

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Evidenčné číslo: FIIT-5212-73688

Matúš Cuper

Optimalizácia konfiguračných parametrov predikčných metód

Bakalárska práca

Vedúci práce: Ing. Marek Lóderer

máj 2017

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Evidenčné číslo: FIIT-5212-73688

Matúš Cuper

Optimalizácia konfiguračných parametrov predikčných metód

Bakalárska práca

Študijný program: Informatika

Študijný odbor: 9.2.1 Informatika

Miesto vypracovania: Ústav informatiky a softvérového inžinierstva, FIIT STU v Bratislave

Vedúci práce: Ing. Marek Lóderer

máj 2017

Anotácia

Slovenská technická univerzita v Bratislave

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLOGIÍ

Študijný program: Informatika

Autor: Matúš Cuper

Bakalárska práca: Optimalizácia konfiguračných parametrov predikčných metód

Vedúci práce: Ing. Marek Lóderer

máj 2017

V práci sme sa zamerali na problémy vznikajúce pri predikcii časových radov. V súčasnosti existuje veľké množstvo metód, ktoré nám zabezpečujú predpoveď sledovanej veličiny s prijateľne malou odchýlkou na krátke obdobie v blízkej budúcnosti. Cieľom bakalárskej práce bolo vytvoriť systém, ktorý používateľovi poskytne jednoduché rozhranie pre porovnanie jednotlivých predikčných algoritmov nad množinou dát, ktorú si sám zvolí. Hľadanie ich optimálneho nastavenia sa vykonáva pomocou optimalizačných algoritmov založených na správaní sa živočíchov v prírode.

V práci sme analyzovali a opísali množinu predikčných a optimalizačných algoritmov. Navrhli sme systém na hľadanie optimálnych parametrov predikčných metód, čím sme výrazne ovplyvnili ich presnosť. Systém bol implementovaný v programovacom jazyku R a na vytvorenie používateľského rozhrania bola použitá knižnica Shiny. Optimalizácie sme vykonávali nad dátovými množinami v doméne energetiky. Výsledný systém umožňuje používateľovi využívať silu predikčných algoritmov a nájsť ich optimálne parametre pre zabezpečenie čo najpresnejšej predikcie.

Annotation

Slovak University of Technology Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
Degree Course: Computer Science
Author: Matúš Cuper
Bachelor thesis: Optimizing configuration parameters of prediction methods
Supervisor: Ing. Marek Lóderer
May 2017

In the thesis we focused on problems, which appear in time series prediction. In present there are many methods, which predict observed value with acceptable small deviation for short time period in near future. The aim of this bachelor thesis was creating system, which provides simple user interface to compare chosen prediction algorithms on dataset, which is chosen by user. Looking for their optimal setup is made by optimization algorithm based on nature-inspired behavior.

In the thesis we analyzed and described set of prediction and optimization algorithms. We designed system for searching optimal parameters of prediction methods, which influence their accuracy significantly. System was implemented in programming language R and the user interface was created by Shiny library. Optimization was provided on datasets in energetics domain. The final system provides to user to use force of prediction algorithms and find out their optimal parameters for the most accurate prediction.

ČESTNÉ PREHLÁSENIE

Čestne prehlasujem, že bakalársku prácu som vypracoval samostatne pod vedením vedúceho bakalárskej práce a s použitím odbornej literatúry, ktorá je uvedená v zozname použitej literatúry.

.....
Matúš Cuper

POĎAKOVANIE

Ďakujem vedúcemu bakalárskej práce Ing. Marekovi Lódererovi za odborné vedenie, cenné rady a pripomienky pri spracovaní bakalárskej práce.

Obsah

1	Úvod	1
2	Analýza problému	2
2.1	Časové rady	2
2.1.1	Analýza časových radov	2
2.1.2	Zložky časových radov	3
2.2	Analýza predikčných algoritmov	5
2.2.1	Lineárna regresná analýza	5
2.2.2	Stochastické modely	7
2.2.3	Regresia založená na podporných vektoroch	8
2.2.4	Rozhodovacie stromy	9
2.2.5	Neurónové siete	10
2.3	Analýza optimalizačných algoritmov	12
2.3.1	Genetický algoritmus	13
2.3.2	Optimalizácia svorkou divých vlkov	14
2.3.3	Umelá kolónia včiel	17
2.4	Meranie presnosti predpovede	18
2.4.1	Stredná chyba predpovede	19
2.4.2	Stredná absolútna chyba	19
2.4.3	Stredná percentuálna chyba	19
2.4.4	Stredná absolútna percentuálna chyba	19
2.4.5	Stredná štvorcová chyba	20
2.5	Zhodnotenie analýzy	20
3	Špecifikácia požiadaviek	21
4	Návrh riešenia	23
4.1	Architektúra aplikácie	23
4.2	Vlastnosti vstupných dát	24
4.3	Použité predikčné metódy	25
4.3.1	Regresia založená na podporných vektoroch	26
4.3.2	Autoregresný integrovaný model kĺzavého priemeru	26
4.3.3	Náhodné lesy	26
4.4	Použité optimalizačné metódy	27
4.4.1	Optimalizácia rojom častíc	28
4.4.2	Umelá kolónia včiel	28
5	Implementácia	29
5.1	Úprava vstupných dát	29
5.2	Použitie knižnice Shiny	32
5.3	Konfigurácia aplikácie	34
5.4	Postup pridávania predikčných algoritmov	34
5.5	Postup pridávania optimalizačných algoritmov	35

6	Experimentálne overenie	37
6.1	Optimalizácia parametrov SVR	37
6.2	Optimalizácia parametrov modelu ARIMA	37
6.3	Optimalizácia parametrov náhodných lesov	38
6.4	Porovnanie predpovedí s rôznymi fitness funkciami	38
7	Záver	39
Dodatok A	Technická dokumentácia	I
A.1	Spustenie aplikácie	I
A.2	Používateľská príručka	I
A.3	Obsah elektronického média	IV
Dodatok B	Popis experimentov	VI
B.1	Nastavenie parametrov optimalizačných metód	VI
B.2	Výsledky meraní	VII
Dodatok C	Plán do letného semestra	XI

Zoznam obrázkov

1	Príklad trendovej zložky časového radu.	3
2	Príklad sezónnej zložky časového radu.	4
3	Príklad reziduálnej zložky časového radu.	4
4	Príklad multiplikatívneho modelu.	5
5	Príklad aditívneho modelu.	6
6	Príklad neurónu v neurónovej sieti.	10
7	Príklad viacvrstvovej spätne propagovanej neurónovej siete.	11
8	Hierarchia vlkov vo svorke.	14
9	Príklad možného umiestnenia vlka na základe polohy koristi.	16
10	Stavový diagram aplikácie.	24
11	Ukážka dátumu a času pred transformáciou.	27
12	Ukážka transformácií dátumu a času pri predikcii náhodnými lesmi.	28
13	Úvodná obrazovka webovej aplikácie.	II
14	Výber trénovacej a testovacej množiny.	II
15	Výber parametrov optimalizačnej metódy.	III
16	Výber parametrov predikčnej metódy.	III
17	Zobrazenie vypočítaného výsledku používateľovi.	III

Zoznam tabuliek

1	Rozdelenie prírodne inšpirovaných algoritmov.	12
2	Proces vytvorenia novej generácie.	13
3	Rozbor formátu dátumu a času.	25
4	Použité knižnice jazyka R.	29
5	Použité datasety.	37
6	Parametre optimalizačných metód pri SVR.	VI
7	Parametre optimalizačných metód pri modeli ARIMA.	VI
8	Parametre optimalizačných metód pri náhodných lesoch.	VI
9	Parametre PSO a náhodných lesov pri rôznych fitness funkciách.	VI
10	Výsledky predpovedí SVR.	VII
11	Výsledky predpovedí modelu ARIMA.	VIII
12	Výsledky predpovedí náhodných lesov.	IX
13	Výsledky predpovedí náhodných lesov pri rôznych fitness funkciách.	X

1 Úvod

V súčasnosti sme obklopení množstvom zariadení, ktoré merajú a zhromažďujú informácie z iných zariadení. Príkladom môžu byť rôzne mobilné aplikácie, meteorologické stanice alebo inteligentné merače merajúce spotrebu elektrickej energie, vody či plynu. Namerané dáta sa môžu meniť v závislosti napríklad od hodiny, dňa alebo počasia. Samozrejme existujú aj merania, ktoré sú ovplyvnené ľudským správaním, ktoré sa môže líšiť v závislosti od vyššie uvedených faktorov, ale aj faktorov, ako sú kultúrne tradície, zvyky či náboženstvo. Na základe nameraných dát vieme vytvoriť predpoveď, ktorá opäť môže ovplyvniť ostatné faktory a celé predpovedanie sa tak stáva opäť komplexnejším.

V práci sme sa zamerali na predpovedanie veličín na základe ich historických meraní. Ostatné faktory pri tom neboli brané do úvahy. Tým sa stáva predpovedanie jednoduchšie a menej presné, čím vzniká priestor pre hlavný zámer práce, optimalizovanie predikčných metód na základe ich vstupných parametrov. Získame tým presnejšiu predpoveď ako pri ručnom nastavení a zároveň aj v kratšom čase ako by sme skúšali všetky riešenia.

Optimalizačné algoritmy, tak ako ich názov napovedá, slúžia na nájdenie optimálneho riešenia. Nájdenie najlepšieho riešenia požaduje preskúmanie všetkých možností. Stáva sa tak pomalým a výpočtovo náročným. Optimalizačné algoritmy rýchlo nájdu riešenie, ktoré je pre potreby našej práce postačujúce. Môžeme ich rozdeliť do viacerých skupín, v práci sa však zameriavame prednostne na prírodne inšpirované algoritmy, ktoré sú jednou z najefektívnejších podskupín.

Výsledný systém poskytuje používateľovi webové grafické rozhranie, pomocou ktorého môže predpovedať hodnoty vložených časových radov. Má možnosť zvoliť si medzi viacerými predikčnými a optimalizačnými algoritmami. Rozhranie poskytuje používateľovi základné informácie o algoritmoch, ako aj vysvetlenie efektov jednotlivých vstupných parametrov metód. Je na zvážení používateľa, aké parametre zvolí pre optimalizačné algoritmy. Vstupné parametre predikčných metód budú zvolené optimálne na základe výstupných hodnôt optimalizačných algoritmov. Používateľ má k dispozícii vyhodnotenie predpovede, ktoré porovnáva predpovedanú a skutočnú hodnotu rôznymi metrikami.

Celá práca je rozdelená do niekoľkých kapitol. V kapitole 2 sme sa zamerali na analýzu problému. Definovali sme kľúčové pojmy, použité metódy, opísali sme časové rady, ich vlastnosti, rozdelili sme predikčné a optimalizačné algoritmy do skupín. Kapitola 3 popisuje funkcionality, ktorú systém poskytuje používateľom. V kapitole 4 sa zameriavame na návrh systému a v kapitole 5 už samotnou implementáciou systému v jazyku R. Výsledky práce sú zhodnotené v kapitole 6.

2 Analýza problému

Predpovedanie spotreby elektrickej energie je kľúčovou činnosťou pre plánovanie a prevádzkovanie rôznych elektronických zariadení. Hľadaný vzor pre časové rady spotreby elektrickej energie je často komplexný a je zložitý aj kvôli faktorom, ako sú napríklad zmeny cien elektrickej energie na trhu. Preto sa stáva implementácia vhodného modelu zaručujúceho presnú predpoveď náročnou [17].

Práve preto vznikajú rôzne metódy na predpovedanie časových radov, ktoré sú bližšie popísané v nasledujúcich podkapitolách. Väčšina z nich poskytuje niekoľko vstupných parametrov, ako rozhranie pre vnútorné nastavenie algoritmov. Vďaka tomu máme možnosť ovplyvňovať mieru natrénovania modelu, veľkosť chyby predikcie alebo dĺžku obdobia, na ktorom budeme model trénovať. Nastavenie týchto parametrov sa môže pri rôznych datasetoch líšiť a preto neexistuje univerzálne riešenie. Hľadať riešenie pomocou prírodne inšpirovaných algoritmov je efektívne a nájdené riešenie je optimálne. Ďalej sú v tejto kapitole opísané spôsoby merania chýb, ktoré slúžia na vyhodnotenie efektivity a správnosti nájdeného riešenia.

2.1 Časové rady

Časový rad je množina dátových bodov nameraná v čase postupne za sebou. Matematicky je definovaný ako množina vektorov $x(t)$, kde t reprezentuje uplynulý čas. Premenná $x(t)$ je považovaná za náhodnú premennú. Merania v časových radoch sú usporiadané v chronologickom poradí [1].

Časové rady delíme na spojité a diskrétné. Pozorovania pri spojitých časových radoch sú merané v každej jednotke času, zatiaľ čo diskrétné obsahujú iba pozorovania v diskrétnych časových bodoch. Hodnoty toku rieky, teploty či koncentrácie látok pri chemickom procese môžu byť zaznamenané ako spojitý časový rad. Naopak, populácia mesta, produkcia spoločnosti alebo kurzy mien reprezentujú diskrétny časový rad. Vtedy sú pozorovania oddelené rovnakými časovými intervalmi, napríklad rokom, mesiacom či dňom [1]. V našom prípade sú namerané dáta dostupné každú celú štvrtinu.

2.1.1 Analýza časových radov

V praxi je vhodný model napasovaný do daného časového radu a zodpovedajúce parametre sú predpovedané na základe známych dát. Pri predpovedaní časových radov sú dáta z predchádzajúcich meraní zhromažďované a analyzované za účelom navrhnutia vhodného matematického modelu, ktorý zaznamenáva proces generovania dát pre časové rady. Pomocou tohto modelu sú predpovedané hodnoty budúcich meraní. Takýto prístup je užitočný, keď nemáme veľa poznatkov o vzore v meraniach idúcich za sebou alebo máme model, ktorý poskytuje nedostatočne uspokojivé výsledky [1].

Cieľom predikcií časových radov je predpovedať hodnotu premennej v budúcnosti na základe doteraz nameraných dátových vzoriek. Matematicky zapísané ako

$$\hat{x}(t + \Delta_t) = f(x(t - a), x(t - b), x(t - c), \dots) \quad (1)$$

Hodnota \hat{x} vo vzorci 1 je predpovedaná hodnota jednorozmerného diskrétného časového radu x . Úlohou je nájsť takú funkciu $f(x)$, pre ktorú bude \hat{x} predstavovať predpovedanú hodnotu časového radu. Táto funkcia by mala predpovedať hodnoty v budúcnosti konzistentne a objektívne [20].

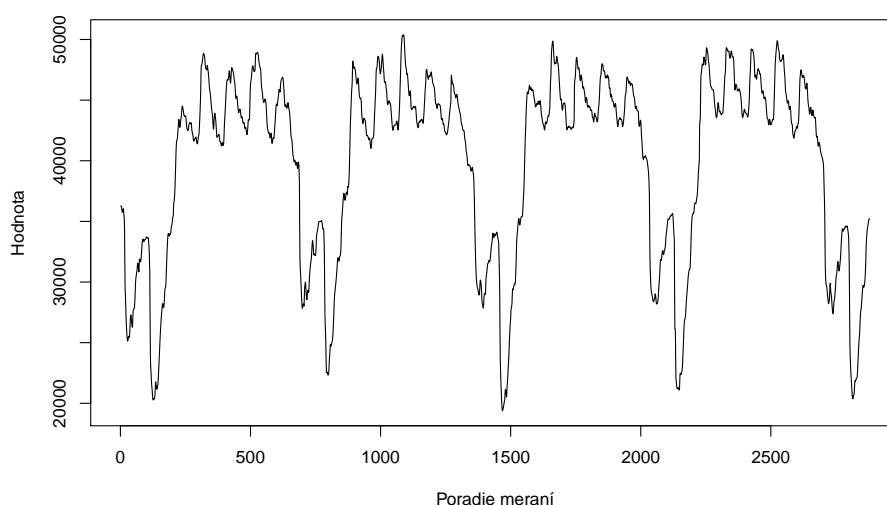
Časové rady sú najčastejšie vizualizované ako graf, kde pozorovania sú na osi y a plynúci čas na osi x . Pre lepšie vysvetlenie časových radov, budú nasledujúce odstavce obsahovať aj obrázky zobrazujúce zvolenú dátovú množinu.

2.1.2 Zložky časových radov

Pri predpovedaní časových radov ako napríklad meraní odberu elektrickej energie vznikajú 2 typy zmien. Prvým typom je trvalá alebo dočasná zmena spôsobená ekonomickými alebo ekologickými faktormi. Druhým typom je sezónna zmena, spôsobená zmenami ročných období a množstvom denného svetla. Môžeme ju pozorovať na úrovni dní, týždňov alebo rokov. Veličina, ktorú sa snažíme predpovedať postupne mení svoje správanie a model sa tak stáva nepresným. Kvôli tomu je nutné v každom modeli rozdeľovať tieto typy tendencií, aby sme vedeli model zmenám prispôbiť [8].

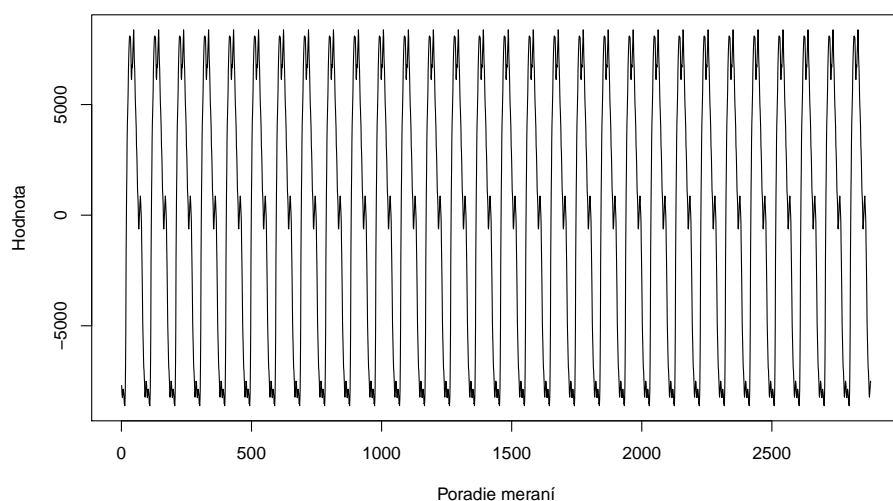
Vo všeobecnosti sú časové rady zložené zo 4 hlavných zložiek. Jedná sa o trendovú, cyklickú, sezónnu a reziduálnu zložku [1].

Trendová zložka predstavuje správanie časového radu v dlhodobom časovom horizonte. Z tohto pohľadu má časový rad tendenciu klesať, rásť alebo stagnovať. Príkladom môže byť nárast populácie či klesajúca úmrtnosť. Dekomponovanú zložku môžeme vidieť na obrázku 1 [1].

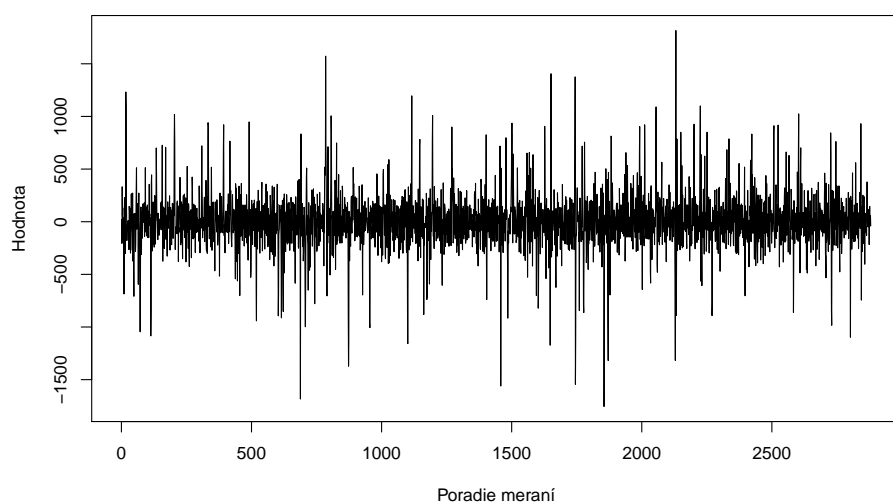


Obr. 1: Príklad trendovej zložky časového radu.

Cyklická zložka je spôsobená zmenami, ktoré sa cyklicky opakujú. Dĺžka periódy je 2 a viac rokov, čo zodpovedá strednodobému časovému horizontu. Táto zložka je zastúpená najmä pri ekonomických časových radoch napríklad podnikateľský cyklus pozostávajúci zo 4 fáz, ktoré sa stále opakujú [1].



Obr. 2: Príklad sezónnej zložky časového radu.



Obr. 3: Príklad reziduálnej zložky časového radu.

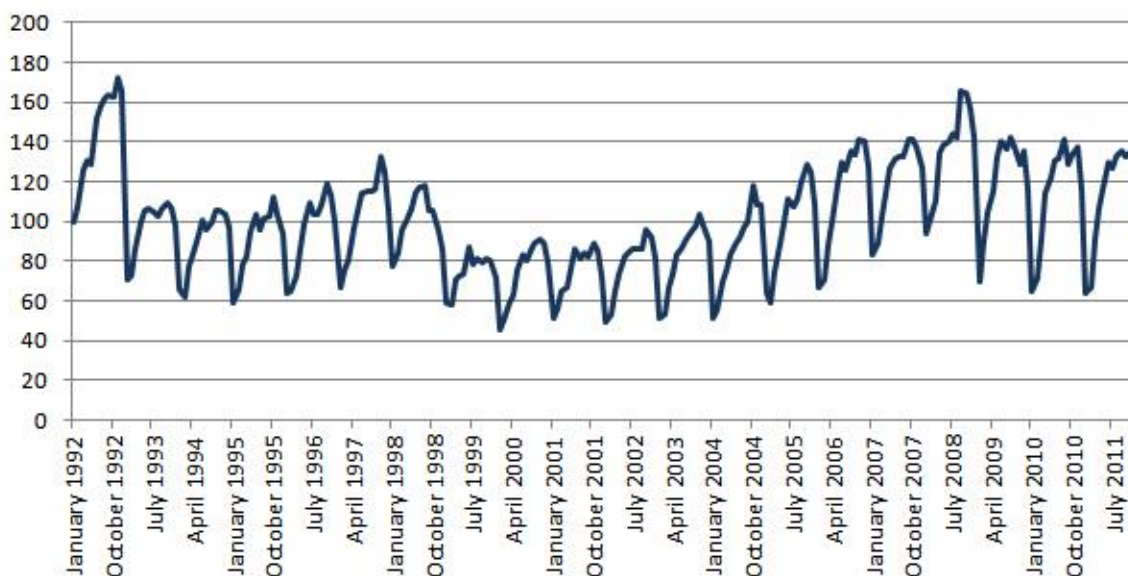
Sezónna zložka predstavuje kolísanie časových radov počas ročných období. Dôležitými faktormi pri tom sú napríklad klimatické podmienky, tradície alebo počasie. Napríklad predaj zmrzliny sa v lete zvyšuje, ale počet predaných lyžiarskych súprav klesá. Príklad dekomponovanej zložky môžeme vidieť na obrázku 2 [1].

Reziduálna zložka predstavuje veličinu, ktorá nemá žiadny opakovateľný vzor a ani dlhodobý trend. V časových radoch má nepredvídateľný vplyv na pozorovanú veličinu. V štatistike zatiaľ nie je definovaná metóda jej merania. Označuje sa aj ako náhodná zložka alebo biely šum. Je spôsobená nepredvídateľnými a nepravidelnými udalosťami. Viď obrázok 3 [1].

Vo všeobecnosti sa pre tieto 4 zložky používajú 2 rôzne modely. Je to multiplikatívny model a aditívny model.

$$\begin{aligned} Y(t) &= T(t) \times S(t) \times C(t) \times I(t) \\ Y(t) &= T(t) + S(t) + C(t) + I(t) \end{aligned} \quad (2)$$

Vo vzorci 2 predstavuje $Y(t)$ meranie v čase t . Premenné $T(t)$, $S(t)$, $C(t)$ a $I(t)$ sú zložkami trendu, sezónnosti, cyklu a náhodnosti. Multiplikatívny model, zobrazený na obrázku 4 je založený na predpoklade, že časové rady môžu byť na sebe závislé a môžu byť ovplyvňované medzi sebou, zatiaľ čo aditívny model, zobrazený na obrázku 5 predpokladá nezávislosť zložiek [1].



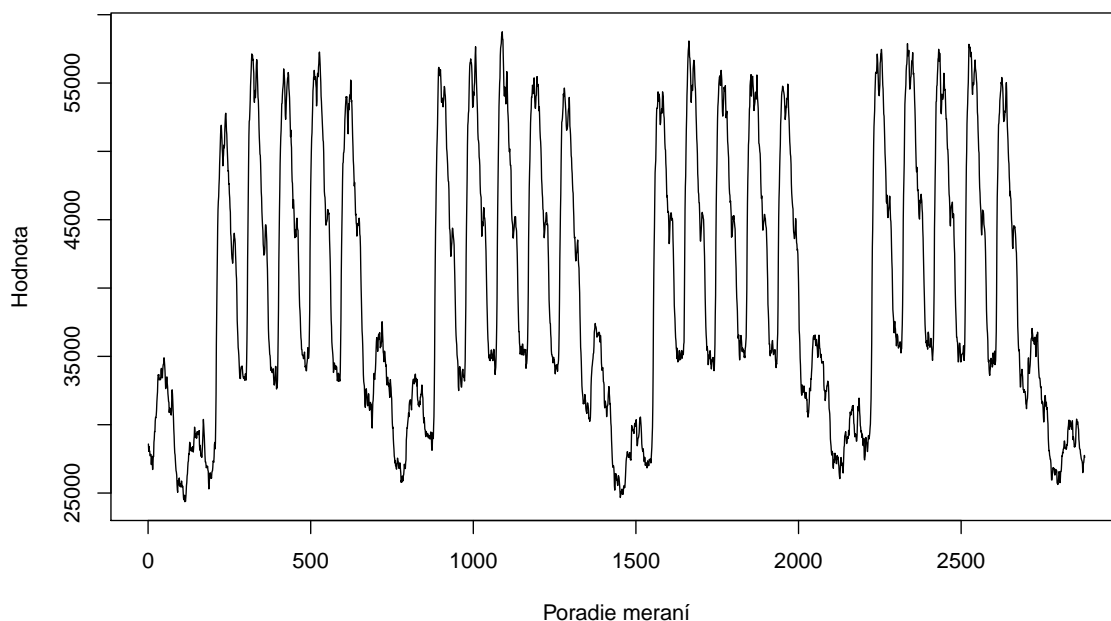
Obr. 4: Príklad multiplikatívneho modelu, index stavebnej produkcie Slovenska, Eurostat.

2.2 Analýza predikčných algoritmov

Na základe množstva výskumov, ktoré analyzovali predikčné algoritmy, bola zvolená reprezentatívna vzorka algoritmov. Našou snahou bude rôzne predikčné metódy optimalizovať pomocou optimalizačných algoritmov. Pred tým je potrebné analyzovať a pochopiť použité predikčné metódy. Taktiež je potrebné identifikovať ich vstupné parametre, keďže ich neskôr budeme optimalizovať.

2.2.1 Lineárna regresná analýza

Jedná sa o štatistickú metódu používanú na modelovanie vzťahov, ktoré môžu existovať medzi veličinami. Nachádza súvislosti medzi závislou premennou a potenciálnymi vysvetľujúcimi premennými. Používame pri tom vysvetľujúce premenné, ktoré môžu byť namerané súčasne so závislými premennými alebo aj premenné z úplne iných zdrojov. Regresná analýza môže byť tiež použitá na zlúčenie trendu a sezónnych zložiek do modelu. Keď je raz model vytvorený, môže byť použitý na zásah do spomínaných vzťahov alebo, v prípade dostupnosti vysvetľujúcich premenných, na vytvorenie predikcie [16].



Obr. 5: Príklad aditívneho modelu.

Lineárna regresia je najpoužívanější statistická metóda, ktorá modeluje vzťah závislej premennej a vysvetľujúcej premennej. Závislú premennú predstavuje veličina, ktorú sa snažíme predpovedať, čo je v našom prípade spotreba elektrickej energie. Vysvetľujúca premenná v sebe zahŕňa rôzne faktory, ktoré ovplyvňujú závislú premennú. Môžeme si pod tým predstaviť deň v týždni, počasie, tradície alebo rôzne udalosti, ktoré majú vplyv na predpoveď. [13, 17].

Predpokladajme typický regresný problém. Dáta pozostávajúce z množiny n meraní majú formát $\{(x_1, f(x_1)), \dots, (x_n, f(x_n))\}$. Úlohou regresie je odvodiť funkciu \hat{f} z dát, kde

$$\hat{f} : X \rightarrow \mathbb{R}, \text{ kde, } \hat{f}(x) = f(x), \forall x \in X, \quad (3)$$

Funkcia f vo vzorci 3 reprezentuje reálnu neznámu funkciu. Algoritmus, použitý na odvodenie funkcie \hat{f} , sa nazýva indukčný algoritmus. Funkcia \hat{f} sa nazýva model alebo prediktor. Obvykle je úlohou regresie minimalizovať odchýlku funkcie pre štvorcovú chybu, konkrétne strednú štvorcovú chybu MSE [18].

Keďže časový rad pozostáva z viacerých zložiek, môžeme ho zapísať ako funkciu $L(t)$ definovanú ako

$$L(t) = L_n(t) + \sum a_i x_i(t) + e(t) \quad (4)$$

Vo vzorci 4 funkcia $L_n(t)$ predstavuje odber elektrickej energie v čase t . Hodnota a_i je odhadovaný pomaly meniaci sa koeficient. Faktory $x_i(t)$ nezávisle vplyvajú na spotrebu elektrickej energie. Môže sa jednať napríklad o počasie alebo zvyky ľudí. Komponent $e(t)$ je biely šum, ktorý má nulovú strednú hodnotu a pevnú variáciu. Číslo n je počet meraní, obvykle 24 alebo 168, v našom prípade 96 meraní počas jedného dňa [13].

Viacnásobná lineárna regresia sa snaží modelovať vzťah medzi dvoma alebo viacerými vysvetľujúcimi premennými a závislou premennou vhodnou lineárnou rovnicou pre poz-

rované dáta. Výsledný model je vyjadrený ako funkcia viacerých vysvetľujúcich premenných [8].

Túto funkciu môžeme zapísať ako

$$Y(t) = V_t a_t + e_t \quad (5)$$

Vo vzorci 5 t označuje čas, kedy bolo meranie uskutočnené. $Y(t)$ predstavuje celkový nameraný odber elektrickej energie. Vektor V_t reprezentuje hodnoty vysvetľujúcich premenných v čase merania. Vysvetľujúce premenné môžu predstavovať meteorologické vplyvy, ekonomický nárast, ceny elektrickej energie či kurzy mien. Chybu modelu v čase t zapíšeme ako e_t [13, 17].

2.2.2 Stochastické modely

Tieto metódy časových radov sú založené na predpoklade, že dáta majú vnútornú štruktúru, ako napríklad autokoreláciu, trend či sezónnu variáciu. Najprv sa precízne zostaví vzor zodpovedajúci dostupným dátam a potom sa na jeho základe predpovie budúca hodnota veličiny [13].

Autoregresný model predpovedá budúcu hodnotu premennej ako súčet lineárnej kombinácie p predchádzajúcich meraní, náhodnej chyby a konštanty. V literatúre sa označuje ako AR (angl. autoregressive model). Matematicky môžeme autoregresný model zapísať ako

$$y_t = c + \sum_{i=1}^p \varphi_i y_{t-i} + \varepsilon_t \quad (6)$$

Vo vzorci 6 hodnota y_t predstavuje predpovedanú hodnotu v čase t . Náhodnú chybu v čase t zapíšeme ako ε_t . Hodnoty φ_i sú parametre modelu a c je konštanta. Konštantou p označujeme rad modelu [1].

Model kľzavého priemeru na rozdiel od autoregresného modelu používa ako vysvetľujúce premenné chyby predchádzajúcich meraní a nie priamo hodnoty. V literatúre sa označuje ako MA (angl. moving average). Matematicky môžeme tento vzťah zapísať ako

$$y_t = \mu + \sum_{j=1}^q \Theta_j \varepsilon_{t-j} + \varepsilon_t \quad (7)$$

Vo vzorci 7 hodnota y_t predstavuje strednú hodnotu postupnosti meraní v čase t . Hodnoty Θ_j sú parametre modelu a konštantou q označujeme rad modelu. Vychádzame z predpokladu, že náhodná zložka ε_t je biely šum, čo je rovnomerne distribuovaná náhodná premenná, ktorá má nulovú strednú hodnotu a konštantnú variáciu σ^2 [1].

Autoregresný model kľzavého priemeru reprezentuje súčasnú hodnotu časového rádu lineárne na základe jeho hodnôt a hodnôt bieleho šumu v predchádzajúcich periódach. V literatúre sa označuje ako ARMA (angl. autoregressive moving average) [13].

Ide o kombináciu autoregresie (AR) a kľzavého priemeru (MA), vhodnú pre modelovanie jednorozmerných časových radov. Rad modelu určuje p a q . Matematicky môžeme reprezentovať tento model ako súčet predchádzajúcich modelov [1]

$$y_t = c + \varepsilon_t + \sum_{i=1}^p \varphi_i y_{t-i} + \sum_{j=1}^q \Theta_j \varepsilon_{t-j} \quad (8)$$

Autoregresný integrovaný model kľzavého priemeru je generalizáciou modelu ARMA. V literatúre sa označuje ako ARIMA (angl. autoregressive integrated moving average). Modely typu ARMA môžu byť použité iba na statické časové rady. Mnoho časových radov v praxi vykazujú nestatické správanie, napríklad tie, ktoré obsahujú komponenty trendu a sezónnosti. Kvôli tomu bol navrhnutý model ARIMA, ktorý zahŕňa v sebe aj prípady nestatických časových radov. Z nestatických časových radov sa vytvárajú statické pomocou konečného počtu derivovaní dátových bodov. Vzniká tak matematický model, ktorý môžeme zapísať ako

$$\left(1 - \sum_{i=1}^p \varphi_i L^i\right)(1 - L)^d y_t = \left(1 + \sum_{j=1}^q \Theta_j L^j\right) + \varepsilon_t \quad (9)$$

Vzorec 9 môžeme zapísať aj jednoduchšie a to

$$\varphi(L)(1 - L)^d y_t = \Theta(L)\varepsilon_t \quad (10)$$

Vo vzorci 9 predstavujú premenné p , d a q rad autoregresného modelu, modelu kľzavého priemeru a integrovaného modelu. Hodnota d zodpovedá stupňu derivovania, zvyčajne je rovná 1. V prípade, že $d = 0$ dostaneme klasický ARMA model. Rovnakým spôsobom vieme dostať modely AR a MA [1].

2.2.3 Regresia založená na podporných vektoroch

Regresia založená na podporných vektoroch a metóda podporných vektorov je založená na štatistickej teórii učenia, nazývanej aj VC teória, podľa svojich autorov, Vapnik a Chervonenkisa. Táto predikčná metóda poskytuje malý počet voľných parametrov. Garantuje konvergenciu k ideálnemu riešeniu a môže byť výpočtovo efektívna [20].

Metóda podporných vektorov je použitá na množstvo úloh strojového učenia ako je rozoznávanie vzorov, klasifikácia objektov a v prípade predikcií časových radov to je regresná analýza. Regresia založená na podporných vektoroch je postup, ktorého funkcia je predpovedaná pomocou nameraných dát, ktorými je metóda podporných vektorov natrénovaná. Toto je odklon od tradičných predpovedí časových radov v zmysle, že metóda podporných vektorov nepoužíva žiadny model, ale predikciu riadia samotné dáta [20].

Uvažujme množinu tréningových dát $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n) \subset \chi \times \mathbb{R}$, kde v našom prípade hodnota x_i predstavuje odber elektrickej energie v čase i a množina χ označuje vstupnú množinu dát. Úlohou algoritmu je nájsť funkciu $f(x_i)$, pre ktorú hodnota ε nadobúda pre všetky tréningové dáta čo najväčšie hodnoty oproti y_i a súčasne nadobúda najmenšie možné hodnoty menšie ako ε [24].

Matematicky môžeme túto lineárnu zapísať ako

$$f(x) = \langle w, x \rangle + b \quad (11)$$

Vo vzorci 11 platí $w \in \chi$ a $b \in \mathbb{R}$. Výsledok skalárneho súčinu $\langle w, x \rangle$ sa nachádza v množine χ . Našou snahou je dosiahnuť čo najmenšiu hodnotu w . Zabezpečiť to môžeme minimalizovaním hodnoty $\|w\|^2 = \langle w, w \rangle$. Pri celi minimalizovať hodnotu $\frac{1}{2}\|w\|^2$, môžeme opisovaný problém zapísať ako

$$\begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon \end{cases} \quad (12)$$

Predpokladom pre vzorec 12 je, že funkcia f existuje pre všetky páry (x_i, y_i) s presnosťou ε [24].

Na vykonanie regresie založenej na podporných vektoroch je potrebné zmapovať vstupný priestor $x(i)$ do vyššej dimenzie označovanej ako $\varphi(x(i))$, pričom riešenie závisí od skalárneho súčinu vstupného priestoru a kernel funkcie, ktorá vykonáva spomínané mapovanie. Kernel funkcia mapuje nelineárne dáta do vyššej dimenzie a musí pritom spĺňať Mercerovu podmienku, ktorú zapíšeme ako

$$k(x, x') = \langle \phi(x), \phi(x') \rangle \quad (13)$$

Použitie kernelových funkcií je kľúčové pri použití regresii založenej na podporných vektoroch. Pri výbere vhodnej funkcie je nevyhnutná empirická analýza. Jednou z najčastejších volieb je Gaussovská kernel funkcia [20].

2.2.4 Rozhodovacie stromy

Rozhodovacie stromy sú jednou z najrozšírenejších učiacich metód. Používajú sa najmä na klasifikáciu, ale v súčasnosti sa využívajú aj na regresiu. Rozhodovací strom je reprezentovaný ako množina uzlov a im prislúchajúcich hrán. Uzly reprezentujú atribúty a výstupné hrany sú vždy označené konkrétnou hodnotou pre atribút, z ktorého vychádzajú. Rozhodovanie začína v koreni stromu a končí po dosiahnutí listového uzla. Pre riešenie jedného problému je možné vytvoriť stromy s rôznym počtom a usporiadaním uzlov. Najlepším riešením je strom s najmenším počtom rozhodovacích uzlov [19].

Hlavnou výhodou rozhodovacích stromov oproti ostatným modelom je, že strom produkuje model, ktorý je reprezentovateľný ako pravidlá alebo logické výroky. Taktiež klasifikácia sa môže vykonávať bez komplikovaných výpočtov. Táto technika môže byť použitá ako na kategorické premenné tak aj na spojité. Rozhodovacie stromy neposkytujú také dobré výsledky pre nelineárne dáta ako neurónové siete. Vo všeobecnosti je táto metóda vhodnejšia pre predpovedanie kategorických dát alebo na dáta, ktoré v sebe obsahujú viditeľný trend [25].

Regresný rozhodovací strom obsahuje odozvoyý vektor Y reprezentujúci odozvu hodnôt ku každému meraniu v matici X . Vetvy sú rozdeľované na základe štvorcového zostatkového minimalizačného algoritmu. Ten zabezpečuje, že očakávaný súčet variácií dvoch uzlov bude minimalizovaný. Algoritmus tak nájde optimálnu podmienku rozdelenia uzla na jeho potomkov [3].

Priradením hodnoty 1 pre triedu k a hodnoty 0 pre ostatné triedy, získame variáciu rovnú $p(k|t)[1 - p(k|t)]$. Sčítaním K tried získame funkciu

$$i(t) = 1 - \sum_{k=1}^K p^2(k|t) \quad (14)$$

Tým sme vytvorili maximálny strom, čo znamená, že uzly sa rozdeľovali až do posledného merania, ktoré sa nachádzalo v trénovacej množine. Maximálny strom sa tak môže stať veľmi veľkým [3].

Jedným z najpoužívanějších algoritmov na rozdeľovanie uzlov sa nazýva index Gini alebo rozhodovacie pravidlo Gini. Tento algoritmus zapíšeme ako

$$i(t) = \sum_{k \neq l} p(k|t)p(l|t) \quad (15)$$

Vo vzorci 15 predstavujú k a l indexy tried o 1 po K a $p(k|t)$ označuje podmienenú pravdepodobnosť triedy k za predpokladu, že aktuálny uzol je uzol t . Aplikovaním Gini algoritmu nájdeme v trénovacej množine najväčšiu triedu, ktorú odizolujeme od ostatných dát [3].

Náhodné lesy sú kombináciou predpovedí stromov. Každý strom závisí od hodnoty náhodného vektora s rovnakým rozdelením. Chyba lesu závisí od sily jednotlivých stromov a koreláciou medzi nimi. Náhodný les môžeme definovať ako klasifikátor pozostávajúci z množiny stromov $\{h(x, \Theta_k), k = 1, 2, \dots\}$, kde $\{\Theta_k\}$ sú nezávislé rovnomerne distribuované náhodné vektory a každý strom sa podieľa hlasom na voľbe triedy vstupu x . S nárastom počtu stromov hodnota $\{\Theta_k\}$ konverguje k určitému bodu. Tým je zabezpečené, že náhodné lesy sa s pridávaním počtom stromov nepretrénujú, ale veľkosť chyby sa postupne ustáli. Pri výbere náhodného vektora sa snažíme pri zachovaní jeho sily minimalizovať koreláciu, čím zvyšujeme presnosť celého výpočtu [4].

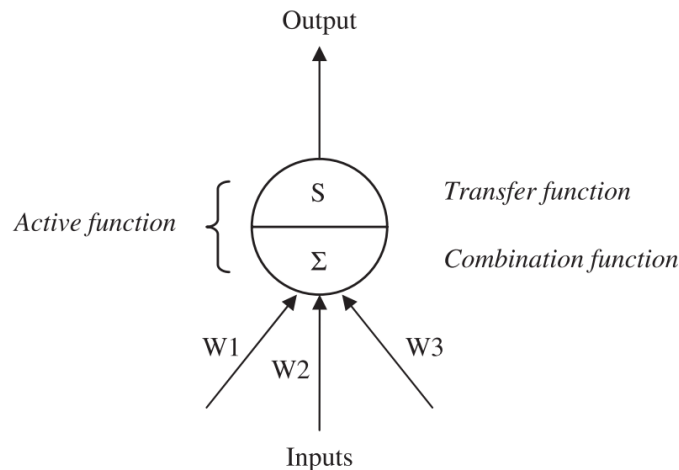
Väčšinou sú náhodné lesy používané na klasifikačné problémy, avšak je možné ich aplikovať aj na regresiu. Regresné náhodné lesy sú tvorené rastom stromov závislých na náhodnom vektore $\{\Theta\}$. Prediktor stromu $h(x, \Theta)$ nadobúda číselné hodnoty na rozdiel od štítkov tried ako je to pri klasifikačných problémoch. Predpokladáme trénovaciu množinu, ktorá je nezávislou distribúciou náhodného vektora Y, X . Potom môžeme strednú štvorcovú generalizačnú chybu pre číselný prediktor h matematicky zapísať ako

$$E_{X,Y}(Y - h(X))^2 \quad (16)$$

Prediktor náhodného lesu je tvorený priemerom k stromov, čo zapíšeme ako $h(x, \Theta_k)$ [4].

2.2.5 Neurónové siete

Návrh neurónových sietí je inšpirovaný neurofyziológiou ľudského mozgu. Model je analytická technika modelujúca procesy učenia v kognitívnych systémoch a neurologických funkciách mozgu. Má schopnosť predpovedať budúcu hodnotu merania konkrétnej premennej na základe hodnôt z predchádzajúcich meraní. Tento proces sa inak nazýva aj učenie z existujúcich dát [25]. Tok v neurónovej sieti preteká cez jednotlivé neuróny. Na obrázku 6 môžeme vidieť príklad takéhoto neurónu [25].

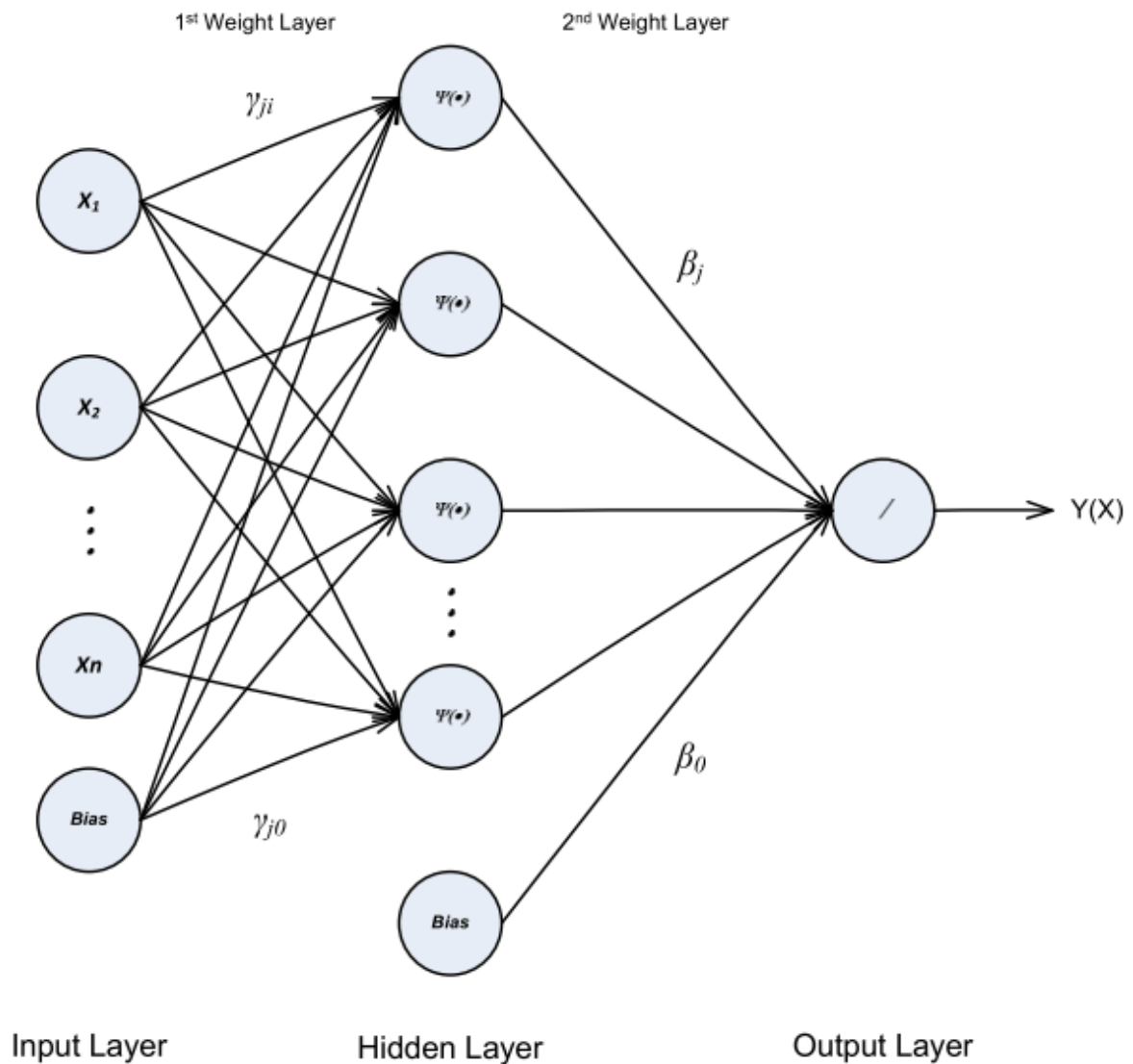


Obr. 6: Príklad neurónu v neurónovej sieti [25].

Neurónová sieť predstavuje orientovaný graf uzlov. Uzol neurónovej siete sa nazýva neurón. Každý uzol počíta svoj výstup na základe vstupov od susedných uzlov. Výpočet sa vykonáva aplikovaním funkcie, ktorá sa nazýva aplikačná funkcia na vážený súčet vstupov [9].

Trénovanie siete je proces nastavovania, čo najlepších váh na vstupy jednotlivých neurónov. Chyba neurónovej siete sa najčastejšie počíta pomocou spätnej propagácie (angl. backpropagation), čím dostaneme rast chyby pre danú neurónovú sieť [25].

Je veľa typov neurónových sietí napríklad viacvrstvové percepčné siete, samoriadiace siete, siete s viacerými skrytými vrstvami, alebo aj viacvrstvové spätne propagované neurónové siete, pričom príklad takejto siete môžeme vidieť aj na obrázku 7. V každej skrytej vrstve je určité množstvo neurónov. Trénovanie neurónovej siete obvykle zaberá veľa času. Výstupom je lineárna rovnica váh prepojených so vstupom [13].



Obr. 7: Príklad viacvrstvovej spätne propagovanej neurónovej siete [12].

Viacvrstvový percepčón je jednou z najpoužívanějších neurónových sietí. Pozostáva z uzlov a im prislúchajúcim hranám. Uzly sú zoskupované do rôznych vrstiev. Prvá vrstva je vstupná vrstva, kde počet d označuje počet vstupných parametrov vstupujúcich do siete. Táto vrstva je následne prepojená hranami so skrytou vrstvou pozostávajúcou z h uzlov. Tá je potom prepojená s výstupnou vrstvou s c uzlami. Kvôli tomu sa tieto siete zvyknú označovať aj ako dopredné siete [19].

Elementy skrytých a výstupných vrstiev sú neuróny pozostávajúce z uzlov, viacerých vstupujúcich a jednou výstupnou hranou. Funkciou neurónu je transformovať lineárnu kombináciu vstupov pomocou nelineárnej aktivačnej funkcie, čiže každú vstupnú hranu prenasobiť jej váhou a výsledok týchto súčinov sčítať. Tak dostaneme pre neurón j vzorec 17 opisujúci

lineárnu kombináciu vstupov a_j

$$a_j = \sum_{i=1}^d w_{ji} x_i \quad (17)$$

Váha w_{ji} označuje váhu medzi neurónom i na vstupnej vrstve a neurónom j na skrytej vrstve [19].

Aktivovanie neurónu j závisí od jeho aktivačnej funkcie $g(a_j)$. Jednu z najpoužívanejších aktivačných funkcií, logistickú sigmoidnú funkciu, môžeme matematicky zapísať ako

$$g(a) \equiv \frac{1}{1 + \exp(-a)} \quad (18)$$

Je zrejmé, že funkcia zo vzorca 18 vracia hodnoty v rozmedzí $(0, 1)$ [19].

2.3 Analýza optimalizačných algoritmov

Optimalizačné algoritmy majú potenciál nájsť globálne alebo lokálne riešenie problému. Lokálne optimalizácie, nazývané aj vyhľadávacie algoritmy, sa pokúšajú nájsť lokálne minimum v okolí štartovacieho riešenia. Väčšina týchto algoritmov je deterministická. Pri hľadaní minima používajú vyhodnocovaciu funkciu, na základe ktorej aktualizujú doterajšie riešenie. Z nových možných riešení je najlepšie to s najnižšou hodnotou, čiže to, ktoré je najlacnejšie. Kvôli tejto vlastnosti sa zvyknú tieto algoritmy označovať aj ako nenásytné algoritmy (angl. greedy algorithms). Spravidla tieto algoritmy nenájdu globálne minimum v prípade, že sa nachádza ďalej od štartovacieho riešenia ako nejaké lokálne minimum [22].

Na druhej strane, optimalizačné algoritmy s potenciálom nájdania globálneho minima nachádzajú riešenie, ktoré postupne konverguje k optimálnemu riešeniu. To ale nie je algoritmami úplne garantované. Algoritmy s globálnym potenciálom majú väčší prehľad o svojom okolí, a preto uviaznutie v lokálnom minime je zriedkavé [22].

Nasledujúca tabuľka 1 zobrazuje rozdelenie algoritmov s potenciálom nájdania globálneho minima. Algoritmy sú prírodne inšpirované a ich ďalšie delenie vyplýva zo spoločných znakov, na ktorých sú založené. V nasledujúcich kapitolách sa zaoberáme niektorými z nich, konkrétne genetickým algoritmom, optimalizáciou svorkou divých vlkov a umelou kolóniou včiel.

Tabuľka 1: Rozdelenie vybraných prírodne inšpirovaných algoritmov [7].

Model ľudskej mysle	Umelé imunitné systémy	Rojová inteligencia	Ostatné napr. prírodné úkazy
Teória fuzzy množín	Genetický algoritmus	Optimalizácia kolóniu mravcov	Tektonické dosky
Teória približných množín	Umelé neurónové siete	Optimalizácia rojom častíc	Veľký kolaps
Granulárne výpočty	Celulárny automat	Inteligentná kvapka vody	Oceánske prúdy
Predbežné výpočty	Membránové výpočty	Včelí zhlukovací algoritmus	Prílivovalné vlny
		Vyhľadávanie kukučkou	Sopečné erupcie
			Zemetrasenie

2.3.1 Genetický algoritmus

Genetický algoritmus patrí medzi prírodne inšpirované algoritmy patriace do triedy umelých imunitných systémov [5]. Genetický algoritmus je stochastický optimalizačný algoritmus, ktorého úlohou je nájsť globálne riešenie pre zadaný problém, čiže sa nestane, že riešenie spadne do lokálneho minima a nenájde sa tak optimálne riešenie. Od tradičných algoritmov sa líšia hlavne v počte riešení, ktoré sú kandidátmi na najlepšie riešenie. Tradičné vyhľadávacie algoritmy prehľadávajú dôkladne iba jedno riešenie, zatiaľ čo genetické algoritmy hlbšie spracujú viacero kandidátov naraz. Každý kandidát na optimálne riešenie problému je reprezentovaný dátovou štruktúrou, ktorú označujeme pojmom jedinec. Súbor jedincov tvorí populáciu. Začiatok procesu začína náhodnými riešeniami populácie, ktorý sa postupne vylepšuje [5].

Vytvorenie novej generácie sa vykonáva pomocou genetických operátorov: a to selekciou, krížením a mutáciou. Proces selekcie vyberie kvalitnejšie chromozómy, ktoré prežijú a vyskytnú sa tak aj v ďalšej generácii [23]. Pri genetických algoritmoch sa zavádzajú pojmy ako chromozóm, fitness funkcia, kríženie, elitárstvo, operátor reprodukcie či mutácie [5].

Chromozóm je pomenovanie pre jedinca. V literatúre sa tiež zvykne používať označenie kandidát [2].

Inicializácia je proces, ktorý po vygenerovaní generácie priradí kandidátom náhodné hodnoty. Pri prehľadávaní dvojrozmerného priestoru to budú náhodné hodnoty oboch súradníc [15].

Fitness funkcia je funkcia určujúca efektívnosť chromozómu. Každý jedinec v množine je ohodnotený pomocou tejto funkcie, ktorá vracia číselnú reprezentáciu riešenia, čiže ako dobre daný kandidát vyriešil zadaný problém. Pri hľadaní optimálneho riešenia sa porovnáva hodnota fitness funkcie aktuálneho riešenia s hodnotou funkcie cieľového riešenia, ale vo všeobecnosti, čím je hodnota väčšia, tým je kandidátovo riešenie lepšie [5, 15, 23].

Operátor reprodukcie je obvykle prvý operátor, ktorý sa uplatní na populáciu. Operátor náhodne vyberie reťazce z dvoch chromozómov na kríženie [5].

Kríženie je operátor kombinácie. Kríženie vykonáva výmenu blokov chromozómov. Z druhého chromozómu je vybraný reťazec náhodnej veľkosti, ktorý sa vymení s rovnako dlhým reťazcom z prvého chromozómu. V tabuľke 2 je tento proces znázornený graficky [5].

Tabuľka 2: Proces vytvorenia novej generácie z rodičovských chromozómov.

rodič č. 1	0110	0101 0010	0011
rodič č. 2	1100	1101 1110	1111
potomok č. 1	0110	1101 1110	0011
potomok č. 2	1100	0101 0010	1111

Elitárstvo je proces pridávania chromozómov s najlepšou hodnotou funkcie fitness priamo do ďalšej populácie. Zaisťuje to, že najlepšie riešenie budúcej generácie bude vždy lepšie alebo prinajhoršom rovnaké, ako najlepšie riešenie predchádzajúcej generácie [6].

Operátor mutácie sa vykoná po vykonaní operátora reprodukcie. Mutácia chromozómu väčšinou predstavuje operáciu, ktorá s nízkou pravdepodobnosťou invertuje jednu hodnotu v chromozóme [5].

Princíp genetického algoritmu môžeme znázorniť v nasledujúcom pseudokóde [5]

Algorithm 1 Pseudokód genetického algoritmu.

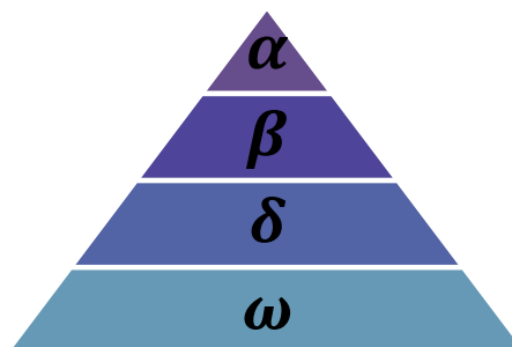
- 1: Náhodne inicializovanie jedincov
 - 2: Vyhodnotenie fitness funkcie pre každého jedinca
 - 3: Výber jedincov pre ďalšiu populáciu na základe fitness funkcie
 - 4: Kríženie jedincov
 - 5: Mutovanie jedincov
 - 6: Ak bolo nájdené žiadané optimálne riešenie pokračuj, inak návrat na krok 2
 - 7: Vráť optimálne riešenie
-

Veľkou výhodou genetického algoritmu je, že mutácia predchádza sklznutiu do lokálnych miním a kombinácia chromozómov vedie k rýchlemu približovaniu sa k optimálnemu riešeniu. Napriek týmto výhodám, majú genetické algoritmy aj niekoľko nevýhod [6].

- Reprezentovanie kandidátov je príliš obmedzujúce
- Mutácia a kríženie sú v súčasnosti aplikovateľné iba na chromozómy reprezentované bitovým reťazcom alebo číslami
- Definovanie fitness funkcie je často netriviálnou záležitosťou a jej generalizácia je náročná

2.3.2 Optimalizácia svorkou divých vlkov

Algoritmus je založený na správaní vlka sivého, ktorý je na vrchole potravinového reťazca a preferuje život vo svorke. Lov pozostáva zo stopovania, prenasledovania, približovania sa, obkľúčenia koristi a útokom na korisť. Vlkov vo svorke možno rozdeliť do niekoľkých skupín. V obrázku 8 je znázornená hierarchia týchto skupín [21].



Obr. 8: Hierarchia vlkov vo svorke [21].

Alfa vlky sú vodcami svorky. Ich úlohou je robiť rozhodnutia ohľadom lovu, miesta na spanie či času zobudenia. Tieto príkazy diktujú svorke, avšak bolo pozorované aj demokratické správanie, kedy alfa vlky nasledovali ostatných členov svorky. Všetky vlky uznávajú postavenie alfy vo svorke. Zaujímavosťou je, že vodcom nemusí byť najsilnejší jedinec, ale môže to byť aj jedinec najlepší v organizovaní svorky [21].

Beta vlky sú druhým stupňom v hierarchii, podriadené alfa vlkom, ktorým pomáhajú v rozhodovaní a organizácii. Sú najlepšími kandidátmi na alfu v prípade úmrtia alebo zostarnutia alfa jedincov. Hrajú rolu radcu a ďalej distribuujú príkazy a vracajú sa s odozvou na ne [21].

Delta vlky inak nazývané aj podriadené, zastupujú vo svorke úlohy prieskumníkov, strážcov, starejších, lovcov či opatrovateľov. Prieskumníci hliadajú hranice a varujú svorku pred nebezpečením. Strážcovia sa starajú o bezpečie svorky. Starejší sú skúsenými vlkami, ktorí boli na pozícii alfa alebo beta. Lovci pomáhajú alfa a beta vlkom pri love. Opatrovatelia sa starajú o slabé, choré alebo zranené jedince [21].

Omega vlky majú najnižšiu hodnotu vo svorke. Hrajú rolu obetných baránkov, ktoré sa podrobia ostatným. K potrave sa dostanú ako úplne posledné. Aj keď sa možno javí ich postavenie zbytočné, boli pozorované prípady, kedy ich strata spôsobila vo svorke nedorozumenia [21].

Spoločenská hierarchia reprezentovaná matematickým modelom, označuje najvhodnejšie riešenie ako alfa, druhé najvhodnejšie ako beta resp. delta. Riešenia ostatných kandidátov označujeme ako omega. Algoritmus optimalizácie (v prírode lovu) je vedený alfa, beta a delta kandidátmi, ktorí sú nasledovaní kandidátmi omega [21].

Obkľúčenie koristi jednotlivými vlkami α , β a δ môžeme matematicky vyjadriť nasledujúcimi rovnicami

$$\vec{D}_\alpha = |\vec{C}_1 \cdot \vec{X}_\alpha - \vec{X}| \quad (19)$$

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1 \cdot \vec{D}_\alpha \quad (20)$$

$$\vec{D}_\beta = |\vec{C}_2 \cdot \vec{X}_\beta - \vec{X}| \quad (21)$$

$$\vec{X}_2 = \vec{X}_\beta - \vec{A}_2 \cdot \vec{D}_\beta \quad (22)$$

$$\vec{D}_\delta = |\vec{C}_3 \cdot \vec{X}_\delta - \vec{X}| \quad (23)$$

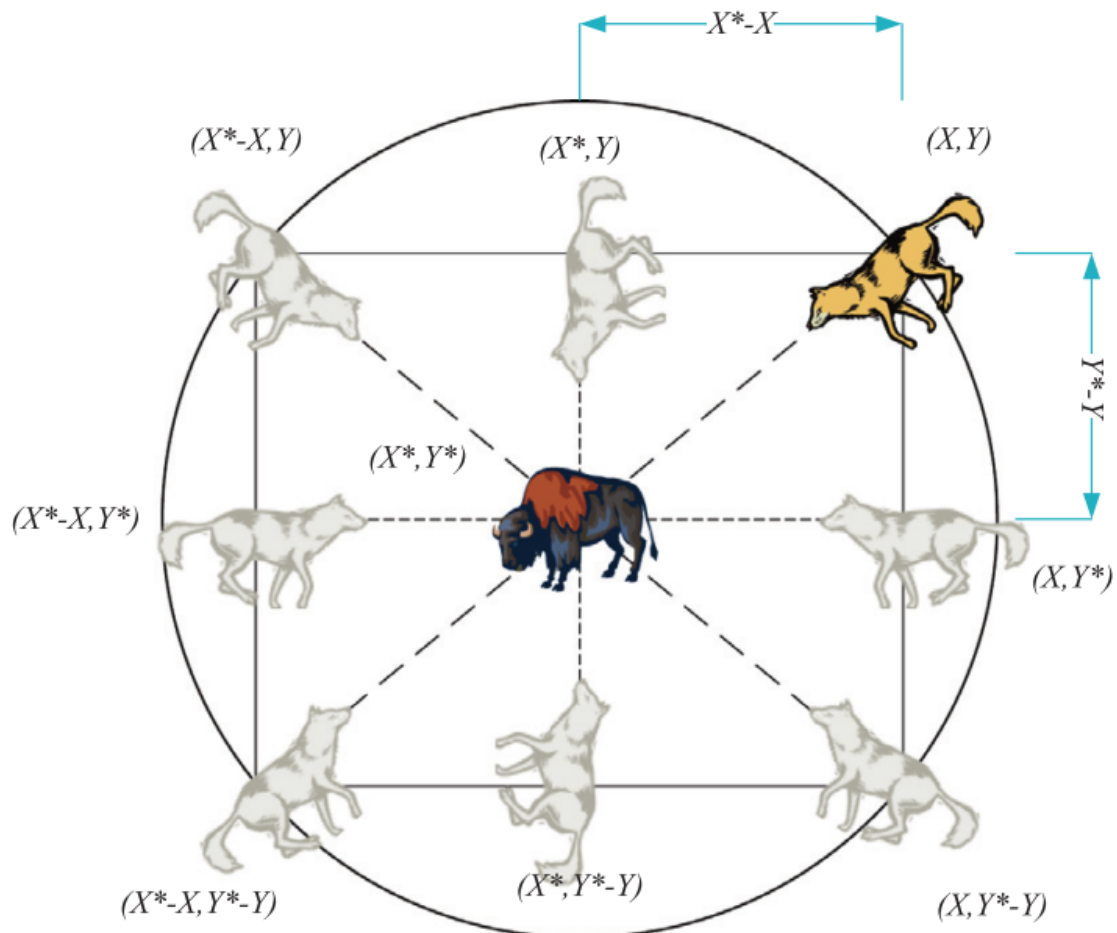
$$\vec{X}_3 = \vec{X}_\delta - \vec{A}_3 \cdot \vec{D}_\delta \quad (24)$$

Vo vzorcoch 20, 22 a 24 predstavujú vektory \vec{X}_1 , \vec{X}_2 a \vec{X}_3 polohu koristi a vektory \vec{X}_α , \vec{X}_β a \vec{X}_δ polohu vlkov α , β a δ . Vektory \vec{A} a \vec{C} sú koeficienty, ktoré zapíšeme ako

$$\vec{A} = 2\vec{a} \cdot \vec{r}_1 - \vec{a} \quad (25)$$

$$\vec{C} = 2 \cdot \vec{r}_2 \quad (26)$$

Vo vzorci 25 sa premenná \vec{a} počas výpočtu lineárne znižuje od 2 po 0. Vektory \vec{r}_1 a \vec{r}_2 vo vzorci 26 sú náhodnými vektormi v rozsahu $[0, 1]$. Vlk môže svoju pozíciu (X, Y) aktualizovať v závislosti od pozície koristi (X^*, Y^*) . Obrázok 9 ilustruje možné aktualizované pozície, ktoré môže najlepší agent dosiahnuť. Tieto pozície získame aplikovaním vzorcov 19 až 24 [21].



Obr. 9: Príklad možného umiestnenia vlka v dvojrozmernom priestore na základe polohy koristi [21].

Hľadanie koristi je zabezpečené vektorom \vec{A} , ktorý mimo intervalu $[-1, 1]$ núti vlky divergovať od seba, čím je zdôraznená potreba prehľadávať okolie a neskĺznuť do lokálneho minima. Náhodný vektor \vec{C} simuluje prekážky v prírode, s ktorými sa vlk stretne pri hľadaní koristi. V závislosti od vygenerovanej hodnoty, môže simulovať aj opačný prípad, ktorý je pre vlka priaznivejší [21].

Lov je vedený alfa vlkami, beta a delta vlky sa tiež môžu na ňom príležitostne podieľať. V prírode disponujú schopnosťou rozpoznať umiestnenie koristi a obkľúčiť ju, avšak pri simulácii tohto správania, nemáme vedomosť o presnej polohe koristi. Preto na základe prvých troch najlepších riešení vypočítame predpokladanú polohu koristi, čím vznikne vzorec 27. Kandidáti, vrátane omega vlkov, potom na základe rovníc 20, 22 a 24 aktualizujú svoju

polohu [21].

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \quad (27)$$

Útok na korisť v matematickom modeli dosiahneme približovaním sa ku korisťi, čiže znížením hodnoty \vec{a} . Tak dosiahneme aj zníženie hodnoty \vec{A} až bude jeho hodnota v intervale $[-1, 1]$. Potom ďalšia vypočítaná hodnota pozície vlka sa bude nachádzať medzi jeho pôvodnou polohou a polohou korisťi. V prípade, že hodnota \vec{A} sa nachádza mimo spomínaného intervalu, vlk diverguje od korisťi, čím je reprezentovaná fáza hľadania korisťi. Tým je zabezpečené prehľadávanie priestoru agentami [21].

Princíp optimalizácie svorkou divých vlkov môžeme znázorniť v nasledujúcom pseudokóde [21]

Algorithm 2 Pseudokód optimalizácie svorkou divých vlkov.

- 1: Náhodná inicializácia vlkov
 - 2: Inicializácia a , A a C
 - 3: Vyhodnotenie fitness funkcie pre každého jedinca
 - 4: Priradenie do 3 najlepších riešení do X_α , X_β resp. D_δ
 - 5: Opakuj kroky 5 až 10 pokiaľ nebude prekročený maximálny počet iterácií
 - 6: Aktualizovanie polohy na základe rovníc 20, 22 a 24
 - 7: Aktualizovanie a , A a C
 - 8: Vyhodnotenie fitness funkcie pre každého jedinca
 - 9: Aktualizovanie X_α , X_β a D_δ
 - 10: Aktualizovanie počtu iterácií
 - 11: Vráť optimálne riešenie X_α
-

2.3.3 Umelá kolónia včiel

V literatúre označovaný ako ABC algoritmus (angl. Artificial bee colony) je pomerne nový medzi rojovými algoritmi. Princíp je založený na biologickom procese správania medonosných včiel pri hľadaní potravy. Každá včela pracujúca v roji sa spolupodieľa na tvorbe celého systému na globálnej úrovni. Správanie systému je určené lokálnym správaním, kde spolupráca a zladenie jedincov vedie k štruktúrovanému kolaboračnému systému [5].

V algoritme, kolónia umelých včiel pozostáva z 3 typov. Sú to včely robotnice (angl. employed bees), prehľadávačky (angl. onlookers) a prieskumníčky (angl. scouts). Informácie o zdrojoch potravy si vymieňajú tancovaním (angl. waggle dance) na tzv. tanečnej ploche. Prehľadávačky čakajúce na tanečnej ploche sa rozhodujú, ktorý zdroj potravy si vyberú. Robotnice navštevujú zdroje potravy, ktoré v minulosti už navštívili. Úlohou včiel prieskumníček je vykonávať náhodné prehľadávanie priestoru. Pri aplikovaní ABC algoritmu, polovica včiel predstavuje robotnice, druhá polovica prehľadávačky. Počet robotníc zodpovedá počtu zdrojov potravy v okolí úľu. Robotnica, ktorej zdroj je vyčerpaný inými včelami sa stáva prieskumníčkou [11].

Zväčšovaním množstva nektáru jednotlivých zdrojov potravy sa zvyšuje aj pravdepodobnosť výberu prehľadávačkou. Tým je zabezpečené, že tanec robotníc, ktoré navštívili zdroje s najväčším množstvom nektáru, presvedčí najviac prehľadávačok. Tie na základe informácie o polohe zdroja nájdu v jeho okolí nový zdroj, z ktorého budú zberať nektár. Po vyčerpaní nektáru je tento zdroj opustený a včela sa presunie na zdroj, ktorý našli prieskumníčky [11].

Pozícia zdrojov jedla je reprezentácia možného riešenia daného problému. Množstvo nektáru je úmerné kvalite riešenia, ktoré je určené fitness funkciou. Počet robotníč a prehľadávačov je rovný počtu riešení v danej populácii, pričom každé je reprezentovateľné ako D -dimenzionálny vektor. Hodnota D môže predstavovať počet optimalizačných parametrov [11].

Včely majú riešenie uložené vo vlastnej pamäti, z ktorého modifikáciou vznikajú nové, ktoré následne vyskúšajú. Ak je nové riešenie lepšie ako predchádzajúce, včely si ho zapamätajú a predchádzajúce zabudnú. Inak si pamätajú staré. Keď všetky robotnice skončia proces hľadania, zdieľajú informácie o zdrojoch s prehľadávačkami. Tie vyhodnotia zhromaždené informácie a vyberú zdroj s najlepším riešením a jeho modifikovaním vytvoria nové, ktoré následne skontrolujú [11].

Prehľadávačky vyberajú zdroj v závislosti od pravdepodobnosti zdroja p_i , ktorú vypočítame ako

$$p_i = \frac{fit_i}{\sum_{j=1}^n fit_j} \quad (28)$$

Vo vzorci 28 predstavuje fit_i fitness funkciu, ktorá ohodnocuje riešenie i . Týmto spôsobom si vymieňajú medzi sebou informácie robotnice a prehľadávačky [11].

Algoritmus počíta nové riešenie na základe starého, čo matematicky zapíšeme ako

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) \quad (29)$$

Ak si počet včiel definujeme ako B potom na základe vzorca 29 platí $k \in 1, 2, \dots, B$ a $j \in 1, 2, \dots, D$. Indexy k a j sú vybrané náhodne, ale k musí byť rôzne od i . Hodnota ϕ_{ij} je z intervalu $[-1, 1]$ vybraná náhodne. Je zrejmé, že čím bude menší rozdiel medzi $(x_{ij}$ a $x_{kj})$, tým bude menší rozdiel aj medzi starým a novým riešením [11].

Algoritmus má 3 konfigurovateľné parametre a to: počet robotníč alebo prehľadávačov, ktorý je rovný počtu zdrojov jedla, limit určujúci počet cyklov, po ktorom bude zdroj opustený v prípade nevylepšenia pozície a maximálny počet cyklov, ktorý sa vykoná ak sa skôr nenájde optimálne riešenie [11].

Princíp umelej kolónie včiel znázorňuje nasledujúci pseudokód [11]

Algorithm 3 Pseudokód umelej kolónie včiel.

- 1: Inicializácia včiel na náhodné zdroje potravy
 - 2: Opakuj kroky 2 až 7 pokiaľ nebude dosiahnutý cieľ
 - 3: Umiestnenie robotníč na zdroje potravy, ktoré boli nájdené
 - 4: Vyhodnotenie zdroja na základe množstva nektáru
 - 5: Umiestnenie prehľadávačov na zdroje potravy na základe získaných informácií
 - 6: Vyhodnotenie, ktoré včely sa stanú prieskumníčkami
 - 7: Vyslanie prieskumníčok do prehľadávania priestoru za účelom objavenia nových zdrojov potravy
 - 8: Vráť optimálne riešenie
-

2.4 Meranie presnosti predpovede

Pre vyhodnotenie efektívnosti a presnosti modelov je potrebné merať ich vlastnosti tak, aby sme ich vedeli medzi sebou porovnávať. V nasledujúcich spôsoboch merania sú použité pojmy ako aktuálna hodnota y_t , predpovedaná hodnota f_t alebo chyba predpovede e_t definovaná ako $e_t = y_t - f_t$. Veľkosť testovacej množiny budeme označovať ako n [1].

2.4.1 Stredná chyba predpovede

V literatúre označovaná ako MFE (angl. mean forecast error). Matematickú funkciu môžeme zapísať ako

$$MFE = \frac{1}{n} \sum_{t=1}^n e_t \quad (30)$$

Týmto spôsobom meriame priemernú odchýlku predpovedanej hodnoty od aktuálnej. Zistíme tak smer chyby. Nevýhodou je, že kladné a záporné chyby sa vynulujú a potom nie je možné zistiť presnú hodnotu chyby. Pri nameraní extrémnych chýb nedochádza k žiadnej špeciálnej penalizácii. Taktiež hodnota chyby závisí od škály meraní a môže byť ovplyvnená aj transformáciami dát. Dobré predpovede majú hodnotu blízku 0 [1].

2.4.2 Stredná absolútna chyba

V literatúre označovaná ako MAE (angl. mean absolute error). Patrí k jedným z najpoužívanjších. Funkciu môžeme zapísať ako

$$MAE = \frac{1}{n} \sum_{t=1}^n |e_t| \quad (31)$$

Týmto spôsobom meriame priemernú absolútnu odchýlku predpovedanej hodnoty od aktuálnej. Zistíme tak celkový rozsah chyby, ktorá nastala počas predpovede. Na rozdiel od merania chyby pomocou vzorca 30 sa kladné a záporné chyby nevynulujú, no ani napriek tomu nevieme určiť celkový smer chyby. Na druhej strane tiež nenastáva žiadna penalizácia pri extrémnych chybách. Hodnota chyby závisí od škály meraní a transformácií dát. Dobré predpovede majú hodnotu čo najbližšiu 0 [1, 10].

2.4.3 Stredná percentuálna chyba

V literatúre označovaná ako MPE (angl. mean percentage error). Matematicky môžeme túto funkciu zapísať ako

$$MPE = \frac{1}{n} \sum_{t=1}^n \frac{e_t}{y_t} \times 100 \quad (32)$$

Vlastnosti sú veľmi podobné ako pri MAPE v časti 2.4.4. Chyba nám poskytuje prehľad o priemernej chybe, ktorá sa vyskytla počas predpovede. Navyše oproti MAPE získame prehľad o smere chyby, čo má však za následok, že opačné znamienka sa vynulujú. O modeli, ktorého chyba MPE sa blíži k 0, nemôžeme s určitosťou tvrdiť, že funguje správne [1].

2.4.4 Stredná absolútna percentuálna chyba

V literatúre označovaná ako MAPE (angl. mean absolute percentage error). Vzorec, ktorým ju zapíšeme bude veľmi podobný vzorcu 32

$$MAPE = \frac{1}{n} \sum_{t=1}^n \left| \frac{e_t}{y_t} \right| \times 100 \quad (33)$$

Pomocou tohto merania chyby získavame percentuálny prehľad o priemernej absolútnej chybe, ktorá sa vyskytla počas predpovedi. Veľkosť chyby nezávisí od škály merania, ale je závislá od transformácií dát. Tiež nie je možné zistiť smer chyby a ani nenastáva žiadna penalizácia pri extrémnych chybách [1].

2.4.5 Stredná štvorcová chyba

V literatúre označovaná ako MSE (angl. mean squared error). Vzorcom ju zapíšeme ako

$$MSE = \frac{1}{n} \sum_{t=1}^n e_t^2 \quad (34)$$

Chyba meria priemernú štvorcovú odchýlku predpovedanej hodnoty. Opačné znamienka sa neovplyvňujú. Neposkytuje nám pohľad na smer chyby. Zabezpečuje penalizáciu extrémnych chýb. Zdôrazňuje fakt, že celková chyba je viac ovplyvnená jednotlivými veľkými chybami ako viacerými malými. Nevýhodou je, že chyba je veľmi citlivá na zmenu škály alebo transformáciu dát [1].

2.5 Zhodnotenie analýzy

S rastúcim množstvom dát z rôznych zdrojov vzniká potreba predpovedať správanie týchto veličín v budúcnosti. Existuje množstvo článkov, ktoré sa zaoberajú týmto problémom. Neustále prinášajú a vylepšujú existujúce matematické modely, ktoré sú schopné predpovedať budúce hodnoty meraných veličín. Výsledky nie sú presné, ale chyba výpočtu je dostatočne nízka na to, aby sme takýto výsledok mohli považovať za relevantný. Zvýšiť presnosť výsledkov je možné viacerými metódami, či už kombináciou niekoľkých matematických modelov alebo ich optimálnym nastavením. My sme sa v práci zamerali na hľadanie optimálneho nastavenia používaných modelov.

Preskúmané predikčné modely potrebujú pre svoje fungovanie správne hodnoty vstupných parametrov. Pomocou nich vieme ovplyvniť chybu predikcie. Problémom je nájdenie takých parametrov, pre ktoré by bola chyba predikcie čo najmenšia. Hľadanie najlepšieho riešenia by bolo časovo neprípustné, a preto budeme hľadať iba optimálne riešenie, ktoré nám poskytne prijateľnú chybu predikcie. Je dôležité poznamenať, že vstupné parametre, ktoré nájdeme optimálnu predikciu závisia aj od samotných dát. Ako bolo spomenuté v sekcii 2.1.1, dáta pozostávajú z viacerých zložiek a ich pomer sa môže líšiť pri rôznych veličinách.

Na hľadanie optimálneho nastavenia predikčných algoritmov sme použili prírodne inšpirované optimalizačné algoritmy. Ako ich názov napovedá, hľadanie optimálneho riešenia sa vykonáva na základe správania sa nejakého živočíšneho druhu alebo prírodného javu. Tieto algoritmy sa osvedčili ako efektívne a rýchle riešenie problémov, ktorých prehľadávaný priestor riešení nie je možné prehľadať celý. Algoritmy sa vyhýbajú spadnutiu do lokálnych miním a tak je nájdené riešenie obvykle optimálne v globálnom rozsahu. A taktiež tieto algoritmy poskytujú niekoľko konfiguračných parametrov ovplyvňujúce rýchlosť a presnosť nájdeného riešenia. Avšak cieľom tejto práce je optimalizovať predikčné a nie optimalizačné algoritmy a poskytnúť tak používateľovi univerzálne a jednoduché riešenie predikčných problémov.

Dôležitou súčasťou predpovedí je vyhodnotenie ich úspešnosti porovnávaním predpovedanej hodnoty so skutočnými. Existuje množstvo metód, ktorými vieme zmerať chybu predpovede. Niektoré nám poskytujú informáciu o smere chyby, iné zohľadňujú extrémne chyby. Výsledky niektorých sa viažu na škálu, v ktorej sa nachádzajú naše merania, iné sú nezávislé, merané v percentách. Naším cieľom bolo poskytnúť používateľovi čo najviac informácií o presnosti predpovede, a preto sú použité viaceré metódy merania chýb predikcií.

3 Špecifikácia požiadaviek

Cieľom práce je navrhnúť systém, ktorý umožní používateľovi nájsť optimálne nastavenie vstupných parametrov predikčných metód. Používateľ bude mať k dispozícii niekoľko predikčných a optimalizačných algoritmov, bude môcť meniť ich konfiguračné parametre a vykonávať výpočty, vďaka ktorým zistí okrem veľkosti predikčnej chyby aj optimálne nastavenie parametrov. Tieto výsledky môže použiť vo vlastnej aplikácii alebo výskume. Predmetom optimalizácie budú parametre predikčných metód, konfigurácia optimalizačných algoritmov je ponechaná na používateľovi. Množinu dát si sám zvolí, aplikácia je nezávislá na doméne, v ktorej boli dáta namerané. Ku každému z uvedených parametrov bude poskytnuté vysvetlenie a opis zmeny správania na základe ich zmeny. Presnosť predpovede bude určená pomocou používaných metrík na meranie chyby predikcie. Keďže metrík je viacero, používateľ bude mať možnosť si vybrať jednu z nich a tá bude následne použitá optimalizačným algoritmom.

Pri návrhu aplikácie boli použité dáta z inteligentných elektromeračov používaných na Slovensku. Nameraná hodnota bude reprezentovaná reálnym číslom. Našou úlohou bolo optimalizovať parametre predikčných metód na základe reálne nameraných hodnôt a preto aplikácia nepracuje so žiadnymi vysvetľujúcimi premennými. Jej fungovanie je nezávislé na doméne, z ktorej dáta pochádzajú. Používateľ tak môže optimalizovať predikčné parametre aj na základe iných dát ako z elektromeračov. Stačí, ak každé meranie je jednoznačne identifikovateľné dátumom a časom zhotovenia.

Používateľ si bude môcť vyberať z ponúknutých predikčných a optimalizačných algoritmov. K dispozícii bude mať stochastické modely, regresiu založenú na podporných vektoroch, rozhodovacie stromy a prírodne inšpirované optimalizačné algoritmy ako umelú kolóniu včiel alebo optimalizáciu založenú na správaní roja častíc. Systém bude navrhnutý tak, aby pridanie ďalších algoritmov nespôsobovalo zmeny v pôvodnej aplikácii a pozostávalo tak iba z pridania implementácie algoritmov a prípadne zmeny konfigurácie. Pripadá do úvahy, že samotný systém bude poskytovať rozhranie, ktoré používateľovi umožní použiť vlastnú implementáciu iných algoritmov.

Optimalizačné algoritmy potrebujú mať určený definičný obor. Hodnoty z neho používajú na nájdenie optimálneho riešenia, čo je v našom prípade nájdenie kombinácie vstupných parametrov predikčných metód, ktorých výsledná chyba je čo najmenšia. Z tohto dôvodu je dôležité myslieť pri návrhu aplikácie na to, aby výstupom zvolených metód na meranie chyby boli nezáporné čísla. V aplikácii sa budú nachádzať prednastavené definičné obory, ktoré bude môcť používateľ v definovanom rozmedzí upravovať. Predídeme tak dlhým odozvám systému pri zvolení veľkého definičného oboru a zároveň nevzniknú situácie, kedy používateľ zvolí hodnotu, ktorá nebude patriť do definičného oboru predikčnej metódy.

Systém bude implementovaný v jazyku R a webové používateľské rozhranie bude vytvorené pomocou knižnice Shiny. Web aplikácie bude navrhnutý tak, aby používateľovi poskytoval interaktívne rozhranie s predikčnými metódami. Bude dostatočne robustný a intuitívny, aby s jeho používaním nemali problémy ani tí menej skúsení. Zároveň bude skúsenému používateľovi poskytovať modulárnosť, konfigurovateľnosť, ale hlavne rozhranie pre jednoduché pridávanie ďalších algoritmov, ktoré si môže sám implementovať.

Spomenuté vlastnosti môžeme rozdeliť na funkcionálne a nefunkcionálne nasledujúcimi odrážkami.

Funkcionálne požiadavky:

- **webová aplikácia** implementovaná pomocou jazyka R a knižnice Shiny

- **vkladanie dát** používateľom v preddefinovanom formáte
- **výber predikčnej metódy** z implementovaných metód závisí od používateľovej voľby
- **výber optimalizačnej metódy** ovplyvňuje rýchlosť a optimálnosť vypočítaného výsledku
- **chyba predpovedi** je použitá pri optimalizácii ako fitness funkcia
- **vyhodnotenie výpočtu** zobrazením veľkosti chyby, nájdených optimálnych parametrov a vykreslením grafu

Nefunkcionálne požiadavky:

- **efektívnosť** implementácie je kľúčová pre zabezpečenie rýchleho výpočtu výsledku
- **jednoduchosť** použitia webovej aplikácie aj napriek množstvu parametrov, ktoré môže používateľ meniť
- **modulárnosť** dovoľuje používateľovi pridávať vlastné predikčné a optimalizačné algoritmy
- **konfigurovateľnosť** aplikácie bude zabezpečená viacerými konfiguračnými súbormi

4 Návrh riešenia

Výsledná webová aplikácia bude používateľovi poskytovať možnosť zvoliť si optimalizačný algoritmus, ktorým bude optimalizovať nastavenie predikčného algoritmu. Keďže každý predikčný algoritmus potrebuje pre správnu funkcionálnosť iné parametre s rôznymi hodnotami, je najskôr potrebné implementovať rozhranie, ktoré zabezpečí správnu interakciu s optimalizačným algoritmom. Používateľ preto bude vyberať iba z algoritmov, ktoré budú predom implementované alebo tak sám urobí.

V práci sme sa rozhodli zvoliť ako predikčné algoritmy metódu podporných vektorov, autoregresný integrovaný model kľzavého priemeru a náhodné lesy. Aplikácia je však navrhnutá genericky a pridanie ďalšieho algoritmu vyžaduje minimálne úsilie. Medzi zvolené optimalizačné algoritmy patrí optimalizácia rojom častíc a ABC optimalizácia. Používateľ bude môcť pomocou rozhrania určiť parametre pre tieto algoritmy, ako napríklad počet častí alebo cyklov.

4.1 Architektúra aplikácie

Pri vytváraní webovej aplikácie pomocou knižnice Shiny sa program rozdeľuje na časť webového rozhrania a serverovú časť. Tie sa pri jednoduchých aplikáciách môžu nachádzať v jednom súbore avšak my sme ich podľa odporúčaní dokumentácie rozdelili do súborov *ui.R* a *server.R* pre webové rozhranie a serverovú logiku. Konfiguračné súbory, pomocou ktorých sa do aplikácie jednoducho pridávajú ďalšie algoritmy sa nachádzajú v adresári *conf*. Zdrojové kódy, ktoré zabezpečujú úpravu dát a výpočet sa nachádzajú v adresári *src*. Keď sa podrobnejšie pozrieme na obsah tohto adresára zistíme, že súbory sú očíslované prefixom, ktorý zodpovedá logickému poradiu ich používania, ale zároveň sú v tomto poradí aj importované pri spustení Shiny servera.

Predikčné algoritmy sú implementované v súboroch s prefixami *01* a *03* za čím nasleduje skrátený názov implementovaného algoritmu. Rozdelením implementácie do dvoch súborov sa oddelí príprava dát na výpočet od samotného výpočtu. Názvy funkcií sú potom pomocou konfiguračného súboru namapované na jednotlivé grafické komponenty vo webovom rozhraní. Názov funkcie má vždy ako prefix skrátený názov algoritmu, ktorý ho používa. Podobná konvencia je dodržaná aj pri optimalizačných algoritmoch s tým, že prefix súborov je *04*. Volanie optimalizačných algoritmov zväčša pozostáva z jedného volania funkcie, a preto nevzniká potreba delenia na viacero súborov. Načítanie dát zo súboru a rôzne fitness funkcie sa nachádzajú v osobitnom súbore.

Aplikácia je navrhnutá tak, aby pridávanie ďalších algoritmov vyžadovalo čo najmenšie úsilie. Kvôli tomu sú grafické komponenty vykresľované zo súboru *server.R* a *ui.R* ako je to pri stránkach so statickým, prípadne jednoducho dynamickým rozložením komponentov. Vykresľované sú na základe obsahu konfiguračného súboru *server-config.yml*, ktorý v sebe obsahuje mapovanie na použité metódy a na základe vstupov z grafického rozhrania, kde si používateľ volí používané algoritmy. V závislosti od toho sa mení počet zobrazovaných komponentov, ich obsah a význam. Komponenty vykresľované serverom by sme mohli rozdeliť na parametre optimalizačných metód a rozsahy prehľadávania riešenie pre predikčné algoritmy.

Používateľské rozhranie webovej aplikácie je veľmi jednoduché a intuitívne. Používateľ najskôr vloží vstupné dáta, nad ktorými budú vykonávané predikcie. Dáta by nemali byť veľké, aby optimalizácia netrvala príliš dlho, ale zároveň aby vzorka dát bola dostatočne reprezentatívna. Následne môže meniť predikčné a optimalizačné metódy, ich parametre,

fitness funkciu či veľkosti trérovacej a testovacej množiny. V momente, keď mu nastavenie vyhovuje, môže spustiť výpočet. Jeho výstupom je optimálne nájdená hodnota a zodpovedajúce riešenie. Pre lepšie znázornenie je priložený obrázok 10, na ktorom je diagram opisujúci spomínaný postup.



Obr. 10: Stavový diagram aplikácie.

4.2 Vlastnosti vstupných dát

Vstupné dáta, ktoré vkladá do aplikácie používateľ, by mali byť vo formáte CSV, teda súbor s tabuľkovými dátami, kde hodnoty v stĺpcoch sú oddelené čiarkami. Súbor by mal pri tom

obsahovať aspoň dva pomenované stĺpce, kde stĺpec *timestamp* obsahuje dátum a čas získania hodnoty, ktorá sa nachádza v stĺpci *value*. Hodnoty v ostatných stĺpcoch budú ignorované. Zámerom bolo, aby zápis dátumu a času zodpovedal medzinárodnému štandardu ISO 8601. Kvôli zložitosti čítania času v takomto formáte v použitom implementačnom prostredí sme zvolili vlastný formát dátumu, ktorý mu je veľmi podobný. Formát dátumu je *%Y-%m-%d %H:%M:%S*, podrobný rozbor sa nachádza v tabuľke 3. Z pohľadu návrhu aplikácie je potrebné, aby vstupné dáta mali rovnaký počet meraní nameraných v jeden deň v celej vstupnej množine. To znamená, že veľkosť periódy dát by mala byť rovnaká a zároveň sa rovnala veľkosti periódy, ktorú zvolil používateľ vo webovej aplikácii.

Tabuľka 3: Rozbor formátu dátumu a času.

Názov	Formátový reťazec	Hodnota
Rok	%Y	2013
Mesiac	%m	07
Deň	%d	10
Hodina	%H	20
Minúta	%M	15
Sekunda	%S	15

* Dátum z tabuľky je 2013-07-10 20:15:00

Stĺpec s hodnotou obsahuje reálne číslo, ktoré reprezentuje veľkosť nameranej veličiny, ktorú sa bude aplikácia snažiť predpovedať. Jednotky, v ktorých budú hodnoty uvedené nie sú podstatné, lebo výpočet je nezávislý na použitej škále. Ako oddeľovač desatinných miest bude použitá bodka. Obmedzenia pre stĺpec s hodnotou sú, aby skutočne obsahoval iba reálne číslo zodpovedajúce nameranej hodnote a aby v rámci jedného súboru bola konzistentne použitá rovnaká škála hodnôt. V stĺpci sa tak nesmie nachádzať názov či skratka jednotky, v ktorej je hodnota nameraná.

V prípade, že pri vkladaní súboru aplikácia nenájde spomínané stĺpce, nedovolí používateľovi spustiť výpočet. Pri návrhu aplikácie sa predpokladalo, že vstupné dáta budú vždy pochádzať z jedného zariadenia a budú teda mať rovnakú časovú zónu. Z tohto dôvodu stĺpec *timestamp* neobsahuje údaj o časovej zóne, v ktorej bola hodnota nameraná. Dáta, ktoré sme používali pri návrhu aplikácie preto museli byť orezané tak, aby všetky dni mali rovnaký počet meraní alebo čas musel byť prekonvertovaný do takej časovej zóny, ktorá by nespôsobovala rôznu veľkosť periódy. Keďže množina dát, s ktorými sme pracovali bola dostatočne veľká zvolili sme prvú možnosť.

Dáta by samozrejme nemali byť náhodné, ale malo by sa jednať o časový rad, ktorý je možné dekomponovať na cyklickú, trendovú a náhodnú zložku. Niektoré predikčné algoritmy priamo používajú dekompozíciu časových radov na predpovedanie hodnôt. Jednotlivé záznamy o meraniach by mali za sebou chronologicky nasledovať. V rámci jedného vstupného súboru musia mať dáta všetky záznamy rovnakú periódu časového radu. Ak používateľ vo webovom rozhraní zvolí, že vložené dáta majú periódu napríklad 96 meraní za deň, potom aj aplikácia bude očakávať zvolenú veľkosť periódy. Ak pri overovaní dôjde k zlyhaniu aplikácie ohlásí chybu.

4.3 Použité predikčné metódy

Z pohľadu návrhu aplikácie nás budú pre každú predikčnú metódu zaujímať najmä vstupné parametre, ktoré budeme optimalizovať a transformácie, ktoré je potrebné nad vstupnými

dátami vykonať. Takéto delenie je dôležité aj kvôli organizácii programu pri implementácii, keďže úprava dát sa vykoná iba raz, zatiaľ čo vstupné parametre sa budú meniť neustále. Výstupom výpočtu predpovede bude samotná predpoveď alebo dosiahnutá veľkosť chyby.

4.3.1 Regresia založená na podporných vektoroch

Parametre, ktoré budeme optimalizovať budú C a ε , ktoré môžu nadobúdať iba kladné hodnoty. Hodnota parametra C reprezentuje cenu za prekročenie hranice, ktorú sme definovali. Hranicu si označíme ako ε . Pri veľmi nízkej hodnote sa model nenatrénuje dostatočne, výsledkom čoho je predikcia s veľkou chybou. Pri veľmi vysokej hodnote môže dôjsť k pretrénovaniu modelu a zvýšeniu výslednej chyby na množine testovacích dát. Hodnota parametra ε je hodnota použitá v nesenzitívnej stratovej funkcii, ktorá sa používa pre regresiu. Zvyšovaním tejto hodnoty sa funkcia reprezentujúca predpoveď postupne vyhladzuje, až sa bude podobáť na konštantnú funkciu.

Vstupné dáta je potrebné transformovať na 2 matice a síce jednu trénovaciu maticu a jednu testovaciu maticu. Trénovacia matica obsahuje stĺpec s hodnotami, ktoré boli v daných časoch namerané, testovacia má tento stĺpec prázdny, hodnoty doplní algoritmus. Počet ďalších stĺpcov sa môže líšiť v závislosti od veľkosti zvolenej periódy. Ak si označíme počet meraní za deň ako n , potom stĺpce M_1 až M_n budú obsahovať hodnotu 1 práva vtedy, keď poradie merania v dni bude totožné s poradím stĺpca. Rovnakým postupom pridáme ešte ďalších 7 stĺpcov, ktoré budú predstavovať poradie dňa v týždni. Pre iné stĺpce D_1 až D_7 bude potom platiť, že napríklad v prvom stĺpci bude priradená hodnota 1 tým záznamom, ktoré boli namerané v pondelok. Zvyšok matice bude doplnený nulami.

4.3.2 Autoregresný integrovaný model kľzavého priemeru

Parametre stochastického modelu ARIMA sú v literatúre označované ako p , d a q . Z kapitoly 2.2.2 vieme, že model ARIMA je generalizáciou modelu ARMA, ktorý vznikol kombináciou autoregresného modelu a modelu kľzavých priemerov. Vstupné parametre sú nezáporné celé čísla, kde p a q predstavuje rad modelu a d zodpovedá stupňu derivovania. Pri veľkom počte derivovaní sa funkcia predpovedanej hodnoty podobá lineárnej funkcii.

Príprava vstupných dát je veľmi jednoduchá. Keďže model pracuje priamo s časovými radmi a vstupné dáta sú časový rad, jedinou úpravou je rozdelenie dát na trénovaciu a testovaciu množinu a určenie veľkosti periódy, ktorú zadáva používateľ.

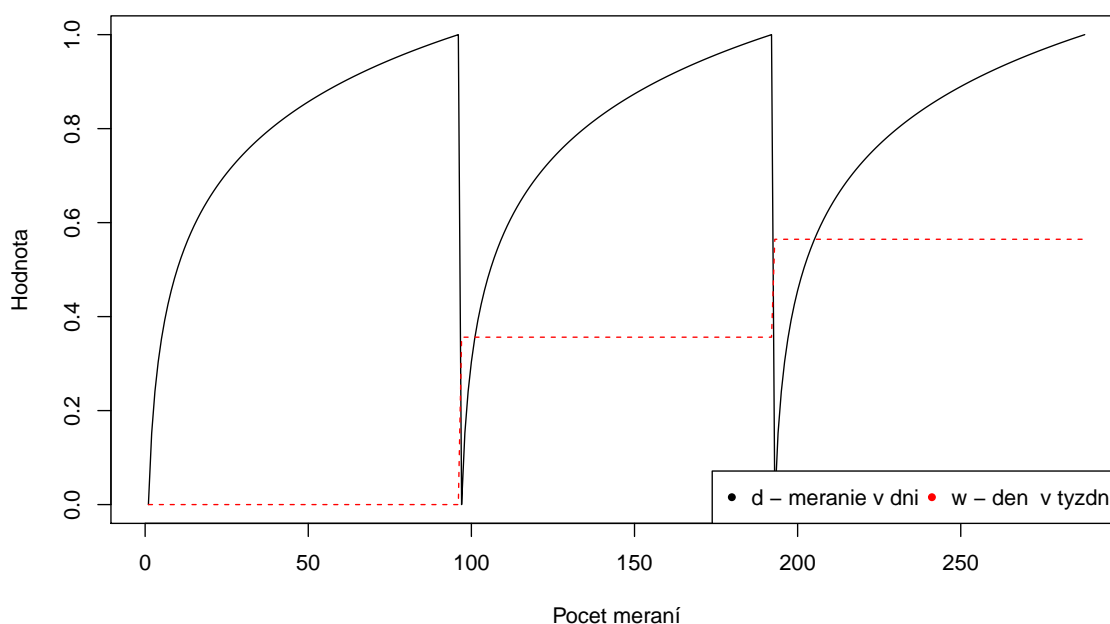
4.3.3 Náhodné lesy

Vstupné parametre, ktoré sa budeme snažiť v aplikácii optimalizovať budú počet stromov a minimálna veľkosť uzla v stromoch. Je zrejmé, že číslo definujúce počet stromov bude musieť byť kladné a celé, zatiaľ čo veľkosť uzla môže byť definovaná ako ľubovoľné nezáporné číslo. Pri zvyšovaní počtu stromov, ktoré budú vytvorené sa funkcia predpovedanej hodnoty javí, akoby každé pridanie stromu poskytovalo funkcii ďalšiu možnosť zmeny smeru. Funkcia tak pri malom počte stromov vyzerá nerovnomerne a náhodne, zväčšovaním počtu sa extrémny vyhladzujú a predpovedané hodnoty postupne konvergujú do funkcie, ktorá je obrysami podobná funkcii reálnych dát. Zväčšovaním uzlov sa funkcia predikovaných hodnôt postupne vyhladzuje, až nakoniec opisuje konštantnú funkciu.

Rovnako ako model ARIMA tak aj náhodné lesy pracujú priamo s časovými radmi. Je však potrebné pridať k dátam informáciu o tom, kedy boli namerané. Zároveň by táto funkcia mala byť spojitá. To najjednoduchšie dosiahneme tak, že z dátumu extrahujeme poradie dňa

v týždni a z časovej zložky zasa poradie merania v dni. Funkcie zatiaľ nie sú spojité, a preto na ne aplikujeme transformáciu popísanú vo vzorci 35 [14] a goniometrické funkcie sínus a kosínus. Výsledné funkcie sú spojité a svojou harmonickosťou opisujú vlastnosti dát skryté v dátume a čase. Na obrázku 11 môžeme vidieť priebehy funkcií pred transformáciou. Kvôli čitateľnosti je pre hodnoty použitá logaritmická mierka. Priebehy pomocných funkcií po transformácii sú znázornené na obrázku 12. Grafy vychádzajú zo vzorky dát o veľkosti 3 dni a a znázorňujú rovnaké dni.

$$\frac{\sin(2\pi \frac{\text{poradie merania}}{\text{veľkosť periódy}}) + 1}{2} \text{ resp. } \frac{\cos(2\pi \frac{\text{poradie merania}}{\text{veľkosť periódy}}) + 1}{2} \quad (35)$$

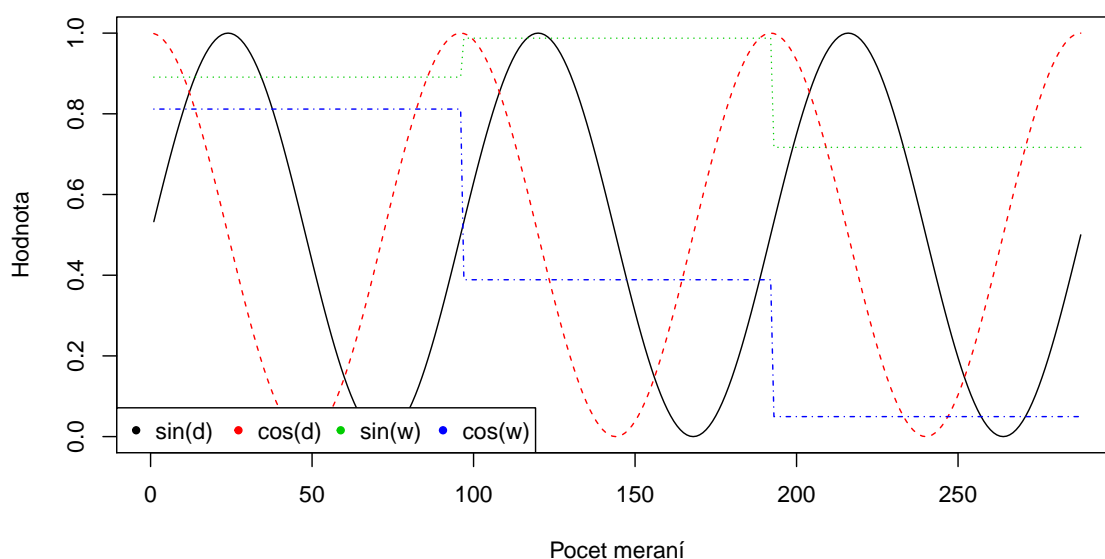


Obr. 11: Ukážka dátumu a času pred transformáciou.

4.4 Použité optimalizačné metódy

Aplikácia bude používateľovi poskytovať rozhranie na nastavenie vstupných parametrov predikčných metód a taktiež nastavovanie rozsahu hodnôt, ktoré majú tieto metódy použiť pri optimalizovaní. Výsledkom výpočtu bude porovnanie reálnych a predpovedaných dát v podobe grafu a veľkosti výslednej chyby. Taktiež budú používateľovi zobrazené optimálne parametre pre zvolenú predikčnú metódu. Každá kombinácia parametrov vystupuje v optimalizačnom algoritme ako agent, ktorého úlohou je dosiahnuť čo najnižšiu hodnotu fitness funkcie. V závislosti od použitého algoritmu a získaných vedomostí sa v nasledujúcich cykloch menia hodnoty reprezentujúce agenta. Hodnoty parametrov môžu byť ovplyvňované aj voľbou funkcie na meranie chyby, ktorá plní úlohu fitness funkcie pre optimalizačné algoritmy.

Pri návrhu aplikácie je dôležité najmä správne definovať rozhranie medzi predikčnými a optimalizačnými metódami. Je nežiadúce, aby pri pridávaní ďalších algoritmov bolo potrebné upravovať už používané funkcie, najmä kvôli množstvu kombinácií, ktoré môže vzniknúť. Našťastie každého agenta definujeme práve hodnotami, ktoré vstupujú do predikčného algoritmu. Na výstupe očakáva optimalizačná metóda hodnotu fitness funkcie, ktorá určuje ďalšie



Obr. 12: Ukážka transformácií dátumu a času pri predikcii náhodnými lesmi.

smerovanie agentov. Tento proces sa opakuje až pokiaľ nebude dosiahnutá ukončovacia podmienka. V použitých algoritmoch to je maximálny počet iterácií algoritmu. Túto hodnotu môže používateľ nastaviť v aplikácii v definovanom rozsahu.

4.4.1 Optimalizácia rojom častíc

Používateľ bude mať možnosť v aplikácii zvoliť počet častíc, ktoré budú použité na optimalizovanie problému. Pod každou časticou si môžeme predstaviť agenta, ktorý bude nositeľom informácie o okolitom prostredí a o hodnotách, ktoré ho viedli k aktuálnemu riešeniu. Väčší počet častíc znamená dôkladnejšie prehľadanie vstupného priestoru. Na druhú stranu výpočet je náročnejší. Pre každý optimalizovaný parameter je potrebné definovať rozsah hodnôt, ktoré môže používateľ v aplikácii nastaviť. Je dôležité poznamenať, že okrem hraníc rozsahu nás zaujímajú aj samotné hodnoty, ktoré bude algoritmus používať na optimalizáciu. Je zrejmé, že používať desatinné čísla pre parameter označujúci napríklad počet stromov nie je logické a aplikácia by mala byť schopná sa s tým vysporiadať.

V prostredí jazyka R sú k dispozícii 2 rôzne knižnice implementujúce optimalizáciu rojom častíc. V aplikácii sa budú pre porovnanie použité obidve implementácie.

4.4.2 Umelá kolónia včiel

Rovnako ako pri optimalizácii rojom častíc tak aj pri tomto algoritme bude mať používateľ možnosť zvoliť rozsah hodnôt, ktoré budú použité na optimalizovanie. Opäť bude dôležité vysporiadať sa s desatinnými číslami. Používateľ bude môcť navyše nastavovať parametre ABC algoritmu ako je počet zdrojov potravy alebo limit pre daný zdroj. Pod limitom môžeme rozumieť počet iterácií cyklu, počas ktorého sa daný zdroj potravy nezlepšil. Týmto parametrom môžeme výrazne urýchliť nájdenie optimálneho riešenia, na druhej strane môže nastať situácia, kedy by sme rýchlym opustením zdroja potravy mohli stratiť lepšie riešenie. Zvyšovaním počtu zdrojov potravy sa zvyšuje aj čas výpočtu, ale zároveň bude nájdené optimálnejšie riešenie.

5 Implementácia

Webová aplikácia je implementovaná v jazyku R verzie 3.4.0. Pri implementácii sme používali vývojové prostredie RStudio verzia 1.0.143. Webové rozhranie je vytvorené pomocou knižnice *shiny*. Konfigurácia aplikácie je zabezpečená knižnicou *config*, informácie o ďalších použitých knižniciach a ich verziách sa nachádzajú v tabuľke 4. Nasledujúce podkapi-toly obsahujú popis použitých transformácií nad dátami, popis organizácie zdrojového kódu do funkcií a súborov, úpravu konfigurácie, ale aj postupy na pridanie ďalších predikčných a optimalizačných algoritmov.

Tabuľka 4: Použité knižnice jazyka R.

Názov	Použitá verzia
ABCOptim	0.14.0
config	0.2
dygraphs	1.1.1.4
forecast	8.0
kernlab	0.9-25
pso	1.0.3
psoptim	1.0
randomForest	4.6-12
shiny	1.0.2
shinyBS	0.61
shinyjs	0.9
xts	0.9-7

5.1 Úprava vstupných dát

Počas návrhu aplikácie bolo k dispozícii viacero datasetov, ktoré sme mohli použiť. Keďže je fungovanie aplikácie nezávislé na zvolenej doméne a nevzniká potreba trénovať model na veľkom množstve dát, jediným kritériom na vstupné dáta je, aby mali vlastnosti časových radov. Použité dáta pochádzajú z inteligentných elektromeračov nachádzajúcich sa na Slovensku. Veličina, ktorú sme predpovedali je odber elektrickej energie. Dáta sú k dispozícii každých 15 minút a celý dataset obsahuje merania z obdobia takmer 2 rokov. Formát dát je nasledovný:

```
DATUM, CAS, Suma_odbery
01/07/2013, -105, 1979.647687
01/07/2013, -90, 1866.069666
01/07/2013, -75, 1657.762122
01/07/2013, -60, 1646.840106
01/07/2013, -45, 1654.063798
01/07/2013, -30, 1719.126487
01/07/2013, -15, 1725.385136
01/07/2013, 0, 1702.126518
01/07/2013, 15, 1653.233316
01/07/2013, 30, 1601.772426
01/07/2013, 45, 1577.106243
```

```
01/07/2013, 60, 1553.637892
01/07/2013, 75, 1557.940842
01/07/2013, 90, 1523.56265
```

Môžeme si všimnúť, že vstupný CSV súbor pozostáva z 3 stĺpcov, ktorých názvy sú *DATUM*, *CAS* a *Suma_odbery*. Hodnoty času pritom nezodpovedajú formátu, na ktorý sme bežne zvyknutý, hlavne preto, že sa jedná o jedno celé číslo, ktoré nadobúda aj záporné hodnoty. Je to z dôvodu, že toto číslo označuje poradie merania v dni v minútach v časovej zóne UTC, čiže záporné hodnoty tohto stĺpca označujú prvé hodiny v dni. Pri letnom čase sú to dve hodiny a pri zimnom iba jedna. Ako bolo spomenuté v kapitole 4, aplikácia nebude závislá od domény, časovej zóny alebo pomocných premenných, no zároveň je potrebné zachovať konzistentnú veľkosť periódy naprieč celým datasetom. Z toho dôvodu sa nám naskytujú dva spôsoby, ktorými môžeme upraviť dátum a čas do požadovaného formátu.

Prvým je odstránenie nadbytočnej hodiny pri prechode na zimný čas, pridanie chýbajúcej hodiny pri prechode na letný čas (napr. opakovaním hodnôt z predchádzajúcej hodiny) a následne prekonvertovať stĺpec *CAS* do formátu *%H:%M:%S*, s ktorým už ďalej vieme bez problémov pracovať. Druhým jednoduchším spôsobom je obmedziť používanie aplikácie na dáta s veľkosťou rádovo v týždňoch. Optimalizácia parametrov pri väčších dátach je aj tak časovo veľmi náročná. Potom môžeme záporné hodnoty presunúť do predchádzajúceho dňa a následne ich konvertovať spolu s dátumom do požadovaného formátu. Týmto spôsobom stratíme iba jeden deň na začiatku a na konci časového radu. Cenou za to je posunutie sledovanej veličiny spolu s posunutím času. Na výsledných dátach sa to prejaví tak, že vzory, ktoré si môžeme všimnúť v dátach z letného času, budú v zimnom čase o hodinu posunuté. Vstupná množina dát však nikdy nebude väčšia ako pár mesiacov, a tak nie je problém vyhnúť sa dvom dátumom v roku. Navyše tieto dáta boli použité iba pri návrhu a používateľ bude môcť použiť dáta, ktoré sám uzná za vhodné. Výsledný formát dát je potom nasledovný:

```
timestamp, value
2013-07-01 00:00:00, 1702.126518
2013-07-01 00:15:00, 1653.233316
2013-07-01 00:30:00, 1601.772426
2013-07-01 00:45:00, 1577.106243
2013-07-01 01:00:00, 1553.637892
2013-07-01 01:15:00, 1557.940842
2013-07-01 01:30:00, 1523.56265
```

Dáta v takomto formáte sú vhodné na spracovanie, avšak nie je možné ich priamo použiť predikčnými algoritmami. Každý z nich očakáva, že pracuje s dátami v rôznych dátových štruktúrach s rôznymi vysvetľujúcimi premennými. Aj keď bolo spomenuté, že aplikácia ich nebude používať v prípade, že vstupné dáta obsahujú viacero stĺpcov, v tomto prípade sa jedná o dekomponovanie zložiek časového radu a transformovanie dátumu a času tak, ako to požaduje konkrétny algoritmus.

Ako už bolo spomenuté SVR požaduje trénovacie aj testovacie dáta v štruktúre matice, ktorá bude až na stĺpec s nameranou alebo predpovedanou hodnotou obsahovať iba hodnoty 0 a 1. Po vytvorení matice, ktorá obsahuje iba 0, sa postupne na ňu aplikujú metódy *svr.setOnesForTimestamp* a *svr.setOnesForDayOfWeek*, ktoré priradia hodnotu 1 pre stĺpec reprezentujúci poradie merania v dni a rovnako aj pre stĺpec reprezentujúci poradie dňa v týždni, kedy bolo meranie uskutočnené. Poslednou úpravou je pridanie stĺpca s nameranými hodnotami pre trénovaciu maticu a prázdneho stĺpca pre testovaciu maticu. Implementácia spomínaných funkcií je:


```
svr.setOnesForTimestamp <- function(matrixM, dates,
  ↪ recordsCount, measurementsPerDay) {
  for (i in 1:recordsCount) {
    matrixM[i, svr.orderOfTimestamp(dates[i],
      ↪ measurementsPerDay) + 1] <- 1
  }
  return(matrixM)
}
```

```
svr.setOnesForDayOfWeek <- function(matrixM, dates,
  ↪ recordsCount, measurementsPerDay) {
  for (i in 1:recordsCount) {
    matrixM[i, measurementsPerDay + svr.nthInWeek(dates[i])
      ↪ ] <- 1
  }
  return(matrixM)
}
```

Pri použití modelu ARIMA je transformácia vstupných dát jednoduchá. Model požaduje dáta v štruktúre časového radu, v ktorej už samotné dáta sú. Príprava dát potom spočíva iba v rozdelení množiny na časti požadované používateľom. Implementácia je nasledovná:

```
trainingTS <- ts(preparedData$trainingData$value, frequency
  ↪ = preparedData$measurementsPerDay)
testingTS <- ts(preparedData$testingData$value, frequency =
  ↪ preparedData$measurementsPerDay)
```

V prípade transformácie dát pre náhodné lesy, by sme mohli hovoriť o kombinácii predchádzajúcich dátových štruktúr s miernym vylepšením. Pri SVR vzniká matica s veľkosťou desiatok stĺpcov a stoviek riadkov, ktoré v každom riadku obsahujú práve dve hodnoty 1, zvyšok sú samé 0. Dáta v takomto formáte je zložité graficky znázorniť a pritom sa jedná iba o 2 neklesajúce funkcie. Toto správanie by sme vedeli zachytiť aj dvojicou čísel, ktoré by reprezentovali rovnaké funkcie. Ak by sme však chceli získať spojité funkcie, tak ako to požaduje implementácia náhodných lesov, je potrebné aplikovať na dáta mapovanie znázornené obrázkom 12. Harmonické funkcie potom opisujú pozíciu merania v dni a týždni, čo implementujeme nasledujúcimi funkciami:

```
rf.setSinForDay <- function(dates, frequencyF) {
  return((sin(2 * pi * rf.nthInDay(dates, frequencyF) /
    ↪ frequencyF) + 1) / 2)
}
```

```
rf.setCosForDay <- function(dates, frequencyF) {
  return((cos(2 * pi * rf.nthInDay(dates, frequencyF) /
    ↪ frequencyF) + 1) / 2)
}
```

```
rf.setSinForWeek <- function(dates, frequencyF) {
  return((sin(2 * pi * rf.nthInWeek(dates, frequencyF) /
    ↪ frequencyF) + 1) / 2)
```

```

}

rf.setCosForWeek <- function(dates, frequencyF) {
  return((cos(2 * pi * rf.nthInWeek(dates, frequencyF) /
    ↪ frequencyF) + 1) / 2)
}

```

5.2 Použitie knižnice Shiny

Webová aplikácia, s ktorou prichádza používateľ do styku je vytvorená pomocou knižnice Shiny v jazyku R. Zdrojový kód býva obvykle organizovaný do súborov *ui.R* pre grafické komponenty a *server.R* pre logiku skrytú za nimi. Zároveň je možné vykresľovať jednotlivé komponenty aj zo serverovej časti aplikácie na preddefinované miesto v *ui.R*. Takéto použitie má zmysel v prípade, že zobrazované komponenty sa budú dynamicky pridávať a odoberať v závislosti od zvolených možností. Tento príklad presne zodpovedá problému, s ktorým sme sa stretli pri implementácii aplikácie. Zobrazovanie jednotlivých parametrov závisí od predom zvolených metód. Navyše kvôli zachovaniu konfigurovateľnosti je potrebné vlastností komponentov čítať z konfiguračného súboru, čím strácame možnosť sa na ne jednoducho odkazovať. Zobrazovanie komponentov priamo z *server.R* povolíme volaním funkcie *uiOutput("predictionParameters")* v *ui.R*. Následne je potrebné priradiť komponenty do premennej *output\$predictionParameters* v serverovej časti. To zabezpečíme nasledujúcim zdrojovým kódom:

```

output$predictionParameters <- renderUI({
  numberOfParameters <- length(server.properties$
    ↪ predictionAlgorithms[[as.numeric(input$
    ↪ predictionAlgorithms)]]$predictionParameters)
  fluidRow(
    lapply(1:numberOfParameters, function(i) {
      column(5,
        numericInput(server.properties$predictionAlgorithms[[
          ↪ as.numeric(input$predictionAlgorithms)]]$
          ↪ predictionParameters[[i]]$id,
        label = server.properties$predictionAlgorithms[[as.
          ↪ numeric(input$predictionAlgorithms)]]$
          ↪ predictionParameters[[i]]$label,
        value = server.properties$predictionAlgorithms[[as.
          ↪ numeric(input$predictionAlgorithms)]]$
          ↪ predictionParameters[[i]]$value,
        min = server.properties$predictionAlgorithms[[as.
          ↪ numeric(input$predictionAlgorithms)]]$
          ↪ predictionParameters[[i]]$min,
        max = server.properties$predictionAlgorithms[[as.
          ↪ numeric(input$predictionAlgorithms)]]$
          ↪ predictionParameters[[i]]$max,
        step = server.properties$predictionAlgorithms[[as.
          ↪ numeric(input$predictionAlgorithms)]]$
          ↪ predictionParameters[[i]]$step
      )
    })
  )
}

```

```

    )
  })
)
})

```

- **numberOfParameters** počet parametrov zvoleného predikčného algoritmu
- **as.numeric(input\$predictionAlgorithms)** index zvoleného predikčného algoritmu pomocou, ktorého sa pristupuje do poľa algoritmov v konfiguračnom súbore
- **server.properties\$predictionAlgorithms** pole predikčných algoritmov v konfiguračnom súbore
- **lapply(1:numberOfParameters, function(i)** vytvorenie objektov typu *column* v cykle
- **predictionParameters[[i]]** pristupovanie k jednotlivým parametrom nachádzajúcim sa v konfiguračnom súbore

Po vykreslení komponentov vzniká nový problém, ktorým je získanie hodnôt, ktoré nastavil používateľ. Keďže nikdy nie sú vykreslené všetky komponenty a každý z nich má iný identifikátor, museli by sme sa pri dotazovaní najprv uistiť, či daný komponent existuje. To by nebolo najvhodnejšie riešenie, keďže zároveň by sme dosiahli aj množstvo duplicitných častí kódu a znížila by sa udržiateľnosť a čitateľnosť. Ak by sme problém riešili rovnakými identifikátormi stále by to neriešilo rôzne počty parametrov. Jediným rozumným riešením je opäť použiť odkazovanie na hodnoty v konfiguračnom súbore, čo je zabezpečené funkciou *setOptimizationParameters*. Kľúčovou časťou kódu je cyklus:

```

for (i in 1:predictParamsCount) {
  value <- eval(parse(text = paste("input", "$",
    ↪ predictParams[[i]]$id, sep = "")))

  if (grepl("^min", predictParams[[i]]$id)) {
    lows <- c(lows, value)
  }
  else if (grepl("^max", predictParams[[i]]$id)) {
    highs <- c(highs, value)
  }
}

```

- **predictParamsCount** počet parametrov zvoleného predikčného algoritmu
- **predictParams** pole parametrov predikčného algoritmu
- **value** hodnota, ktorú zadal používateľ, vzniká evaluovaním reťazca, ktorý spája používanú syntax a identifikátor komponentu z konfiguračného súboru
- **if** vzhľadom na to, že pri optimalizácii vždy určujeme rozsah použitých hodnôt, môžeme ich identifikovať prefixom *min* a *max* a oddeliť ich od seba

5.3 Konfigurácia aplikácie

Aplikácia je ľahko konfigurovateľná a rozšíriteľná aj vďaka konfiguračným súborom v adresári *conf*, ale aj súboru *global.R*, ktorý sa automaticky importuje pri spustení Shiny servera. Obsahuje informácie o cestách k súborom, ktoré aplikácia potrebuje pre správne fungovanie. Pri použití aplikácie na inom operačnom systéme ako Linux, prípadne s iným rozmiestnením súborov, je možné v tomto súbore jednoducho zmeniť prednastavené cesty.

Spomínaný konfiguračný adresár *conf* obsahuje ďalšie 3 súbory. Najdôležitejším je *server-config.yml* a možno ho logicky rozdeliť na 3 časti a to predikčné algoritmy, optimalizačné algoritmy a fitness funkcie. Každý z predikčných algoritmov obsahuje názvy parametrov, ktoré budú optimalizované, názvy funkcií, ktoré na to budú použité a názvy, ktoré budú zobrazované používateľovi pri používaní aplikácie. Parametre v sebe zahŕňajú aj informáciu o minimálnych, maximálnych a prednastavených hodnotách.

Ďalším konfiguračným súborom je *ui-config.yml*, ktorý obsahuje väčšinou texty a názvy jednotlivých komponentov, no nájdú sa tu aj hodnoty argumentov pre komponenty, ktoré sa počas behu aplikácie nemenia. Takto je zmena jazyku aplikácie veľmi jednoduchá a a vykonané zmeny by sa týkali iba týchto dvoch súborov. Posledným konfiguračným súborom je *app-config.yml*, kde sú umiestnené hodnoty, ktoré sa netýkajú webovej aplikácie, ale samotných algoritmov. Veľa z nich poskytuje väčšiu funkcionálnosť ako sme reálne použili. To vieme ovplyvniť pri volaní funkcie napríklad nastavením príznaku alebo konkrétnou hodnotou vymenovaného typu. Konkrétnym príkladom je predpovedanie pomocou SVR, ktorý sa bežne používa na klasifikačné problémy ako SVM. V súbore sa potom nachádza vymenovaný typ označujúci použitie regresie a nie klasifikácie.

5.4 Postup pridávania predikčných algoritmov

Najdôležitejšou časťou pridania ďalšieho algoritmu je implementovanie 3 funkcií, na ktoré sa neskôr budeme odkazovať z webového rozhrania. Prvá funkcia, v konfiguračnom súbore označovaná ako *prepareFn*, bude na vstupe očakávať výsledok načítania súboru, čiže zoznam s dátami z tréningovej a testovacej množiny so stĺpcami *timestamp* a *value* a počtom meraní za deň. Výstupom funkcie je opäť zoznam obsahujúci pripravené dátové štruktúry pre tréningové a testovanie množiny. Ten je uložený do globálnej premennej *params.prediction* zjednocujúcej parametre predikčnej metódy, ktoré nebudú optimalizované. Ďalšou funkciou je *predictFn*, ktorá je jadrom celého procesu optimalizácie. Bude používaná v každom cykle každým agentom, a preto je dôležité dbať na efektívnosť funkcie. Na vstupe očakáva pole číselných parametrov, ktoré budú optimalizované. Telo funkcie sa vo väčšine prípadov skladá z vytvorenia modelu, vypočítania predpovedi a vypočítania chyby, ktorú predpoveď dosiahla. Pri vytváraní modelu sú používané hodnoty z *prepareFn*, ale môžeme dodefinovať aj parametre, ktoré sa nebudú optimalizovať a vložiť ich do *app-config.yml*. Výpočet chyby spočíva vo volaní funkcie podľa jej názvu, ktorý je uložený v *params.prediction\$errorFn*. Parametrami sú skutočná hodnota z testovacích dát a predpovedaná hodnota. Pri pridávaní ďalších fitness funkcií je dôležité nezabudnúť, aby výsledkom bolo vždy nezáporné číslo, inak by mohla nastať situácia, kedy by optimalizačné algoritmy nachádzali nevhodné riešenia. Posledná funkcia *predictDataFn* je implementáciou rovnaká líši sa iba v tom, že namiesto veľkosti chyby vracia predpovedané hodnoty. Tie sú potom použité vo webovom rozhraní na vykreslenie grafu s reálnymi hodnotami a predpovedanými.

Samozrejme, ak by sme teraz spustili aplikáciu, nepribudla by možnosť použitia nového algoritmu. Dôležité je vytvoriť v konfiguračnom súbore *server-config.yml* nový ele-

ment v poli predikčných algoritmov. Pre správne fungovanie aplikácie je dôležité, aby element obsahoval nasledujúce premenné:

- **label** názov algoritmu používaný v UI
- **parameterLabels** názvy optimalizovaných parametrov používaných v UI
- **prepareFn** funkcia pripravujúca dáta pre predikciu
- **predictFn** predikčná funkcia vracajúca veľkosť chyby
- **predictDataFn** predikčná funkcia vracajúca predpoveď
- **predictionParameters** pole predikčných parametrov, pre každý parameter sa tu nachádza minimum a maximum, obsahuje nasledujúce hodnoty:
 - **id** identifikátor komponentu používaný na odkazovanie sa na serveri
 - **description** popis zvoleného parametra v UI
 - **label** názov parametra zobrazujúci sa v UI
 - **value** prednastavená hodnota
 - **min** dolná hranica parametra
 - **max** horná hranica parametra
 - **step** krok, o ktorý sa minimálne zmení hodnota parametra pri nastavovaní komponentu

Ani teraz by sa po spustení nevykonala výpočet. Aj keď máme k dispozícii voľbu nového algoritmu, po spustení výpočtu aplikácia zlyhá, lebo server sa pokúša použiť metódu, ktorá zatiaľ nebola importovaná. Preto je poslednou úpravou pridanie príkazu na importovanie vytvorených súborov do zdrojového súboru *server.R*.

Pri implementácii funkcií *predictFn* a *predictDataFn* je dôležité myslieť na to, pre aké vstupné hodnoty je predikčný algoritmus definovaný. Definičný obor určujeme rozsahom v konfiguračnom súbore, ale problém nastáva v prípade, ak je definovaný iba na množine celých čísel. Optimalizačné algoritmy budú volať funkciu aj s desatinnými číslami v rámci určeného rozsahu. Niektoré implementácie predikčných algoritmov sa s týmto problémom vedú vysporiadať a zaokrúhľujú desatinné čísla na celé podľa potreby. Niektoré nemajú ošetrené vstupy, a tak sa môže program začať správať nepredvídateľne alebo prestane reagovať úplne. Z toho dôvodu sme hodnoty, ktoré majú byť celočíselne vnútri týchto metód zaokrúhľovali. Najlepším riešením by bolo postihovať tie riešenia, ktoré nie sú celočíselné, ale to by mohlo spôsobiť rozsiahle zmeny v implementovaných funkciách.

5.5 Postup pridávania optimalizačných algoritmov

Pridanie ďalšieho optimalizačného algoritmu je jednoduchší proces, no vo svojej podstate pozostáva z rovnakých krokov ako pridanie nového predikčného algoritmu. Implementácia väčšinou pozostáva z jednej funkcie vnútri, v ktorej sa okrem optimalizácie iba pripravujú dáta pre predikčný algoritmus a transformujú výstupy optimalizačného algoritmu na jednotný formát. Optimalizácia pozostáva obvykle z jedného volania knižničnej funkcie. Vstupné parametre, ktoré sa nemenia, môžeme umiestniť do konfiguračného súboru. Ostatnými parametrami bývajú hranice, v rámci ktorých má algoritmus hľadať riešenie a parametre špecifické pre

použitý algoritmus. Výstupom tejto metódy je zoznam obsahujúci najnižšiu nájdenú chybu a pole parametrov, ktorými bola hodnota získaná.

Pred spustením aplikácie je potrebné vykonať kroky, ktoré boli opísané v predchádzajúcej kapitole. Konkrétne pridať implementovaný algoritmus do konfiguračného súboru, zvýšiť počet dostupných algoritmov a pridať importovania do súboru *server.R*. Pridaný záznam v konfiguračnom súbore bude mať nasledovný formát:

- **label** názov algoritmu zobrazujúci sa v UI
- **optimizeFn** optimalizačná funkcia vracajúca najlepšie nájdené riešenie
- **optimizationParameters** pole optimalizačných parametrov, pre každý parameter sa tu nachádza minimum a maximum, obsahuje nasledujúce hodnoty:
 - **id** identifikátor komponentu používaný na odkazovanie sa na serveri
 - **label** názov parametra zobrazujúci sa v UI
 - **description** popis zvoleného parametra v UI
 - **value** prednastavená hodnota
 - **min** dolná hranica parametra
 - **max** horná hranica parametra

6 Experimentálne overenie

Dáta, na ktorých bolo uskutočnené testovanie webovej aplikácie pochádzajú z inteligentných elektromerov, vďaka ktorým máme merania k dispozícii každú štvrtú hodinu. Merania sú združené podľa PSČ do súborov, kde každý z nich obsahuje 96 meraní za deň po dobu približne 1 roka a 11 mesiacov. Je potrebné najskôr vybrať reprezentatívne dáta s dĺžkou periódy, ktorá bude zabezpečovať dostatočne rýchly výpočet s uspokojujúcou veľkosťou chyby. Zároveň je potrebné eliminovať prvok náhodnosti a každé meranie vykonať niekoľko krát. Následne môžeme porovnávať jednotlivé optimalizačné metódy s primitívnymi metódami ako je napríklad prehľadávanie celého priestoru.

Pri testovaní sme používali tri datasety *94_nitra_sum.csv*, *95_partizanske_sum.csv* a *96_zvolen_sum.csv* nachádzajúce sa v adresári *data*. V nasledujúcich kapitolách budeme na ne odkazovať ako na datasety *D94*, *D95* a *D96*. Pri experimentoch sme používali trénovaciu a testovaciu množinu s veľkosťou 2 týždne. Každý experiment bol opakovaný 10 krát, výsledky boli spriemerované a bola vypočítaná ich štandardná odchýlka. V tabuľkách budeme používať ich skrátené označenie *AVG* (angl. average) pre priemer a *SD* (angl. standard deviation) pre smerodajnú odchýlku. Jednotlivé prípady testovania sme označovali v tabuľkách skrátené písmenom *T*. V kapitole B.1 sú opísané parametre optimalizačných a rozsahy predikčných algoritmov v tabuľkách 6, 7, 8 a 9. V tabuľke 10 označenie *tune* znamená použitie funkcie *tune.svm* rovnakými parametrami ako pri ostatných pokusoch. Rovnako označenie *auto* v tabuľke 11 znamená použitie funkcie *auto.arima*, ktorej parametre je možné zhora ohraničiť. Na označenie prehľadávania všetkých riešení pomocou cyklu sme použili skratku *all*. Tabuľky v kapitole B.2 obsahujú záznamy z experimentov, ktoré sú predmetom nasledujúcich kapitol.

Tabuľka 5: Použité datasety.

Názov súboru	Skrátený názov	Použité obdobie
94_nitra_sum.csv	D94	júl
95_partizanske_sum.csv	D95	august
96_zvolen_sum.csv	D96	september

6.1 Optimalizácia parametrov SVR

Na základe výsledkov uvedených v tabuľke 10, môžeme zhodnotiť, že biologicky inšpirované optimalizačné algoritmy dosiahli výsledky s menšou chybou za kratší čas v porovnaní s optimalizáciou pomocou funkcie *tune.svm*. Nájdené parametre boli vo väčšine prípadov rovnaké pri všetkých algoritmoch. Taktiež si môžeme všimnúť, že výsledky algoritmov PSO a ABC sú porovnateľné, aj keď vo viacerých prípadoch bol výsledok PSO nájdený rýchlejšie. Ako fitness funkcia bola použitá metóda MAE.

6.2 Optimalizácia parametrov modelu ARIMA

Dosiahnuté výsledky v tabuľke 11 sú pri metódach PSO, ABC a prehľadávaní celého priestoru takmer rovnaké, v ojedinelých prípadoch sa nájdené parametre mierne líšia. Doba výpočtu bola pri optimalizačných metódach PSO a ABC väčšia ako pri prehľadávaní celého priestoru, ktorý bol vzhľadom na nízky počet riešení malý. Najlepšie výsledky dosiahla funkcia *auto.arima*, ktorá okrem optimalizovaných parametrov nastavuje aj ďalšie koeficienty,

vďaka ktorým je výsledok presný a rýchlo vypočítaný. Okrem toho sa nájdené optimálne parametre líšia oproti ostatným metódam. Pri experimentoch sme použili ako fitness funkciu MAPE.

6.3 Optimalizácia parametrov náhodných lesov

Pri experimente boli optimálne parametre hľadané pomocou algortimov PSO, ABC a prehľadávaním celého priestoru. V tabuľke 12 môžeme vidieť výsledky. Nájdené optimálne parametre vo väčšine prípadov nadobúdali približne rovnaké hodnoty. Prehľadávanie celého vstupného priestoru a PSO dosiahli približne rovnaké veľkosti chyby, ale výpočet PSO bol rýchlejší pri zvolení väčších rozsahov optimálnych parametrov. Je to očakávané správanie, keďže PSO neprehľadáva všetky možné riešenia. Prekvapením boli výsledky algoritmu ABC, ktoré dosahujú väčšiu nameranú chybu a zároveň výpočet trval niekoľko násobne viac ako pri PSO. Na meranie veľkosti chyby bola použitá metóda MSE.

6.4 Porovnanie predpovedí s rôznymi fitness funkciami

V tabuľke 13 sú výsledky zoskupené v závislosti od použitej metódy na meranie chyby, konkrétne MAE, MAPE a MSE. Opäť sú výsledky nájdených optimálnych parametrov približne rovnaké a líšia sa iba v dĺžke trvania výpočtu. Rozdiely nie sú také zrejmé, napríklad MAE v porovnaní s MAPE je v priemere pomalšie iba o sekundu. Najrýchlejšie výsledky zabezpečila metóda MSE, ktorej výpočet bol rýchlejší vo väčšine prípadov.

7 Záver

Našou úlohou bolo vytvoriť webovú aplikáciu, ktorá bude používateľovi poskytovať možnosť výberu predikčných algoritmov, ktorých optimálne vstupné parametre budú nachádzané pomocou optimalizačných algoritmov. Zároveň bolo cieľom navrhnuť aplikáciu tak, aby pridávanie nových algoritmov nespôsobovalo vážne zmeny v jej implementácii. To sa nám do značnej miery podarilo zabezpečiť pridaním konfiguračných súborov a navrhnutím rozhrania medzi jednotlivými funkciami. Taktiež aplikácia spĺňa opísanú funkcionality a poskytuje používateľovi výsledky v podobe nájdených optimálnych parametrov. Vstupné dáta nie sú závislé od domény, v ktorej boli namerané. Musia však spĺňať formát stanovený v predchádzajúcich kapitolách. Používateľ má možnosť pridania ďalších algoritmov jednoduchým spôsobom s minimálnym zásahom do existujúcich zdrojových súborov.

Počas analýzy problému sme sa zaoberali jednotlivými predikčnými a optimalizačnými metódami, ich výhodami a nevýhodami. Zamerali sme sa najmä na biologicky inšpirované algoritmy, ktoré nám poskytujú v krátkom čase uspokojivé výsledky. Taktiež sme si spomenuli možnosti merania chyby predikcií, ktoré zároveň v našom prípade reprezentujú fitness funkciu optimalizačnej metódy. Pri návrhu sme zistili, že nie všetky metódy merania chyby sú vhodným kandidátom na fitness funkciu. Ďalším testovaním sme dokázali, že doba výpočtu sa pri použití rôznych fitness funkcií môže líšiť, aj keď nájdené parametre sú približne rovnaké.

Meraním a experimentami sme prišli k záveru, že niektoré predikčné algoritmy neposkytujú existujúce riešenie, ktoré by bolo efektívnejšie ako nami použité optimalizačné algoritmy. Pri regresii založenej na podporných vektoroch sme knižničné funkcie prekonali ako presnosťou, tak aj rýchlosťou nájdeného riešenia. Naopak pri autoregresnom integrovanom modeli kľavých priemerov sú použité optimalizačné algoritmy ťažkopádne vzhľadom na malú množinu optimalizovaných parametrov.

Literatúra

- [1] Adhikari, R.: *An Introductory Study on Time Series Modeling and Forecasting*. Saarbrücken: LAP LAMBERT Academic Publishing, 2013, ISBN 9783659335082.
- [2] Arun, K.; Rejimoan, R.: A survey on network path identification using bio inspired algorithms. In *2016 2nd International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB)*, Feb 2016, s. 387–389, doi:10.1109/AEEICB.2016.7538314.
- [3] Bel, L.; Allard, D.; Laurent, J. M.; aj.: CART Algorithm for Spatial Data: Application to Environmental and Ecological Data. *Comput. Stat. Data Anal.*, ročník 53, č. 8, Jún 2009: s. 3082–3093, ISSN 0167-9473, doi:10.1016/j.csda.2008.09.012.
URL <http://dx.doi.org/10.1016/j.csda.2008.09.012>
- [4] Breiman, L.: Random Forests. *Machine Learning*, ročník 45, č. 1, 2001: s. 5–32, ISSN 1573-0565, doi:10.1023/A:1010933404324.
URL <http://dx.doi.org/10.1023/A:1010933404324>
- [5] Chavan, S. D.; Kulkarni, A. V.; Khot, T.; aj.: Bio inspired algorithm for disaster management. In *2015 International Conference on Energy Systems and Applications*, Oct 2015, s. 776–781, doi:10.1109/ICESA.2015.7503455.
- [6] Deolekar, R. V.: Defining parameters for examining effectiveness of genetic algorithm for optimization problems. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, March 2016, s. 1061–1064.
- [7] Goel, L.; Gupta, D.; Panchal, V. K.; aj.: Taxonomy of nature inspired computational intelligence: A remote sensing perspective. In *2012 Fourth World Congress on Nature and Biologically Inspired Computing (NaBIC)*, Nov 2012, s. 200–206, doi:10.1109/NaBIC.2012.6402262.
- [8] Grmanová, G.; Laurinec, P.; Rozinajová, V.; aj.: Incremental Ensemble Learning for Electricity Load Forecasting. *Acta Polytechnica Hungarica*, ročník 13, č. 2, 2016.
- [9] Gruau, F.; Lyon I, L. C. B.; Doctorat, O. A. D. D.; aj.: Neural Network Synthesis Using Cellular Encoding And The Genetic Algorithm. 1994.
- [10] Gutiérrez, P. A.; Pérez-Ortiz, M.; Sánchez-Monedero, J.; aj.: Ordinal Regression Methods: Survey and Experimental Study. *IEEE Transactions on Knowledge and Data Engineering*, ročník 28, č. 1, Jan 2016: s. 127–146, ISSN 1041-4347, doi: 10.1109/TKDE.2015.2457911.
- [11] Karaboga, D.; Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, ročník 39, č. 3, 2007: s. 459–471, ISSN 1573-2916, doi:10.1007/s10898-007-9149-x.
URL <http://dx.doi.org/10.1007/s10898-007-9149-x>
- [12] Kennedy, J.; Eberhart, R.: Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, ročník 4, Nov 1995, s. 1942–1948 vol.4, doi:10.1109/ICNN.1995.488968.

- [13] Kumar Singh, A.; Khatoon, S.; Muazzam, M.; aj.: An Overview of Electricity Demand Forecasting Techniques. *Network and Complex Systems*, ročník 3, č. 3, 2013, ISSN 2225-0603.
URL www.iiste.org
- [14] Laurinec, P.: R j- Slovakia meetup started to build community in Bratislava. <https://petolau.github.io/First-R-Slovakia-meetup/>, 2017, dostupné 2017-04-26.
- [15] Lazinica, A.: *Particle swarm optimization*. Rijek, Croatia: InTech, 2009, ISBN 978-953-7619-48-0.
- [16] Liu, L.; Hudak, G.: *Forecasting and Time Series Analysis Using the SCA Statistical System*, ročník zv. 1. Scientific computing Associates Corporation, 1992.
URL <https://books.google.sk/books?id=8-HrAAAACAAJ>
- [17] Mahalakshmi, G.; Sridevi, S.; Rajaram, S.: A survey on forecasting of time series data. In *2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16)*, Jan 2016, s. 1–8, doi:10.1109/ICCTIDE.2016.7725358.
- [18] Mendes-Moreira, J. a.; Soares, C.; Jorge, A. M.; aj.: Ensemble Approaches for Regression: A Survey. *ACM Comput. Surv.*, ročník 45, č. 1, December 2012: s. 10:1–10:40, ISSN 0360-0300, doi:10.1145/2379776.2379786.
URL <http://doi.acm.org/10.1145/2379776.2379786>
- [19] Merz, C. J.: *Classification and Regression by Combining Models*. Dizertačná práca, University of California, 1998, aAI9821450.
- [20] Sapankevych, N. I.; Sankar, R.: Time Series Prediction Using Support Vector Machines: A Survey. *IEEE Computational Intelligence Magazine*, ročník 4, č. 2, May 2009: s. 24–38, ISSN 1556-603X, doi:10.1109/MCI.2009.932254.
- [21] Seeley, T. D.; Camazine, S.; Sneyd, J.: Collective decision-making in honey bees: how colonies choose among nectar sources. *Behavioral Ecology and Sociobiology*, ročník 28, č. 4, 1991: s. 277–290, ISSN 1432-0762, doi:10.1007/BF00175101.
URL <http://dx.doi.org/10.1007/BF00175101>
- [22] Sen, M.; Stoffa, P.: *Global Optimization Methods in Geophysical Inversion*. Advances in Exploration Geophysics, Elsevier Science, 1995, ISBN 9780080532561.
URL <https://books.google.sk/books?id=PT5MqSIvREMC>
- [23] Simonova, S.; Panus, J.: Genetic algorithms for optimization of thematic regional clusters. In *EUROCON 2007 - The International Conference on Computer as a Tool*, Sept 2007, s. 2061–2066, doi:10.1109/EURCON.2007.4400359.
- [24] Smola, A. J.; Schölkopf, B.: A tutorial on support vector regression. *Statistics and Computing*, ročník 14, č. 3, 2004: s. 199–222, ISSN 1573-1375, doi:10.1023/B:STCO.0000035301.49549.88.
URL <http://dx.doi.org/10.1023/B:STCO.0000035301.49549.88>
- [25] Tso, G. K.; Yau, K. K.: Predicting electricity energy consumption: A comparison of regression analysis, decision tree and neural networks. *Energy*, ročník 32, č. 9, 2007: s.

1761 – 1768, ISSN 0360-5442, doi:<http://dx.doi.org/10.1016/j.energy.2006.11.010>.

URL <http://www.sciencedirect.com/science/article/pii/S0360544206003288>

A Technická dokumentácia

A.1 Spustenie aplikácie

Jediným potrebným krokom na spustenie aplikácie je otvorenie obsahu elektronického média v RStudio a pri zvolení súboru *server.R* kliknúť na *Run App*, čím sa nám otvorí nové okno v prehliadači s aplikáciou. Odporúčaná verzia vývojového prostredia RStudio je 1.0.143.

A.2 Používateľská príručka

Obrazovku webovej aplikácie možno rozdeliť na časti:

- výber algoritmov
- informácie o vstupných dátach
- parametre optimalizačnej metódy
- rozsah parametrov predikčnej metódy
- vypočítaný výsledok

Používateľ môže podľa potreby zvoliť algoritmy, ktoré chce použiť, ich vstupné parametre, rozsahy, prípadne zmeniť fitness funkciu. Na obrázku 13 si môžeme v dolnej časti všimnúť tlačidlo, ktoré je zakázané. Je to spôsobené tým, že zatiaľ nevložil do aplikácie dáta, nad ktorými sa majú vykonávať predpovede. Po pridaní súboru sa objavia na obrazovke nové komponenty, ktoré slúžia na ohraničenie tréningových a testovacích dát. Môžeme ich vidieť na obrázku 14. V prípade, že sa polia pre výber dátumov nezobrazia, používateľ vložil nevalidné dáta, čiže môžu chýbať hlavička súboru, nemusí sa jednať o CSV alebo názvy stĺpcov nie sú *timestamp* a *value*. Po vložení validných dát sa spomínané komponenty vykreslia.

Optimalizácia konfiguračných parametrov predikčných metód

Aplikácia umožňuje používateľovi vybrať si predikčný a optimalizačný algoritmus, pomocou ktorých bude nájdená optimálna hodnota vybranej fitness funkcie. Používateľ má možnosť konfigurovať parametre optimalizačnej a interval jednotlivých parametrov predikčnej metódy. Nad dátami, ktoré vloží bude vypočítaná najmenšia odchýlka a parametre, ktoré k tomuto výsledku viedli. Dáta musia byť vo formáte CSV, kde prvý stĺpec timestamp označuje čas merania, a stĺpec value hodnotu tohto merania.

Predikčné algoritmy	Optimalizačné algoritmy
SVR	PSO z knižnice pso
Vstupné dáta	Počet meraní za deň
Vložiť... Nie je zvolený žiadny sú	96

Parametre optimalizačnej metódy

Optimalizácia rojom častíc, nastaviteľnými parametrami sú počet častíc a maximálny počet iterácií
počet častíc - priamo úmerný počtu preskúmaných riešení
maximálny počet iterácií - ukončovacia podmienka v prípade, že sa skôr nenájde optimálne riešenie

Stredná absolútna chyba	Počet častíc	Maximum iterácií
Fitness funkcia		
MAE	20	20

Parametre predikčnej metódy

Regresia založená na podporných vektoroch, optimalizovanými parametrami sú C a epsilon:
C - cena za prekročenie hranice
epsilon - hranica

Minimum C	Maximum C
100	1000
Minimum epsilon	Maximum epsilon
50	200

Vypočítaj

Obr. 13: Úvodná obrazovka webovej aplikácie.

Vstupné dáta	Počet meraní za deň	Rozsah tréningových dát	Rozsah testovacích dát
Vložiť... 94_nitra_sum.csv	96	2013-07-01 do 2013-07-14	2013-07-15 do 2013-07-28
Upload complete			

Obr. 14: Výber tréningovej a testovacej množiny.

Po vložení dát môže používateľ ďalej upravovať parametre, ktoré má k dispozícii. Pri zmene prednastavených rozsahov pre dátové množiny netreba zabudnúť, že čím bude väčšia testovacia množina dát, tým bude výsledná chyba menšia. Aplikácia môže predpovedať dopredu jednu periódu alebo jej celé násobky. Zmena množiny, z ktorej budú optimalizované parametre predikčnej metódy je priamo úmerná času, ktorý bude zabrať výpočet. Rovnako pri navýšení hodnôt parametrov optimalizačných metód ako je počet jedincov alebo iterácií, dochádza k získaniu presnejšieho výsledku za cenu dlhšej odozvy. Toto samozrejme nemusí platiť vždy a keďže používame algoritmy ako sú napríklad náhodné lesy, vypočítaný výsledok môže byť aj horší ako predchádzajúci. Nastavenie parametrov môžeme vidieť na obrázkoch 15 a 16.

Parametre optimalizačnej metódy

Optimalizácia rojom častíc,

nastaviteľnými parametrami sú počet častíc a maximálny počet iterácií

počet častíc - priamo úmerný počtu preskúmaných riešení

maximálny počet iterácií - ukončovacia podmienka v prípade, že sa skôr nenájde optimálne riešenie

Stredná absolútna percentuálna chyba

Fitness funkcia

MAPE

Počet častíc

20

Maximum iterácií

20

Obr. 15: Výber parametrov optimalizačnej metódy.

Parametre predikčnej metódy

Náhodné lesy,

optimalizovanými parametrami sú počet stromov a veľkosť uzla:

počet stromov - počet by nemal byť veľmi malé číslo

veľkosť uzla - väčšie hodnoty spôsobujú rast nízkych stromov

Minimum stromov

40

Maximum stromov

80

Minimálna veľkosť uzlov

5

Maximum veľkosť uzlov

20

Obr. 16: Výber parametrov predikčnej metódy.

Po kliknutí na tlačidlo *Vypočítaj* sa spustí výpočet a tlačidlo sa zakáže. Povolené je v momente ukončenia výpočtu, kedy sa používateľovi zobrazia výsledky. Výsledky pozostávajú z najmenšej chyby, ktorú zvolený algoritmus našiel, tabuľku obsahujúcu parametre predikčnej metódy, ktoré viedli k tomuto výsledku a graf zobrazujúci reálne a predpovedané hodnoty. Výstup aplikácie môžeme vidieť na obrázku 17.

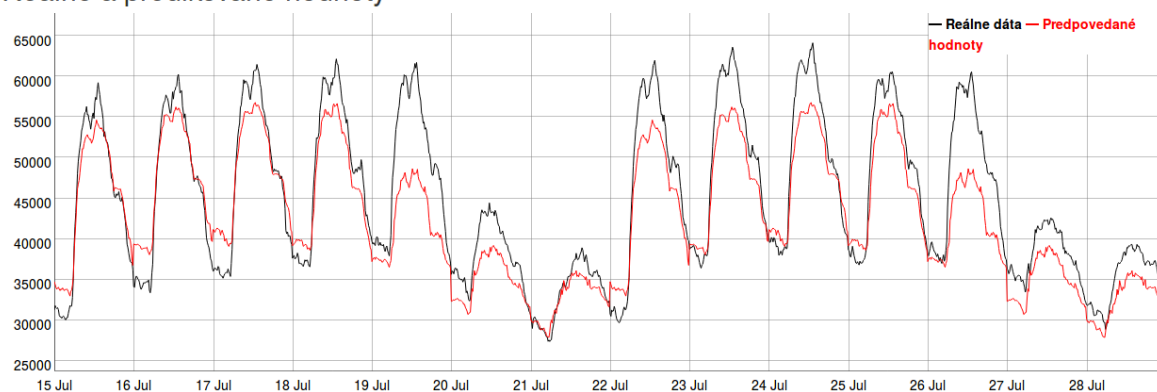
Optimálna konfigurácia parametrov

Názov parametru	Optimálna hodnota
Počet stromov	51
Veľkosť uzla	10

Najnižšia dosiahnutá chyba

Názov chyby	Veľkosť chyby
MAE	3309.21
MAPE	7.02
MSE	16504075.59

Reálne a predikované hodnoty



Obr. 17: Zobrazenie vypočítaného výsledku používateľovi.

A.3 Obsah elektronického média

```
CD nosič
├── conf
│   ├── app-config.yml
│   ├── server-config.yml
│   └── ui-config.yml
├── data
│   ├── 94_nitra_sum.csv
│   ├── 95_partizanske_sum.csv
│   └── 96_zvolen_sum.csv
├── doc
│   └── BP_MATUS_CUPER.pdf
├── shiny
│   ├── bound-dates.R
│   ├── set-labels.R
│   └── validate-input.R
├── src
│   ├── 00-read-data.R
│   ├── 01-arma-prepare.R
│   ├── 01-rf-prepare.R
│   ├── 01-svr-prepare.R
│   ├── 02-measure-error.R
│   ├── 03-arma-predict.R
│   ├── 03-rf-predict.R
│   ├── 03-svr-predict.R
│   ├── 04-abc-optimize.R
│   ├── 04-pso-optimize.R
│   └── 04-psoptim-optimize.R
├── test
│   ├── arima.R
│   ├── fitness.R
│   ├── rf.R
│   └── svr.R
├── util
│   └── convert.R
├── web
│   ├── plot.css
│   └── spinner.gif
├── global.R
├── server.R
└── ui.R
```

- **conf** adresár obsahujúci konfiguračné súbory aplikácie
- **data** vzorové vstupné dáta použité pri návrhu a testovaní webovej aplikácie
- **doc** dokumentácia, obrázky použité v nej a zdrojové súbory pre LaTeX
- **shiny** zdrojové súbory, ktoré používa Shiny server

- **src** implementácia rozhrania predikčných a optimalizačných algoritmov, implementovaná celá funkcionálna aplikácia
- **test** zdrojové súbory použité pri validácii riešenia
- **util** skripty použité pri návrhu
- **web** súbory pre štylizáciu webového rozhrania

B Popis experimentov

B.1 Nastavenie parametrov optimalizačných metód

Tabuľka 6: Parametre optimalizačných metód pri SVR.

Označenie	Maximum iterácií	Počet častíc/ zdrojov potravy	Limit	C	epsilon
T1	10	20	5	20-50	0-10
T2	20	50	10	50-200	0-5
T3	20	20	10	200-300	10-15
T4	20	20	10	400-500	10-30

Tabuľka 7: Parametre optimalizačných metód pri modeli ARIMA.

Označenie	Maximum iterácií	Počet častíc/ zdrojov potravy	Limit	p	d	q
T5	20	10	10	1-4	1-4	1-4
T6	10	20	5	0-3	0-3	0-3
T7	20	20	NA	0-3	1-5	0-3

Tabuľka 8: Parametre optimalizačných metód pri náhodných lesoch.

Označenie	Maximum iterácií	Počet častíc/ zdrojov potravy	Limit	Počet stromov	Veľkosť uzla
T8	100	20	20	5-20	40-80
T9	100	20	10	5-20	80-200
T10	100	20	10	1-40	1-250
T11	100	50	20	7-15	1-400

Tabuľka 9: Parametre PSO a náhodných lesov pri rôznych fitness funkciách.

Označenie	Maximum iterácií	Počet častíc/ zdrojov potravy	Limit	Počet stromov	Veľkosť uzla
T12	100	20	20	5-20	40-80
T13	100	20	10	5-20	80-200
T14	100	20	10	1-40	1-250
T15	100	50	20	7-15	1-400

B.2 Výsledky meraní

Tabuľka 10: Výsledky predpovedí SVR.

		Veľkosť chyby		C		epsilon	
Označenie	Čas	AVG	SD	AVG	SD	AVG	SD
T1-D1 PSO	198.18	8089.35	0	50	0	0.02	0.08
T1-D2 PSO	195.77	6459.48	0.06	50	0	9.74	0.83
T1-D3 PSO	202.36	3888.73	0	50	0	10	0
T2-D1 PSO	654.81	6825.41	0	200	0	2.50	2.64
T2-D2 PSO	637.71	4990.40	0	200	0	5	0
T2-D3 PSO	688.56	2786.12	0	200	0	5	0
T3-D1 PSO	270.28	6385.65	0	300	0	15	0
T3-D2 PSO	265.50	4668.64	0	300	0	15	0
T3-D3 PSO	261.56	2579.58	0.04	300	0	14.50	1.58
T4-D1 PSO	259.45	5981.96	0.01	500	0	10.10	0.32
T4-D2 PSO	259.86	4320.69	0	500	0	30	0
T4-D3 PSO	269.62	2316.15	0	500	0	10	0
T1-D1 ABC	279.25	8089.35	0	50	0	0	0
T1-D2 ABC	197.83	6459.46	0	50	0	10	0
T1-D3 ABC	203.54	3888.73	0	50	0	10	0
T2-D1 ABC	1115.34	6825.41	0	200	0	5	0
T2-D2 ABC	1132.44	4990.40	0	200	0	5	0
T2-D3 ABC	1165.40	2786.21	0	200	0	5	0
T3-D1 ABC	352.38	2579.57	0	300	0	15	0
T3-D2 ABC	338.10	4668.64	0	300	0	15	0
T3-D3 ABC	332.54	6385.65	0	300	0	15	0
T4-D1 ABC	571.09	5981.96	0	500	0	10	0
T4-D2 ABC	343.54	4320.69	0	500	0	30	0
T4-D3 ABC	554.57	2316.15	0	500	0	10	0
T1-D1 tune	231.89	8744.41	0	50	0	7.0	5
T1-D2 tune	224.60	7233.87	0	50	0	2.0	4
T1-D3 tune	248.20	4800.64	0	50	0	0.0	0
T2-D1 tune	1134.74	8570.76	0	200	0	50.0	0
T2-D2 tune	1111.89	7039.34	0.06	200	0	2.5	2.64
T2-D3 tune	1164.47	4570.84	0	200	0	5.0	0
T3-D1 tune	790.36	8450.19	0.72	300	0	12.5	2.64
T3-D2 tune	782.46	6914.44	0.62	300	0	13.5	2.41
T3-D3 tune	832.34	4418.10	0.07	300	0	11.5	2.41
T4-D1 tune	788.23	8248.55	0	500	0	10.0	0
T4-D2 tune	796.42	6670.58	3.18	500	0	24.0	9.66
T4-D3 tune	864.64	4123.90	0.05	500	0	16.0	9.66

Tabuľka 11: Výsledky predpovedí modelu ARIMA.

		Veľkosť chyby		p		d		q	
Označenie	Čas	AVG	SD	AVG	SD	AVG	SD	AVG	SD
T5-D1 PSO	204.28	15.38	0.03	2.9	1.45	2	0	2.30	1.06
T5-D2 PSO	120.39	16.93	0	3	1.05	1	0	1	0
T5-D3 PSO	314.51	20.72	0.07	3.9	0.32	2	0	3.20	0.42
T6-D1 PSO	73.64	13.95	0	0	0	0	0	2	0
T6-D2 PSO	72.76	16.93	0	2	0	1	0	1	0
T6-D3 PSO	80.88	16.82	0.07	2	0	0	0	1.10	0.32
T7-D1 PSO	167.29	15.41	0	1	0	2	0	1	0
T7-D2 PSO	144.30	16.93	0	2	0	1	0	1	0
T7-D3 PSO	297.09	21.10	0	3	0	2	0	2	0
T5-D1 ABC	623.31	15.36	0	4	0	2	0	3	0
T5-D2 ABC	561.59	16.93	0	3.5	0.85	1	0	1	0
T5-D3 ABC	585.02	20.70	0.01	4	0	2	0	3.14	0.38
T6-D1 ABC	214.58	13.95	0	0	0	0	0	1.80	0.42
T6-D2 ABC	189.69	16.93	0	2	0	1	0	1	0
T6-D3 ABC	252.23	16.79	0	2	0	0	0	1	0
T5-D1 all	143.59	15.37	0	4	0	2	0	4	0
T5-D2 all	144.21	16.93	0	1	0	1	0	4	0
T5-D3 all	141.24	20.73	0	4	0	2	0	4	0
T6-D1 all	45.76	13.95	0	0	0	0	0	2	0
T6-D2 all	42.15	16.93	0	2	0	1	0	1	0
T6-D3 all	44.87	16.79	0	2	0	0	0	1	0
T5-D1 auto	7.27	5.61	0	1	0	1	0	0	0
T5-D2 auto	10.82	8.72	0	2	0	1	0	1	0
T5-D3 auto	7.54	3.74	0	1	0	1	0	0	0
T6-D1 auto	7.95	5.61	0	1	0	1	0	0	0
T6-D2 auto	11.22	8.72	0	2	0	1	0	1	0
T6-D3 auto	7.58	3.74	0	1	0	1	0	0	0

Tabuľka 12: Výsledky predpovedí náhodných lesov.

		Veľkosť chyby		Počet stromov		Veľkosť uzla	
Označenie	Čas	AVG	SD	AVG	SD	AVG	SD
T8-D1 PSO	36.35	13740016	498648	10.94	2.90	50.59	6
T8-D2 PSO	36.38	13890649	296515	10.70	2.80	44.39	4.44
T8-D3 PSO	35.69	1911970	94941	12.39	4.68	44.57	2.77
T9-D1 PSO	28.49	15941559	914675	10.2	3.58	97.70	8.50
T9-D2 PSO	28.13	16000900	614517	8.82	2.91	91.57	9.12
T9-D3 PSO	26.81	2944067	219581	8.70	2.70	95.82	10.85
T10-D1 PSO	40.13	13272556	526159	9.88	3.58	24.01	13.89
T10-D2 PSO	49.09	12779244	198751	12.12	9.28	6.55	2.90
T10-D3 PSO	46.01	1414265	57612	18.72	9.22	8.74	5.37
T11-D1 PSO	70.78	12984574	332862	8.34	1.32	20.10	13.52
T11-D2 PSO	79.28	12619478	254532	8.88	3.12	4.16	4.57
T11-D3 PSO	76.74	1404289	97688	9.01	2.65	5.54	4.81
T8-D1 ABC	65.65	19652974	2820962	9.09	3.63	45.69	3.68
T8-D2 ABC	73.24	16251580	1396484	7.65	3.20	41.03	1.55
T8-D3 ABC	73.70	3552771	1275308	10.88	5.14	41.78	3.85
T9-D1 ABC	58.32	23134091	4241761	8.29	3.68	89.39	12.56
T9-D2 ABC	64.02	20505988	2381237	7.61	3.61	87.87	11.47
T9-D3 ABC	59.46	4943165	695312	10.57	4.31	80.49	1.45
T10-D1 ABC	129.27	17340051	1855974	11.42	6.09	8.52	11.20
T10-D2 ABC	160.80	14851327	847895	7.59	4.64	1.87	1.58
T10-D3 ABC	147.67	2125855	444861	17.84	9.28	1.81	2.55
T11-D1 ABC	254.12	18542940	2354286	10.46	2.56	18.59	16.57
T11-D2 ABC	264.87	14552133	1056753	10.03	2.81	1.98	1.59
T11-D3 ABC	261.21	2501470	654087	11.82	2.76	2.11	2.35
T8-D1 all	14.47	13715211	751291	11.4	3.03	42.90	2.77
T8-D2 all	12.51	14347959	225483	9.50	3.10	44.70	5.91
T8-D3 all	12.44	1937406	109873	12	4.67	44.10	5.09
T9-D1 all	35.69	15787615	925014	9.70	4.45	91.90	9.49
T9-D2 all	36.41	16035512	521129	10.50	4.86	86	4.90
T9-D3 all	36.08	2704401	301230	10.40	4.22	83.50	2.84
T10-D1 all	232.57	12641684	322005	6.40	5.06	22	18.83
T10-D2 all	233.35	12540910	148901	7.10	7.03	5.20	5.61
T10-D3 all	233.61	1368331	58355	20.20	9.64	4.20	3.85
T11-D1 all	75.50	13058262	423853	9.70	2.63	18.10	10.46
T11-D2 all	76.09	12764302	124171	10.80	2.44	4	1.94
T11-D3 all	75.50	1422195	69605	11.20	3.19	4.40	1.84

Tabuľka 13: Výsledky predpovedí náhodných lesov pri rôznych fitness funkciách.

		Veľkosť chyby		Počet stromov		Veľkosť uzla	
Označenie	Čas	AVG	SD	AVG	SD	AVG	SD
T12-D1 PSO	41.48	2867.64	67.82	10.80	3.22	47.15	3.51
T12-D2 PSO	44.14	3147.91	17.19	12.14	4.62	46.29	6.37
T12-D3 PSO	40.12	1079.45	36.21	10.95	3.84	46.50	4.99
T13-D1 PSO	30.53	3163.10	79.47	9.51	3.24	96.25	10.02
T13-D2 PSO	30.01	3302.72	29.57	11.36	4.02	87.83	8.66
T13-D3 PSO	32.41	1318.83	50.66	8.85	2.88	88.86	8.25
T14-D1 PSO	39.84	2843.41	48.98	11.50	4.6	47.34	6.46
T14-D2 PSO	37.32	3162.72	15.63	10.61	3.61	47.19	4.97
T14-D3 PSO	33.07	1073.6	27.58	13.49	2.29	46.13	3.50
T15-D1 PSO	70.91	2772.26	42.71	8.22	2.16	15.10	8.35
T15-D2 PSO	90.73	3089.62	11.73	9.43	2.03	11.27	7.41
T15-D3 PSO	92.84	919.14	21.94	10.49	2.66	10.29	6.67
T12-D1 PSO	32.63	6.16	0.16	9.84	2.44	50.31	6.79
T12-D2 PSO	39.81	8.31	0.03	16.13	3.19	45.08	4.26
T12-D3 PSO	38.58	4.29	0.13	11.89	2.55	45.46	4.67
T13-D1 PSO	28.16	6.67	0.17	7.87	2.17	94.92	9.57
T13-D2 PSO	34.24	8.64	0.06	11.09	2.57	92.38	4.80
T13-D3 PSO	32.49	5.36	0.12	8.56	2.92	88.27	10.07
T14-D1 PSO	41.35	6.18	0.09	11.65	2.58	50.71	3.69
T14-D2 PSO	45.38	8.33	0.02	15.17	2.08	46.82	4.23
T14-D3 PSO	40.04	4.30	0.10	12.01	2.01	45.79	3.40
T15-D1 PSO	65.34	5.96	0.09	8.71	1.46	16.47	7.80
T15-D2 PSO	89.74	8.28	0.05	11.22	2.22	20.18	15.44
T15-D3 PSO	84.87	3.67	0.12	9.58	3.02	4.96	4.90
T12-D1 PSO	37.59	13353466	308461	10.47	3.19	47.64	3.76
T12-D2 PSO	33.69	13858656	375048	8.22	2.82	45.78	4.66
T12-D3 PSO	35.11	1847898	78369	12.72	3.27	44.57	2.72
T13-D1 PSO	22.97	15669835	867725	10.45	3.84	98	13.95
T13-D2 PSO	26.19	15998991	513685	7.55	1.75	91.47	6.90
T13-D3 PSO	29.55	2731614	209610	7.97	3.19	91.17	6.65
T14-D1 PSO	31.48	13427976	393115	10.89	4.13	47.70	4.50
T14-D2 PSO	37.90	13963537	159169	9.29	3.20	43.79	3.10
T14-D3 PSO	39.61	1875933	93114	10.94	1.82	46.06	3.03
T15-D1 PSO	71.23	13008269	426929	10.69	2.53	17.53	10.88
T15-D2 PSO	81.57	12723553	203566	7.95	1	8.72	6.83
T15-D3 PSO	79.96	1406932	63047	8.27	2.21	5.84	5.44

C Plán do letného semestra

Poradie týždňa v letnom semestri	Popis plánovanej činnosti
1. týždeň	úprava vstupných dát na požadovaný formát, aplikovanie predikčných algoritmov
2. týždeň	aplikovanie predikčných a optimalizačných algoritmov
3. týždeň	implementácia optimalizačných algoritmov, ktorým chýba podpora knižníc v jazyku R
4. týždeň	implementácia optimalizačných algoritmov, tvorba kostry grafického rozhrania aplikácie
5. týždeň	tvorba jednotného rozhrania medzi aplikáciou a grafickým rozhraním
6. týždeň	implementácia grafického rozhrania a následne prepojenie s aplikáciou
7. týždeň	ošetrenie neplatných akcií vykonaných používateľom
8. týždeň	testovanie aplikácie na rôznych vstupných dátach používateľmi
9. týždeň	tvorba technickej dokumentácie aplikácie
10. týždeň	testovanie algoritmov, dokumentácia a porovnanie algoritmov
11. týždeň	tvorba prezentácie a príprava na obhajobu projektu

Iteratívnym vývojom sme sa snažili plniť určený plán do letného semestra. V prvej polovici semestra bola väčšina úsila venovaná návrhu a implementácii aplikácie tak, aby poskytovala používateľovi vysokú mieru konfigurovateľnosti a modulárnosti. Aj napriek slabej podpore v implementačnom prostredí sa nám to do značnej miery podarilo splniť. Vďaka tomu bolo nasledovné implementovanie nových algoritmov jednoduchým a rýchlym procesom. Zvyšok činnosti počas semestra bol venovaný najmä tvorbe grafického rozhrania, dokumentácii a v neposlednom rade testovaniu. Funkcionalita grafického rozhrania bola v niekoľkých iteráciách zväčšená. Zároveň s používaním aplikácie vznikala potreba ošetrovania nevalidných vstupov a akcií, čo bolo predmetom činnosti najmä v závere semestra. Proces dokumentácie a testovania prebiehal nezávisle od vývoja grafického rozhrania, vzhľadom na to, že jadro aplikácie implementované v prvej polovici semestra už nebolo menené.