UNIVERZITA J. E. PURKYNĚ V ÚSTÍ NAD LABEM
Přírodovědecká fakulta

Moderní metody zpracováni časových řad a sekvencí

KI/MMZCS?

RNDr. Petr Kubera, Ph.D.

Ústí nad Labem 2020

Kurz:	Moderní metody zpracováni časových řad a sekvencí
Obor:	Aplikovaná informatika (nmgr)
Klíčová slova:	Časové řady, neuronové sítě, sekvence
Anotace:	Kurz je zaměřen na praktické seznámení s moderními způsoby zpracování časových řad, signálů a sekvencí za použití volně dostupných nástrojů a knihoven v Pythonu a případně R. Kurz pokrývá oblast predikce časových řad a vyhledávání událostí nad signály a sekvencemi. Cílem je poskytnout studentům jak základní teoretické pochopení modelů, tak i ucelený přehled nástrojů pro práci s nimi.

Jazyková korektura nebyla provedena, za jazykovou stránku odpovídá autor.

Autor: RNDr. Petr Kubera, Ph.D.

Obsah

Úvodní slovo

Předmět je koncipován jako přehledový kurz v oblasti zpracování časových řad a sekvencí. Předmět volně navazuje na předměty z bakalářského studia *Časové řady* a *Úvod do strojového učení* a rozvíjí je. Dále rozšiřuje a konkretizuje znalosti získané v předmětu *SoftComputing*. Časovou řadu můžeme chápat jako diskrétní soubor hodnot nějaké veličiny, např. kurz akcií, v čase. Sledované veličiny mohou být opravdu velmi široké, může se jednat o nějaký biologický signál, či ekonomická data, nebo textová data (slova). Zde budeme předpokládat, že se jedná buď o ekonomická data, nebo textová. Samozřejmě lze celou řadu zde představených postupů použit pro modelování i dat jiných, biologických, meteorologických apod. Poznamenejme, že se zpracováním dat z biologických systémů se částečně seznámíte ve speciálním kurzu *Analýza signálu a obrazu v praxi I a II*.

Máme li nějakou časovou řadu, tak nás může zajímat její **popis**, např. zda a jak se mění některé veličiny popisující řadu v čase. Zda se nějaké děje zachycené v časové řadě cyklicky opakují. Dále též můžeme chtít danou časovou řadu modelovat, tj. provádět její **regresi**, či přímo **predikovat**. K těmto úlohám právě potřebujeme znát některé vlastnosti dané časové řady, neboť volba metod závisí na jejích vlastnostech. Další úlohou může být vyhledávání určitých částí, průběhů signálu, neboť mohou detekovat právě probíhající událost. Příkladem může být detekce útoků na počítačové sítě, kde jako signál považujeme informace předávané operačním systémem. V kurzu budeme postupovat od "tradičních metod" jako jsou AR|I|MA modely, přes rekurentní neronové sítě a jejich vylepšení.Zakončíme zpracováním přirozeného jazyka, kde si ukážeme nástroje pro převod slov na vektory významu, se kterými můžeme dále pracovat.

Vše budeme dělat za použití volně dostupných nástrojů a frameworků v Pythonu.

Splnění kurzu

Zápočet je možné získat za vypracovanou seminární práci, která obsahuje jak teoretickou část = rozbor problému a popis použitých metod, tak i vlastní praktickou část=implementace a použití daných nástrojů. Aplikaci je **nutné** si nechat schálit vyučujícím. Přepodkládá se možnost, po schválení, použít i technologie, které na semináři nebyly probrány.

Zde naleznete několik témat pro motivaci k Vašim seminárním pracím

Ukázky možných témat seminárních prací

1. Otestování stacionarity časové řady - implementace testů ADF, KPSS test a jejich rozbor
2. Ověření použití ARIMA modelů na predikci reálných ekonomických dat
3. Porovnání LSTM a GRU modelů pro krátkodobou predikci počasí
4. Doporučení služby na základě analýzy sentimentu v komentářích

1 Časové řady a jejich vlastnosti

Před výkladem konkrétních metod si ve zkratce zopakujme problematiku časových řad. Klíčovým pojmem je časová řada. Časovou řadou budeme rozumět chronologicky uspořádanou posloupnost hodnot nějakého ukazatele. Můžeme rozlišovat, zda se jedná o hodnotu nějakého ukazatele:

- za interval - **intervalový ukazatel**, např. počet úmrtí za měsíc.
    1. Hodnota ukazatele závisí na délce intervalu
    2. Lze sčítat, průměrovat - má to smysl
    3. Obvykle je nutný přepočet na jednotkový interval, např. očištění kalendářních variací
- v určitém okamžiku - **okamžikový ukazatel**, např. počet zaměstnanců k prvnímu dni měsíce.
    1. Sčítání nemá smysl
    2. Průměrování pomocí *chronologického průměru.*

Dále nás můžou zajímat přímo hodnoty dané řady (absolutní ukazatele), nebo odvozené časové řady (ukazatele spočtené z přímých hodnot). Samozřejmě je možné dělit řady na krátkodobé a dlouhodobé (roční a delší období).

Pro analýzou časových řad je nástrojem první volby graf, např. grafy spojnicové, krabičkové(Box-Whisker plot), grafy jednotlivých složek časové řady.

Pro popis řady používáme popisné charakteristiky:

- Průměry - *aritmetický*/*chronologický*/*vážený chronologický*
- Charakteristiky variability - *rozptyl*, *směrodatná odchylka´*
- Míra dynamiky
    1. *absolutní přírůstek*, *průměrný absolutní přírůstek*
    2. *koeficient růstu* a *průměrný koeficient růstu* (geometrický průměr), *relativní přírůstek*
- Korelace dvou řad

Časové řadu y_t můžeme dekomponovat na dílčí složky:

1. Aditivní $y_t = T_t + S_t + C_t + \epsilon_t$
2. Multiplikativní $y_t = T_t \cdot S_t \cdot C_t \cdot \epsilon_t$,

kde T_t představuje trendovou složku, tj. dlohodobou tendenci vývoje, S_t je sezónní složka, která představuje kolíšání okolo trendu v rámci kalendářního roku, C_t je cyklická složka, což je, většinou nepravidelní, kolísání okolo trendu v období delší než rok a ϵ_t je náhodná složka.

Při modelování trendu nás zajímá, jakou trendovou funkci zvolit, jaká jsou kritéria volby:

- Interpolační: ME (průměrná chyba), MSE (rozptyl), MAE (průměrná absolutní chyba, index determinace R^2.
- Extrapolační, rozdělíme data na část trénovací a testovací. Kvalita modelu je určena pomocí interpolačních kritérií na testovacích datech.

Případně je pro test vhodnosti použitého regresního modelu možné použít testy vycházející z*autokorelace* reziduí, *Durbinovu-Watsonovu statistiku*, či reziduální *autokorelační funkci* (ACF) reziduí.

Sezónnost je třeba nejprve detekovat např. pomocí *periodogramů*, *autokorelační funkce*. Podobně můžeme postupovat i v případě modelování sezónní složky.

Pro další kapitoly si připomeňme pojem *stacionarity* časové řady. Představme si časovou řadu jako realizaci stochastického procesu, který má nějaké statistické charakteristiky (rozdělení). *Striktní stacionaritou* rozumíme, to že rozdělení daného procesu je nezávislé (invariantní) na posun v čase. Toto je poměrně silný předpoklad, proto se zavádí pojem *slabé stacionarity*, kdy požadujeme pouze konstantní střední hodnotu μ a rozpyl σ^2 a kovarianční funkce

$$\gamma(t, t-k) = E\left[(y_t - \mu_t)(y_{t-k} - \mu_{t-k})\right], \tag{1.1}$$

závisí pouze na posunu k, nikoliv na t.

CÍLE KAPITOLY

Věnujte pozornost zejména těmto oblastem:

- Zopakování vybraných pojmů a popisných charakteristik časových řad
- Dekompozice časových řad
- Modelování trendu řady
    1. Typy regresních funkcí (lineární, kvadratická, S, exponenciální) a jejich vlastnosti
    2. Metoda klouzavých průměrů a její varianty
    3. Exponenciální vyrovnávání
- Modelování sezónní složky
    1. Sezónní indexy
    2. Použití periodogramů. Fisherův test
    3. Holtovo-Wintersovo exponenciální vyrovnávání

KLÍČOVÁ SLOVA

časové řady, dekompozice, trend

ÚKOLY

1. Seznamte se grafy pro vizualizaci časových řad v knihovně matplotlib a seaborn. Zaměřte se na spojnicové, krabičkové grafy, histogramy. Pomocí těchto grafů visualisujte vybrané časové řady.
2. Prostudujte si použití knihovny pro dekompozici. Věnujte pozornost metodě *STL* a *Baxter-King* filtru. Proveďte dekompozici některých řad z časové řady.
3. Proveďte regresi pomocí různých trendových funkcí zde uvedeného datasetu. Za pomocí knihovny statsmodel určete pro dané modely Durbinovu-Watsonovu statistiku a visualisujte ACF, proveďte diskusi.
4. Pomocí metody zde aplikujte Holtovo-Wintersovo vyrovnávání na tento dataset.

OTÁZKY

1. Modelování trendu řady.
2. Vysvětlete princip vyrovnávání a exponenciálního vyrovnávání řady.
3. Vysvětlete princip periodogramů.

SHRNUTÍ

Po prostudováni byste měli být schopni:

- Orientovat se v základním názvosloví časových řad
- Určit základní charakteristiky řady.
- Modelovat trend a sezonalitu

ODKAZY NA LITERATURU

- Vše zde uvedené, vyjma popisných charakteristik, je pokryto v prvních sedmi kapitolách zde.(Obsahuje i názorné příklady v R)
- Srozumitelný úvod, včetně praktických příkladů, naleznete v prvních dvou kapitolách zde.

MÍSTO PRO VAŠE POZNÁMKY

2 Boxova-Jenkinsova metodologie I

Box-Jenkinsova metodologie využívá k modelování časových řad tzv. ARMA (*autoregressive moving average*) a ARIMA (*autoregressive integrated moving average*) procesů. Postup je zhruba následující

1. Výběr modelu, detekce stacinoarity a sezonality řady, určení řádu modelu
2. Nastavení parametrů modelu
3. Statistické ověření modelu

Zaměříme se nejprve na popis možných modelů a ne jejich výběr. Pro další předpokládáme, že daná časová řada y_t je stacionární. Ke zjištění, zda to je splněno lze použít následující testy:

- Grafické zhodnocení
- ACF graf (korelogram) - rychlý pokles s rostoucím lagem naznačuje stacionaritu
- Parametrické testy:
    1. Dickey-Fullerův test (DF -test), resp. rozšířený Dickey-Fullerův (ADF) test pro řady s trendem (popis je zde a použití v Pythonu zde).
    2. Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test (zde.

Připomeňme si, že autokorelační funkce ACF:

$$\rho(t, t-k) = \frac{E\left[(y_t - \mu_t)(y_{t-k} - \mu_{t-k})\right]}{\sqrt{\sigma_t^2}\sqrt{\sigma_{t-k}^2}}, \tag{2.1}$$

je sudou funkcí k a podává informaci o síle lineární závislosti mezi y_t a y_{t-k}, předpokládáme stacionárnost řady a proto uvažujeme závislost na k. V praxi používáme místo teoretické hodnoty $\rho(k)$ hodnoty získané z reálných pozorování. Získáme tak *výběrovou autokorelaci* r_k.

Protože korelovanost mezi y_t a y_{t-k} je ovlivněna hodnotami $y_{k+1}, \ldots, y_{t-k-1}$, zavádí se tzv. *parciální autokorelační funkce* (PACF), která je očištěna o tyto hodnoty. Pokud uvažujeme následující *autoregresi*

$$y_t = \Phi_{k1} y_{t-1} + \Phi_{k2} y_{t-2} + \ldots + \Phi_{kk} y_{t-k} + \epsilon_t, \tag{2.2}$$

kde ϵ_t je nekorelovaná s y_{t-l}, $l = 1, \ldots, k$. Pak Φ_{kk} vyjadřuje hodnoty parciální autokorelační funkce k-tého řádu. Její hodnoty určené z reálných dat, nazýváme *výběrovou parciální autokorelační funkcí* f_{kk} a je možné ji spočíst z r_k pomocí Durbinova rekurzivního vztahu.

V případě potřeby lze řadu *stacionarizovat* pomocí diferencí, viz dynamické vlastnosti řady řady a dále ARIMA model. Případně lze použít Boxovu-Jenkinsovu transformaci, či Boxovu-Coxovu transformaci kapitola 8.2. Druhá zmíněna transformace je implementována v Pythonu Boxova-Coxova transformace.

Vyjdeme-li z rovnice 2.2, pak autoregresní proces p-tého řádu je dán vztahem

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \epsilon_t, \tag{2.3}$$

ϵ_t má vlastnosti bílého šumu. Podrobnější popis AR procesů a vliv koeficientů ϕ na chování procesu je popsán zde a podrobněji kapitola 3.2.1. Modelování AR procesů v Pythonu je popsáno zde.

Další důležitý typ procesů jsou MA (moving average) procesy klouzavých průměrů.

$$y_t = \epsilon_t - \theta_1 \epsilon_{t-1} - \theta_2 \epsilon_{t-2} - \ldots - \theta_q \epsilon_{t-q}. \tag{2.4}$$

Opět je zde vliv koeficientů θ na chování procesu. Detailnější popis zde a podrobněji kapitola 3.2.2.

Složením výše uvedených typů procesů dostáváme ARMA(p,q) procesy. Jejich popis je kapitola 3.2.3, případně zde.Modelování ARMA procesů v Pythonu je popsáno zde a zde.

CÍLE KAPITOLY

Věnujte pozornost zejména těmto oblastem:

- Testy stacionarity
- ACF a PACF
- Procesy AR
- Procesy MA
- Procesy ARMA

KLÍČOVÁ SLOVA

lineární procesy, ARMA, stacionarita

ÚKOLY

1. Projděte si modelování kvality ovzduší v Madridu zde.
2. Projděte si modelování AR,AM, ARMA procesů v Pythonu, na odkazech uvedených výše, a zaměřte se i na používání předpovědí modelů (forecasting).
3. Namodelujte v Pythonu procesy AR(1) a AR(2) a diskutujte vliv parametrů ϕ na stacionaritu modelu, viz simulace v R zde.
4. Namodelujte v Pythonu procesy MA(1) a MA(2) a diskutujte vliv parametrů θ na model, viz simulace v R zde.

OTÁZKY

1. Jaký je vliv koeficientů ϕ na AR proces ?
2. Co to je stacionarita a invertibilita ARMA procesu ?

SHRNUTÍ

Po prostudováni byste měli být schopni:

- Určit zda je řada stacionární a případně si ji upravit.
- Modelovat řadu pomocí ARMA procesů a zároveň pomocí daných knihoven provádět predikce hodnot řady.

ODKAZY NA LITERATURU

- Vše zde uvedené naleznete zde a zde.
- Též zajímavé poznámky jsou uvedeny zde v kapitole 4 a 5.

MÍSTO PRO VAŠE POZNÁMKY

3 Boxova-Jenkinsova metodologie II

Zde se zaměříme na popis dalších modelů, konkrétně půjde o modely:

1. ARIMA (*autoregressive integrated moving average*)
2. SARIMA (*seasonal autoregressive integrated moving average*)
3. SARIMAX (*seasonal autoregressive integrated moving average with exogenous regressors*)

Modely ARIMA se používají, obsahuje-li časová řada trend a je-li převeditelná na stacionární pomocí diferencování. Postup je takový:

- Nejprve se řada stacionarizuje pomocí diferencí.
- Poté se aplikuje proce ARMA(p,q).

Protože je někdy nutné provádět pro stacionarizaci řady opakované diferencování, je nutné určit kolikrát se bude diferencovat (parametr d), je ARIMA model popsán parametry ARIMA(p,d,q). Pro postup určení řádu modelu (parametry p, d, q) se používá diferencování spolu s ACF a PACF grafy, postup je možné nalézt např. kapitola 5, či 3.5.1.

Pro použití ARIMA modelu v Pythonu je možné použít následující. Pokud jsme vystaveni před úkol automatizace nastavení parametrů ARIMA modelu pro velké množství dat, je možné použít Hyndmanův-Khandakarův algoritmus, který je typicky implementován v tzv. *autoarima* balíčku. V jazyce R už v době psaní tohoto textu (2020) rozšířený, v Pythonu méně, nicméně je možné použít třeba tuto implementaci v balíku Pyramid.

Pokud se v časové řadě navíc vyskytuje sezonalita, tj. je zde jak závislost mezi y_t, y_{t-1}, y_{t-2}, tak i mezi y_t, y_{t-s}, y_{t-2s}, kde s rozumíme sezónní periodu, tak se používá tzv. SARIMA model. Jde vlastně o složený model ARIMA viz výše, spolu s ARMA aplikovaným na sezónní diference. Tím dostáváme SARIMA$(p, d, q)(P, D, Q)_s$, kde p, d, q jsou parametry ARIMA a P, D, Q jsou parametry sezónní složky modelu(typicky multiplikativní). Podrobnější popis včetně matematického pozadí je uveden 3.4.1. Praktická ukázka je uvedena zde.

V současnosti umožňují softwarové balíky zároveň použít regresi časové řady pomocí exogenní proměnné spolu s ARIMA modelem, dostáváme tak modely typu ARIMAX, SARIMAX. Základní vysvětlení je možno nalézt zde.

Máme-li správně určen čád modelu, je třeba určit hodnoty jednotlivých parametrů θ a ϕ (v případě SARIMA i dalších). Zde se využívá různých variant metody nejmenších čtverců, viz např. kapitola 10.2. a zde.

Poté následuje otestování modelu:

- Testování autokorelací odhadnutého rezidua
- Použití *portmanteau testu* - Boxův-Piercův test

Popis výše uvedených metod verifikace modelu naleznete v kapitola 3.5.2 a kapitola 9.3.

CÍLE KAPITOLY

Cílem této kapitoly je získat všeobecný přehled funkce a konstrukce uvedených modelů a schopnost je používat. Menší důraz je kladen na jejich matematické vlastnosti.

KLÍČOVÁ SLOVA

ARIMA, SARIMA

ÚKOLY

1. Projděte si modelování pomocí (S)ARIMA(X) modelů zde.
2. Ověřte v pomocí nástrojů v Pythonu(R) některé modely uvedené v kapitola 3.5.4, použijte případně i autoarima.

OTÁZKY

1. Vysvětlete princip ARIMA a SARIMA modelu.
2. Co znamená vysoká p-hodnota Boxova-Piercova testu

OTÁZKY K ZAMYŠLENÍ

1. Vysvětlete, jak trend a sezonalita řady ovlivňuje její stacionaritu.

ODKAZY NA LITERATURU

- Dostatečný popis teorie ke zde uvedenému naleznete zde a zde.
- Dále praktické aplikace v R zde.

MÍSTO PRO VAŠE POZNÁMKY

4 Modely ARCH, GARCH

Pro modelování řad, kde nejsou splněny podmínky stacionarity (často výnosy akcií) je možné použít modely ARCH (*autoregressive conditional heteroskedasticity*) a GARCH(*generalized autoregressive conditional heteroskedasticity*). Výše uvedené modely se používají pro modelování volatilních procesů, tj. procesů, kde se mění jejich rozptyl v čase.

Volatilitou rozumíme podmíněný rozptyl, tj. rozptyl je dán pomocí tzv. *skedastická* funkce h_t. Pokud je rozptyl nezávislý na parametru, říkáme že řada je *homoskedastická*, pokud se mění, je nazývána *heteroskedastická*. Modely volatility je možné psát ve tvaru:

$$y_t = \epsilon_t \sqrt{h_t}, \tag{4.1}$$

kde ϵ_t má vlastnosti bílého šumu.

Modely ARCH předpokládají heteroskedasticitu, kde h_t závisí na y_{t-k}, kde $k = 1, \ldots$.

Příkladem může být model ARCH(1), kde $h_t = \alpha_0 + \alpha_1 y_{t-1}^2$, kde $\alpha_0 > 0$ a $\alpha_1 \geq 0$, což vede na $y_t = \epsilon_t \sqrt{\alpha_0 + \alpha_1 y_{t-1}^2}$. Zobecněním je ARCH(p) model, kde

$$h_t = \alpha_0 + \sum_{k=1}^{p} \alpha_k y_{t-k}^2 \tag{4.2}$$

opět jako v případě ARIMA mají koeficienty α vliv na chování modelu, zde pro jeho stacionaritu $\sum_{i=1}^{p} \alpha_i < 1$.

V praxi se ukazuje jako nutné používat velmi vysoký řád modelu (p). Toto odstraňuje tzv. GARCH model. Zde je funkce h_t brána ve tvaru

$$h_t = \alpha_0 + \sum_{k=1}^{p} \alpha_k y_{t-k}^2 + \sum_{k=1}^{q} \beta_k h_{t-k}. \tag{4.3}$$

Je tedy vidět, že GARCH model je dán prametry p,q, značíme GARCH(p,q), přičemž nejpoužívanější je model GARCH(1,1).

Další detaily ohledně vlivu parametrů na model, či jejich nalezení lze nalézt v zde, či zde. Implementace GARCH modelu v jazyce Python je zde.

CÍLE KAPITOLY

Cílem této kapitoly je získat všeobecný přehled funkce a konstrukce uvedených modelů a schopnost je používat (na úrovni sw. balíků). Výrazně menší důraz je kladen na jejich matematické vlastnosti.

KLÍČOVÁ SLOVA

ARCH, GARCH

ÚKOLY

1. Implementujte ARCH(1) model v Pythonu a diskutujte vliv parametrů modelu, tj. parametry jsou vstupem modelu a tedy nemusíte řešit jejich určování.
2. Implementujte GARCH(1,1) model v Pythonu a diskutujte vliv parametrů modelu, tj. parametry jsou vstupem modelu a tedy nemusíte řešit jejich určování.
3. Projděte si následující ukázku použití balíku ARCH zde.

ODKAZY NA LITERATURU

- Základní popis modelů lze nalézt zde.
- Popis principů přímo od autora ARCH modelu je možné nalézt zde.
- Dále praktické aplikace v R zde.

MÍSTO PRO VAŠE POZNÁMKY

5 Rekurentní neuronové sítě

Pro modelování časově souvztažných událostí se nehodí běžně používané dopředné (*feedforward*) sítě, neboť jejich strutkura neumožňuje zachytit onu souvztažnost. Příkladem oblastí kde je toto potřeba je zpracování jazyka, ale modelování časových řad.

Je ovšem možné používat rekurentní neuronové sítě (RNN). Rekurencí rozumíme to, že výstup dané jednotky sítě je zpětně přiveden s nějakou vahou na její vstup, formálně je to možné zapsat takto:

$$\boldsymbol{h}^{(t)} = \boldsymbol{f}(\boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t-1)}, \boldsymbol{\theta}), \tag{5.1}$$

kde říkáme, že stav skryté vrstvy sítě v čase t $h^{(t)}$ je dán vstupem $x^{(t)}$, stavem předchozím $h^{(t-1)}$ a konstantními parametry modelu θ. Při studiuu RNN je užitečné se na danou jednotku nekoukat rekurentně, ale jako na soubor jednotek v různých časech, této technice se říká *unrolling* a narazíte na ní ve většině zdrojů věnovaných RNN. Příkladem mohou být obrázky v následujícím textu zde, a s hlubším nadhledem a více do detailu zde.

Topologie jednotek může být různá, základní jsou následující:

- Propojení na úrovni skrytých vrstev sítě, kde výstupem je sekvence hodnot (*many-to-many*).
- Propojení na úrovni skrytých vrstev sítě, kde výstupem je jedna hodnota (*many-to-one*).
- Propojení výstupů sítě do skrytých vrstev, kde výstupem je sekvence hodnot.

Tato topologie sítě má vliv jak na její učení (volba algoritmu a jeho náročnost), tak i na její úspěšnost při řešení problémů. Sítě realizující propojení na úrovni skrytých vrstev jsou úspěšnější, jejich nevýhoda oproti poslednímu jmenovanému modelu je ovšem to, že jejich učení je časově náročnější (problematika paralelizace). Popis technik učení nalezenete 10.2.1-10.2.3.

Problémem který nastává při učení RNN, je díky propagaci vah při rekurzi, jistá nestabilita při jejich určování, toto se nazývá *vanishing/exploding gradient problem* , více viz zde. Tento jev vede k problémům, když je třeba zachytit závislosti s velkou rozlohou v čase (*long term dependencies*), řešením jsou tzv. *LSTM*(Long Short-Term Memory units) a *GRU*(Gated Recurrent Unit) sítě.

Na zde uvedené rekurentní sítě můžeme nazírat z pohledu časových řad tak, že na základě hodnot z minulosti, je predikována nová hodnota v budoucnosti. Nicméně v některých úlohách např. rozpoznávání řeči, je nutné znát celý kontext, tj. pro určení významu slova je nutné znát celý jeho kontext, tj. i "budoucí" hodnoty. K tomuto účelu se používají obousměrné rekurentní neuronové sítě, viz zde.

Pro Keras lze nalézt použití jednoduchých RNN v Pythonu zde. Pro *PyTorch* a *MXNet* je možné použít praktický tutoriál zde.

CÍLE KAPITOLY

Cílem této kapitoly je získat všeobecný přehled o rekurentních sítích (RNN):

- Topologoie sítí
- Princip učení a možné problémy
- Přehled aplikací těchto modelů

KLÍČOVÁ SLOVA

RNN sítě, učení

ÚKOLY

1. Prostudujte si následující odkaz zde a pokuste se implementovat vlastní jednoduchou RNN síť.
2. Prostudujte a vyzkoušejte ukázky kódu od autora knihovny Keras F. Cholleta pro práci s textovými daty a RNN, viz 6.1.a 6.2
3. Prostudujte si použití RNN na predikování hodnot časové řady zde.
4. Vyzkoušejte si tento přístup k RNN pro určení jazykové oblasti dle jména zde.

ODKAZY NA LITERATURU

- Základní studijní literaturu najdete zde
- Přehled s praktickými aplikacemi a zdrojovými kódy je k dispozici zde.
- Dále velice pěkné a názorné shrnutí je uvedeno zde.

MÍSTO PRO VAŠE POZNÁMKY

6 LSTM, GRU a jejich použití

Jak již bylo řečeno, klasické RNN sítě mají problém se stabilitou gradientu v průběhu učení. Tento nedostatek byl odstraněn za použití moderních sítí typu GRU a LSTM. Ačkoliv LSTM historicky předcházelo sítím GRU, začněme s GRU, protože jsou zjednodušením LSTM sítí.

V modelu GRU (Gated Recurrent Units) se používají tzv. brány (gates) pro zesílení (*update gate*), či zeslabení (*reset gate*) při přenosu současného stavu $h^{(t-1)}$ do budoucí úrovně. Tyto brány jsou propojeny se vstupem a výstupem, viz např. princip zde. Jednotlivé brány jsou implementovány jako plně spojené vrstvy se sigmoidální, případně *tgh* funkcí a mají tedy váhy. Při učení dochází k výpočtu těchto vah a síť je díky tomu schopná zachytit to, zda nějaké znaky v závislosti na pozici jsou, či nejsou důležité. Detaily propojení, včetně možné ukázkové implementace jsou k dispozici zde.

V modelu LSTM (Long Short Term Memory) je použito bran více, je zde opět brána zodpovídající za oslabování předchozího stavu (*forget gate*), brána přinášející informace o důležitosti vstupu (*input gate*), dále je tu další brána zajišťující paměť jednotky (*candidate memory*) a brána výstupu. Přehledné schéma fungování jednotlivých bran, včetně možné implementace je zde.

Model LSTM je komplikovanější vůči GRU modelu, což je způsobeno větším počtem vah k učení. Ve většině Pythonovských frameworků jsou oba modely dostupné, viz Keras, či PyTorch.

Použití těchto modelů je velmi široké, ukažme si například predikci hodnot časové řady y_t. Hodnoty řady použijeme jako vstup sítě. Konkrétně budeme používat přístup *many-to-one* a budeme predikovat z l předchozích hodnot řady hodnotu $l + 1$. Hodnoty časové řady y_t rozdělíme na vstupní vektory pevné délky, např. $x_0 = (y_0, y_1, .., y_l)$, $x_1 = (y_1, y_2, .., y_{l+1})$ atd. Těmto vstupním vektorům jsou přiřazeny očekávané výstupy sítě y_{l+1}, y_{l+} atd. Jako výstupní vrstva sítě je použita hustá vrstva s lineární přechodovou funkcí. Poznamenejme, že pro praktické použití v oblasti ekonomických dat je třeba časovou řadu standardizovat (např. převod na standardní měsí), dále je nutné vstupní data škálovat (použití *scalerů*, např. MinMax). Nad predikovanými hodnotami ju nutné provádět obrácený postup.

CÍLE KAPITOLY

Cílem této kapitoly je získat všeobecný přehled o možném použití LSTM a GRU sítí.

- LSTM a GRU
- Aplikace pro predikci
- Základní přehled aplikací těchto modelů

KLÍČOVÁ SLOVA

GRU, LSTM

ÚKOLY

1. Prostudujte si následující odkaz zde pro predikci chování časové řady. Vyzkoušejte různé architektury sítě.
2. Prostudujte a vyzkoušejte ukázky kódu od autora knihovny Keras F. Cholleta pro práci s časovou řadu 6.3.
3. Vyzkoušejte použití obousměrné RNN v Kerasu pro analýzu sentimentu zde. Pokud Vás zajímá, jak předzpracován text vstupující do RNN (*embedding* vrstva), tak to je popsáno v závěrečné kapitole.
4. V příkladu pro určení jazykové oblasti dle jména (viz předchozí kapitola) zde použijte LSTM a GRU a diskutujte vliv na klasifikaci a učení modelu.

ODKAZY NA LITERATURU

- Přehled s praktickými aplikacemi a zdrojovými kódy je k dispozici zde.
- Popis LSTM v základní studijní literatuře najdete kapitola 10.10.

MÍSTO PRO VAŠE POZNÁMKY

7 Model autoencoder jeho použití

Modely zde uváděné mají použití v oblasti strojového překladu, zpracování přirozené řeči (NLP), odstraňování šumu, či detekce anomálií.

Nejprve se podívejme architektonický model neuronových sítí *encoder-decoder*. Zde se v podstatě jedná od dvě propojené neuronové sítě. První z nich (encoder) přijímá data na vstupu a jeho cílem je data zakódovat. Toto kódování je dáno jeho skrytým stavem h. Výstup encoderu nás obvykle nezajímá. Do decoderu je předán (sdílí) finální vnitřní stav h, což představuje vlastně zakódovaný vstup. Výstup decoderu se bere jako výstup modelu.

Poznamenejme, že pro encoder a decoder se často používají LSTM/GRU jednotky a vnitřním stavem se rozumí v tomto případě váhy jednotlivých bran jednotek. Mechanismus je takový, že máme k dispozici vstupní vektor x a máme k dispozici nějakou jeho transformaci y, oba vektory mohou mít rozdílnou délku, jedná se tedy o tzv. *seq to seq* problém. Můžeme si to představit, že vektor $\mathbf{x}$ je kódovaná sekvence[1] a y je nějaká jiná, transformovaná sekvence[2]. Výsledný model tedy učíme tak, že na vstupu modelu je x a požadovaným výstupem modelu (výstupem decoderu) je y. Triviální, nicméně velmi názorná ukázka je uvedena zde. Ukázkový příklad použití pro překlad z angličtiny do francouštiny v Kerasu je uveden zde, pro lepší pochopení využijte následujícího blogu autora knihovny Keras blog.

Model *autoencoder* je speciálním případem výše uvedeného modelu encoder-decoder. Základní rozdíly jsou v tom, že vstupní vektor $\mathbf{x}$ a výstupní vektor $\mathbf{y}$ jsou identické a nedochází ke sdílení vnitřních stavů h. Jak encoder, tak decoder mohou být plně pospojované vrtsvy, tak LSTM/GRU jednotky, v závislosti na použití. Z výše uvedeného je zřejmé, že se jedná o učení bez učitele.

Použití je poměrně široké, zaměřme se jen na náplň tohoto kurzu:

1. Odšumování dat - denoising autoencoder (DAE)
2. Detekce anomálií
3. Zpracování přirozené řeči (viz další kapitoly)

Použití autoenkodérů v Kerasu je popsáno zde. Zaměřte se na obecný princip konstrukce a na použití DAE Další poměrně zajímavé použití autoencoderů je pro redukci dimensionality jako varianta PCA.

[1] Například věta v jednom jazyce.
[2] Například věta v druhém jazyce.

CÍLE KAPITOLY

Cílem této kapitoly je získat všeobecný přehled o použití neuronových sítí v kontextu práce se sekvencemi

- Model encoder-decoder
- Model autoencoder - obecný princip a některá použití

KLÍČOVÁ SLOVA

encoder, decoder, autoencoder, DAE

ÚKOLY

1. Prostudujte si již výše uvedený odkaz zde a zaměřte se na odstranění šumu z obrázků. Aplikujte uvedený postup na odstranění šumu za časové řady, tj. využijte některou z již dříve použitých časových řad, naučte DAE autoencoder a poté zkoumejte jakou míru šumu je schopen potlačit. Použijte různé intensity bílého šumu.
2. Prostudujte a vyzkoušejte postup pro detekci anomálií v časové řadě uvedený zde, nebo zde.
3. Prostudujte se detekci podvodů pomocí autoencoderu v následující ukázce zde.

ODKAZY NA LITERATURU

- V základní studijní literatuře najdete též popis modelu encoder-decoder 10.4 a 10.2.4 a podrobný popis modelu autoencoder kap. 14.
- Modelování seq to seq pomocí MXNet zde.

MÍSTO PRO VAŠE POZNÁMKY

8 Model Attention

Modely typu Attention představují rozšíření modelu encoder decoder. Myšlenka je taková, že v kontextu zpracování (seq to seq) nějaké věty, jsou některá slova důležitější a některá méně. Příkladem budiž věta "Večer si koupím chlazené pivo.", kde slovo **koupit** je silněji svázano se slovem **pivo**, než se slovem chlazené (důležité je, že si něco kupuji). Tato informace, že si něco kupuji je důležitá pro překlad dané věty. Tato důležitost je vyjádřena vahou daného spojení.

Konkrétněji, původní model Attention je model typu encoder decoder, kde encoder je obousměrná rekurentní síť. Na rozdíl od původního modelu se nepředává do decoderu jen finální skrytý stav h_n, nýbrž vážený součet jednotlivých skrytých stavů h_i, $i = 1, \ldots, n$, kde n je délka vstupní sekvence. Za určení dílčích vah těchto stavů je zodpovědná vložená plně spojená mezivrtsva (MLP). Originální článek je uveden zde. Přehled různých technik a vylepšení modelu Attention je uveden zde.

Mechanismu Attention je využito i u modelu Transformer, což je opět model typu encoder decoder, ale je odstraněna rekurentní část, jelikož u RNN je obtížné paralelní učení, viz zde.

Protože použití modelu Attention v Kerasu (ukázkové kódy) bylo v době psaní tohoto textu ještě ve stádiu vývoje, zaměříme praktické úkoly na knihovnu PyTorch.

CÍLE KAPITOLY

Cílem této kapitoly je získat všeobecný přehled o použití moderních neuronových sítí v kontextu práce se sekvencemi

- Attention model
- Transformer model

KLÍČOVÁ SLOVA

encoder, decoder, Attention, Transformer

ÚKOLY

Projděte si následující tutoriály pro seq to seq za pomocí knihovny PyTorch, postupujte od 1 do 4. Pokud Vám nebude jasný tzv. *embeding*, tak podrobnosti jsou uvedeny v následující kapitole. Jedná se o převod slov do vektorů, protože neuronová síť potřebuje pracovat přímo se slovy.

1. Seq to seq na úrovni znaků (viz úkol č.4 v kapitole 5) zde.
2. Generování sekvencí zde.
3. Použití mechanismu attention pro překlad zde.
4. Použití modelu Transformer zde.

OTÁZKY

1. Co je principem přístupu attention ?
2. Jaký je rozdíl mezi RNN a modelem Transformer ?

ODKAZY NA LITERATURU

- Mechanismus attention v MXNet/PyTorch včetně ukázek implementace zde.
- Praktický popis použití Kerasu pro NLP zde.

MÍSTO PRO VAŠE POZNÁMKY

9 Zpracování přirozeného jazyka pomocí strojového učení

Zde si shrneme zpracování přirozeného jazyka pomocí metod strojového učení, z hlediska sylabu se jedná o devátou a desátou výukovou lekci.

Obvykle je nutné nejprve text očistit přehled nástrojů, na což je výhodné použít knihovny NLTK. Dále, je nutné jej obvykle vektorizovat, tj. model nepracuje přímo s elementy jazyka, ale jejich vektorovou reprezentací. Poznamenejme, že se někdy může jednat o velmi dimenzionální vektory ($n \approx 10^2$ a i více), což má za následek poměrně vysokou výpočetní náročnost. Pro vektorizaci dokumentů se používá algoritmus *Tf-Idf* (Term frequency–Inverse document frequency), popis algoritmu je uveden např. v kapitola 6.. Tento algoritmus je implementován v knihovně scikit-learn TfidfVectorizer. Ukázku použití dalších vektorizerů v Pythonu a scikit-learn zde.

Pro vektorizaci slov je možné použít tzv. *one-hot encoding*, tj. dojde k vytvoření slovníku slov a každé slovo je kódováno vektorem nul, mimo pozice odpovídající danému slovu, viz např. zde. Toto vede k vysocedimenzionálním (velký slovník) řídkým vektorům. Další nevýhoda tohoto přístupu je v tom, že po vektorové reprezentaci požadujeme, aby slova nějakým způsobem podobná ležela blízko sebe i po vektorizaci, což tento způsob nesplňuje. Blízkost vektorů se obvykle měří pomocí *cosinové podobnosti*, viz např. zde a zde.

Pro odstranění výše uvedených nevýhod se používá tzv. *word embedding* algoritmů typu *word2vec* a *glove*, viz dále. Algoritmus word2vec používá model skip-gramu (spojitý) a model CBOW (continuos bag-of-words). Model skip-gramu vektorizuje slova tak, že pro každé slovo daného korpusu je naučena neuronová síť s výstupní softmax vrstvou, kde vzory vstupu x jsou daná slova (desetitisíce) v one-hot encoding a vzory výstupu y jsou slova (opět kódováno pomocí one-hot encoding) ležící v nějakém okně obsahující vstup x. Tyto učební páry jsou automaticky generovány z korpusu. Síť je naučena, ale jako vektor charakterizující dané slovo, se používají váhy sítě vztažené k danému slovu, kterých je mnohem menší počet (stovky), dochází tak k redukci dimenzionality V modelu CBOW je pouze prohozena role vstupů x a výstupů y. Ilustrativně je to ukázáno zde.

Protože se jedná o výpočetně náročný problém, je třeba optimalizovat výpočet vah. V zásadě se dělá to, že se snižuje počet učebních příkladů pro některá slova tzv. *subsampling*, typicky o slova která jsou často v kombinaci s ostaními, např. člen 'the' v angličtině. Další technikou je tzv. *negative sampling*, kdy jsou modifikovány jen některé váhy modelu. Poslední optimalizací je spojování častých sousloví do jednoho slova. Detaily lze nalézt přímo v originálním článku zde a zde. Jako didakticky poučné shledávám tyto zdroje obecně a detaily.

CÍLE KAPITOLY

Cílem této kapitoly je získat přehled o použití strojového učení při zpracování přirozeného jazyka.

- Vektorizace dokumentů
- CBOW
- Skipgram
- word2vec a jeho použití

KLÍČOVÁ SLOVA

kosinová podobnost, word embedding, CBOW, skipgram, word2vec

ÚKOLY

1. Projděte si tutoriály zaměření na zpracování textu pro TensorFlow(Keras) zde.
2. Seznamte se s balíkem *gensim* a projděte si uváděné tutoriály zde.

OTÁZKY

1. Jaký je vliv korpusu na vektorizaci ?
2. Proč se pro měření podobnosti dvou vektorů používá cosinová podobnost a ne euklidovská vzdálenost dvou vektorů?

ODKAZY NA LITERATURU

Zde je uvedeno nejdůležitější, ostatní odkazy máte v textu.

- Word2vec obecně a detaily.
- Tutoriály k gensim zde.

MÍSTO PRO VAŠE POZNÁMKY

Literatura

[1] ARLT, Josef, Markéta ARLTOVÁ a Eva RUBLÍKOVÁ. *Analýza ekonomických časových řad s příklady.* Vyd. 2. Praha: Oeconomica, 2004. ISBN 80-245-0777-3. zde

[2] Statistical forecasting: notes on regression and time series analysis [online]. Fuqua School of Business, Duke University, 2019 [cit. 2020-02-26]. Dostupné zde

[3] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. Deep learning. Cambridge, Massachusetts: The MIT Press, [2016]. Adaptive computation and machine learning. ISBN 978-0-262-03561-3. Dostupné zde

[4] Tutorialy ke Kerasu zde.

[5] Tutorialy k PyTorch zde.