chceme testovat. Soubor s maticemi musí být textový soubor ve kterém jsou jednotlivé binární matice reprezentované jejich řádky a sloupci, prvky matice jsou odděleny mezerou a jednotlivé matice jsou od sebe odděleny



Obrázek 6: Vyžadovaný formát souboru s maticemi

```
@RELATION 201705171132

@ATTRIBUTE height       NUMERIC
@ATTRIBUTE width        NUMERIC
@ATTRIBUTE density      NUMERIC
@ATTRIBUTE numberOfT    NUMERIC
@ATTRIBUTE numberOfComponents   NUMERIC
@ATTRIBUTE excentricitaGrafu    NUMERIC
@ATTRIBUTE radiusGrafu  NUMERIC
@ATTRIBUTE diametrGrafu NUMERIC
@ATTRIBUTE prumernyPocetRuznychCest     NUMERIC
@ATTRIBUTE pocetHranVKostre     NUMERIC
@ATTRIBUTE graphLinkEfficency   NUMERIC
@ATTRIBUTE numberOfComponentsD  NUMERIC
@ATTRIBUTE excentricitaGrafuD   NUMERIC
@ATTRIBUTE radiusGrafuD NUMERIC
@ATTRIBUTE diametrGrafuD        NUMERIC
@ATTRIBUTE prumernyPocetRuznychCestD    NUMERIC
@ATTRIBUTE pocetHranVKostreD    NUMERIC
@ATTRIBUTE graphLinkEfficencyD  NUMERIC
@ATTRIBUTE finalLength  NUMERIC
@ATTRIBUTE complexity   NUMERIC
@DATA
21,17,0.2577030812324929 7,92.0,1,4.2894736842105265,3,5,3.7086834733893 56,37,0.46460407776197393,1,3.0,3,3,12.439775910364146,37,0.6666666666666659,20,697
16,20,0.23125,74.0,1,4.666666666666667,4,6,3.08125,35,0.4426455026455037,1,3.0,3,3,12.159375,35,0.6756613756613755,20,909
17,20,0.23529411764705882,80.0,1,4.297297297297297,4,5,3.411764705882353,36,0.45000000000000145,1,3.0,3,3,12.873529411764705,36,0.6751751751751747,24,1091
22,19,0.24401913875598086,102.0,1,4.170731707317073,3,5,3.944976076555024,40,0.4637601626016251 7,1,3.0,3,3,14.14354066985646,40,0.6719512195121943,22,868
```

Obrázek 7: Ukázka arff souboru

prázdným řádkem. Po vybrání algoritmu je vytvořen arff soubor a zobrazí se nám možnosti pro klasifikaci. Možnosti jsou stejné jako u poslední části nastavení při generování kompletních dat.

7.3 Arff soubor

Aplikace umožňuje načtení již hotového arff souboru s instancemi. Pokud tedy máme arff soubor, který chceme analyzovat, vybereme na úvodní obrazovce možnost *I have arff file*. Poté co vybereme tuto možnost, musíme vybrat arff soubor ze souborového systému. Po načtení souboru jsou vypsány atributy instancí v našem souboru a my můžeme vybrat ty, které se mají použít při klasifikaci, atribut pro klasifikaci a klasifikátory. Výsledek testu je uložen do adresáře s vybraným arff souborem.

Závěr

Cílem práce bylo určit zda, nebo jak přesně, lze odhadovat složitost algoritmů pro konkrétní instance bez toho, abychom algoritmus spustili. Ukázalo se, že u všech testovaných algoritmů jsme byli na základě definovaných vlastností schopni relativně přesně určovat jejich složitost pro konkrétní vstupy. Ve všech testech odhadu složitosti vytvořila nejlepší model neuronová síť. U algoritmů CBO a NC měl model vytvořený pomocí lineární regrese velice špatné výsledky oproti ostatním klasifikátorům. Při odhadu výsledného množství formálních konceptů, které algoritmus Grecond vypočítá, měla lineární regrese o něco lepší výsledky než neuronová síť.
Dalším rozšířením této práce by mohlo být přidání dalších způsobů generování testovacích dat, například definování šance na jedničku pro každý atribut zvlášť. Takto vygenerované matice by byly blíže reálným datům, kde může mít každý atribut jinou pravděpodobnost.

Conclusions

The mail goal of thesis was to determine whether or how to accurately estimate the complexity of algorithms for a particular instance without running the algorithm itself. It turned out that for all tested algorithms, we were able to accurately determine their complexity for specific inputs on the basis of the defined properties. The neural network builded the best model in all complexity testing tests. For CBO and NC algorithms, the model builded by linear regression had very poor results compared to other classifiers. When estimating the final amount of formal concepts that the Grecond algorithm calculates, the linear regression had a bit better results than the neural network.
Extension of this thesis could be to add other ways of generating test data, such as defining chances for each attribute separately. Such matrices would be closer to real data, where each attribute may have a different probability.

8 Spuštění testovací aplikace

Aplikace byla testována na Windows 8.1 a MacOS Sierra 10.12.4. Pro spuštění přiložené aplikace je nutné mít nainstalovaný Java Runtime Environment.

9 Obsah přiloženého CD/DVD

`bin/`
: Obsahuje testovací program, spustitelný přímo z CD.

`doc/`
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

`src/`
: Kompletní zdrojové texty programu Program se všemi potřebnými (příp. převzatými) zdrojovými texty, knihovnami a dalšími soubory potřebnými pro bezproblémové vytvoření spustitelných verzí programu / adresářové struktury pro zkopírování na webový server.

`readme.txt`
: Instrukce pro instalaci a spuštění programu Program, včetně všech požadavků pro jeho bezproblémový provoz.

Literatura

[1] WEKA, Data mining software, The University of Waikato [online].
Dostupné z: https://weka.wikispaces.com

[2] BĚLOHLÁVEK, Radim; VYCHODIL,Vilém
Discovery of optimal factors in binary data via a novel method of matrix decomposition. Copyright © [cit. 15.05.2017]. Dostupné
z: http://upcase.inf.upol.cz/download/journals/jcss/BeVychodil10_Dofbdnmmd.pdf

[3] VYCHODIL, Vilém. A New Algorithm for Computing Formal Concepts.
Copyright © [cit. 21.05.2017].
Dostupné z: http://phoebe.inf.upol.cz/h̃avrlanl/articles/Vychodil08_Anafcfc.pdf

[4] BĚLOHLÁVEK, Radim. Introduction to formal concepty analysis. [cit. 21.05.2017]. Dostupné z: https://phoenix.inf.upol.cz/esf/ucebni/formal.pdf

[5] Efficiency (network science) [cit. 21.05.2017].
Dostupné z: https://en.wikipedia.org/wiki/Efficiency_(network_science)

[6] Support vector machines [cit. 21.05.2017].
Dostupné z: https://cs.wikipedia.org/wiki/Support_vector_machines#SVM_regrese